Дэн Саймон

Алгоритмы эволюционной оптимизации

ЛАНЬ®

EVOLUTIONARY OPTIMIZATION ALGORITHMS

Biologically-Inspired and Population-Based Approaches to Computer Intelligence

Dan Simon
Cleveland State University

АЛГОРИТМЫ ЭВОЛЮЦИОННОЙ ОПТИМИЗАЦИИ

Биологически обусловленные и популяционно-ориентированные подходы к компьютерному интеллекту

Дэн Саймон

Перевод с английского Логунов А. В.

Москва, 2020

УДК 004.421
ББК 32.811
С12

С12 Дэн Саймон

Алгоритмы эволюционной оптимизации / пер. с англ. А. В. Логунова. – М.: ДМК Пресс, 2020. – 1002 с.: ил.

ISBN 978-5-97060-707-7

В данной книге рассматриваются история, теоретические основы, математический аппарат и программирование алгоритмов эволюционной оптимизации. Рассмотренные алгоритмы включают в себя генетические алгоритмы, генетическое программирование, оптимизацию на основе муравьиной кучи, оптимизацию на основе роя частиц, дифференциальную эволюцию, биогеографическую оптимизацию и многие другие.

Издание будет полезно студентам старших курсов, программистам, инженерам, а также всем специалистам, кто интересуется принципами построения различных алгоритмов.

УДК 004.421
ББК 32.811

ISBN 978-0-470-93741-9 (англ.)
ISBN 978-5-97060-707-7 (рус.)

Оглавление

ЛАНЬ®

Благодарности

Хотелось бы поблагодарить всех тех, кто оказывал финансовую поддержку моих исследований по эволюционной оптимизации два десятилетия, в течение которых я участвую в этой увлекательной области исследования: Хоссни Эль-Шарифа из группы по системной интеграции компании TRW, Дорин Драготониу из подразделения по системам безопасности на транспорте компании TRW, Санджай Гарга и Дональда Саймона из подразделения управления и динамики имени Джона Гленна в НАСА, Димитра Филева из Форд Мотор Компани, Брайана Дэвиса и Уильяма Смита из Кливленд Клиник, Национальный научный фонд и государственный университет Кливленда. Хотел бы также поблагодарить студентов и коллег, которые работали со мной и публиковали статьи по проблематике эволюционных алгоритмов: Джеффа Абеля, Тавой Ду, Мехмета Эргецера, Брент Гарднер, Бориса Игельника, Пола Лозового, Хайпинг Ма, Берни Монтавона, Мирелу Оврейю, Рик Рарика, Ганса Рихтера, Дэвида Сейди, Сергея Саморезова, Нину Шайдеггер, Арпита Шаха, Стива Затмари, Джорджа Томаса, Оливера Тибра, Тона ван ден Богерта, Аруна Венкатесана и Тима Уилмота. Наконец, хотел бы поблагодарить всех тех, кто читал предварительные проекты этого материала и сделал ряд полезных предложений для его улучшения: Эмиль Аартс, Дэна Эшлока, Форреста Беннета, Ханса-Георга Байера, Мориса Клера, Карлоса Коэльо Коэльо, Калянмоя Деба, Гаца Эйбена, Цзинь Хао Као, Яосю Джин, Педро Ларранага, Махамеда Омрана, Кеннета В. Прайса, Ханса-Пола Швефеля, Томаса Штюцле, Хамида Тизуша, Даррелла Уитли, а также трех анонимных рецензентов оригинального предложения данной книги. Эти рецензенты не обязательно одобряют данную книгу, но она стала намного лучше благодаря их предложениям и комментариям.

Дэн Саймон

Предисловие от издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте **www.dmkpress.com**, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу **dmkpress@gmail.com**, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу **http://dmkpress.com/authors/publish_book/** или напишите в издательство по адресу **dmkpress@gmail.com**.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу **dmkpress@gmail.com**, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Wiley очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу электронной почты **dmkpress@gmail.com** со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нампредоставлять вам качественные материалы.

Аббревиатуры

ABC	artificial bee colony	искуственная пчелиная семья
ACM	adaptive cultural model	адаптивная культурная модель
ACO	ant colony optimization	оптимизация на основе муравьиной кучи
ACR	acronym	аббревиатура
ACS	ant colony system	система муравьиной кучи
ADF	automatically defined function	автоматически определяемая функция
AFSA	artificial fish swarm algorithm	алгоритм по методу искусственного косяка рыб
ANTS	approximated non-deterministic tree search	аппроксимированный недетерминированный древовидный поиск
AS	ant system	муравьиная система
ASCHEA	adaptive segregational constraint handling evolutionary algorithm	эволюционный алгоритм с обработкой адаптивных сегрегационных ограничений
BBO	biogeography-based optimization	биогеографическая оптимизация
BFOA	bacterial foraging optimization algorithm	алгоритм оптимизации на основе бактериальной кормодобычи
BMDA	bivariate marginal distribution algorithm	алгоритм двумерного маржинального распределения
BOA	Bayesian optimization algorithm	алгоритм байесовой оптимизации
CA	cultural algorithm	культурный алгоритм
CAEP	cultural algorithm-influenced evolutionary programming	эволюционное программирование в условиях культурного алгоритма
CE	cross entropy	перекрестная энтропия
CEC	Congress on Evolutionary Computation	Конгресс по эволюционным вычислениям
CDF	cumulative distribution function	кумулятивная функция распределения
cGA	compact genetic algorithm	компактный генетический алгоритм

CMA-ES	covariance matrix adaptation evolution strategy	ковариационно-матричная адаптивная эволюционная стратегия
CMSA-ES	covariance matrix self-adaptive evolution strategy	ковариационно-матричная самоадаптивная эволюционная стратегия
COMIT	combining optimizers with mutual information trees	объединение оптимизаторов с деревьями взаимной информации
CX	cycle crossover	цикличное скрещивание
DACE	design and analysis of computer experiments	проектирование и анализ компьютерных экспериментов
DAFHEA	dynamic approximate fitness based hybrid evolutionary algorithm	гибридный эвалюционный алгоритм на основе динамической приближенной приспособленности
DE	differential evolution	дифференциальная эволюция
DEMO	diversity evolutionary multi-objective optimizer	диверсифицирующий эволюционный многокритериальный оптимизатор
ϵ-MOEA	ϵ-based multi-objective evolutionary algorithm	ϵ-ориентированный многокритериальный эволюционный алгоритм
EA	evolutionary algorithm	эволюционный алгоритм
EBNA	estimation of Bayesian networks algorithm	алгоритм оценивания байесовой сети
ECGA	extended compact genetic algorithm	расширенный компактный генетический алгоритм
EDA	estimation of distribution algorithm	алгоритм оценивания вероятностного распределения
EGNA	estimation of Gaussian network algorithm	алгоритм оценивания гауссовой сети
EMNA	estimation of multivariate normal algorithm	алгоритм оценивания многомерных нормальных величин
EP	evolutionary programming	эволюционное программирование
ES	evolution strategy	эволюционная стратегия
FDA	factorized distribution algorithm	алгоритм факторизованного распределения
FIPS	fully informed particle swarm	частично информированный рой частиц

FSM	finite state machine	конечный автомат, или машина с конечным числом состояний
GA	genetic algorithm	генетический алгоритм
GENOCOP	genetic algorithm for numerical optimization of constrained problems	генетический алгоритм для численной оптимизации ограниченных задач
GOM	generalized other model	обобщенная другая модель
GP	genetic programming, or genetic program	генетическое программирование или генетическая программа
GSA	gravitational search algorithm	гравитационный поисковый алгоритм
GSO	group search optimizer	оптимизатор на основе группового поиска
GSO	Glowworm search optimization	светлячковая поисковая оптимизация
HBA	honey bee algorithm	алгоритм на основе медоносной пчелы
hBOA	hierarchical Bayesian optimization algorithm	иерархический байесовский оптимизационный алгоритм
HCwL	hill climbing with learning	восхождение к вершине холма с самообучением
HLGA	Hajela-Lin genetic algorithm	генетический алгоритм Хаджела-Лин
HS	harmony search	поиск гармонии
HSI	habitat suitability index	индекс пригодности естественной среды обитания
IDEA	iterated density estimation algorithm	алгоритм итерированного оценивания плотности вероятности
IDEA	infeasibility driven evolutionary algorithm	эволюционный алгоритм, управляемый невыполнимостью
IUMDA	incremental univariate marginal distribution algorithm	инкрементный алгоритм одномерного маржинального распределения
MMAS	max-min ant system	минимаксная муравьиная система
MMES	multimembered evolution strategy	многочленная эволюционная стратегия

MIMIC	mutual information maximization for input clustering	максимизация взаимной информации для кластеризации входных данных
MOBBO	multi-objective biogeography-based optimization	многокритериальная биогеографическая оптимизация
MOEA	multi-objective evolutionary algorithm	многокритериальный эволюционный алгоритм
MOGA	multi-objective genetic algorithm	многокритериальный генетический алгоритм
MOP	multi-objective optimization problem	задача многокритериальной оптимизации
MPM	marginal product model	модель маржинального продукта
$N(\mu, \sigma^2)$	normal PDF with a mean μ and a variance σ^2	нормальная функция плотности вероятности со средним значением и дисперсией
$N(\mu, \sum)$	multidimensional normal PDF with mean μ and covariance $\sum$	многомерная нормальная функция плотности вероятности со средним значением и ковариацией
NFL	no free lunch	никаких бесплатных обедов
NPBBO	niched Pareto biogeography-based optimization	алгоритм биогеографической оптимизации с нишированием по Парето
NPGA	niched Pareto genetic algorithm	генетический алгоритм с нишированием по Парето
NPSO	negative reinforcement particle swarm optimization	оптимизация на основе роя частиц с отрицательным подкреплением
NSBBO	nondominated sorting biogeography-based optimization	алгоритм биогеографической оптимизации с сортировкой по уровню недоминирования
NSGA	nondominated sorting genetic algorithm	генетический алгоритм с сортировкой по уровню недоминирования
OBBO	oppositional biogeography-based optimization	алгоритм оппозиционно-биогеографической оптимизации
OBL	opposition-based learning	оппозиционное самообучение
OBX	order-based crossover	реорганизующее скрещивание
OX	order crossover	упорядоченное скрещивание

PAES	Pareto archived evolution strategy	эволюционная стратегия с архивированием по Парето
PBIL	population based incremental learning	популяционное инкрементное самообучение
PDF	probability density function	функция плотности вероятности
PID	proportional integral derivative	пропорциональная интегральная производная
PMBGA	probabilistic model-building genetic algorithm	генетический алгоритм с построением вероятностной модели
PMX	partially matched crossover	скрещивание с частичным совпадением
PSO	particle swarm optimization	оптимизация на основе роя частиц
QED	quod erat demonstrandum: «that which was to be demonstrated»	что и требовалось доказать
RMS	root mean square	средний корень квадратный; среднеквадратическая ошибка
RV	random variable	случайная величина
SA	simulated annealing	симулированное закаливание
SBX	simulated binary crossover	симулированное бинарное скрещивание
SAPF	self-adaptive penalty function	самоадаптивная штрафная функция
SCE	shuffled complex evolution	перемешанная комплексная эволюция
SEMO	simple evolutionary multi-objective optimizer	простой эволюционный многокритериальный оптимизатор
SFLA	shuffled frog leaping algorithm	алгоритм перемешанных лягушачьих прыжков
SGA	stochastic gradient ascent	стохастический градиентный подъем
SHCLVND	stochastic hill climbing with learning by vectors of normal distributions	стохастическое восхождение к вершине холма с самообучением на основе векторов нормальных распределений
SIV	suitability index variable	переменная индекса приспособленности

SPBBO	strength Pareto biogeography-based optimization	биогеографическая оптимизация и силы Парето
SPEA	strength Pareto evolutionary algorithm	эволюционный алгоритм с расчетом силы Парето
TLBO	teaching-learning-based optimization	оптимизация по принципу учитель–ученик
TS	tabu search	поиск с запретами; табу-поиск
TSP	traveling salesman problem	задача коммивояжера
$U[a, b]$	uniform PDF that is nonzero on the domain [a, b]. This may refer to a continuous PDF or a discrete PDF depending on the context.	равномерная плотность вероятности, которая не является нулевой в области $[a, b]$. В зависимости от контекста может относиться к непрерывной или дискретной плотности вероятности
UMDA	univariate marginal distribution algorithm	алгоритм одномерного маржинального распределения
UMDA_c^G	continuous Gaussian univariate marginal distribution algorithm	алгоритм непрерывного гауссова одномерного маржинального распределения
VEBBO	vector evaluated biogeography-based optimization	алгоритм биогеографической оптимизации с векторным оцениванием
VEGA	vector evaluated genetic algorithm	генетический алгоритм с векторным оцениванием

Часть I

Введение в эволюционную оптимизацию

Глава 1

Введение

И подлинно: спроси у скота, и научит тебя, у птицы небесной, и возвестит тебе; или побеседуй с землею, и наставит тебя, и скажут тебе рыбы морские.

– Иов 12:7-9

В настоящей книге обсуждаются подходы к решению оптимизационных задач. В частности, мы[1] обсуждаем эволюционные алгоритмы (evolutionary algorithms, EA) для оптимизации. Несмотря на то что данная книга содержит немного математической теории, эту книгу не следует рассматривать как пособие по математике. Это скорее пособие по инженерному делу или прикладной информатике. Рассматриваемые в настоящей книге подходы к оптимизации приводятся с целью их потенциальной реализации в программном обеспечении. Задача настоящей книги – представить алгоритмы эволюционной оптимизации самым ясным, но строгим образом, а также предоставить достаточный объем продвинутого материала и ссылок, с тем чтобы читатель был готов внести новый материал в современное состояние дел.

Обзор главы

Эта глава начинается в разделе 1.1 с краткого обзора математических обозначений, которые мы используем в данной книге. Читателю также, возможно, будет полезен список аббревиатур, начинающихся со

[1] В этой книге используется общепринятая практика обращения к третьему лицу с помощью местоимения «мы». Иногда в книге используются местоимения «мы» для ссылок на читателя и автора. В иных случаях в книге используется местоимение «мы», для того чтобы показать, что в ней говорится от имени всего состава преподавателей и исследователей в области эволюционных алгоритмов и оптимизации. Разница должна быть ясна из контекста. Не придавайте слишком большое значение применению местоимения «мы»; это вопрос стиля изложения, а не претензии на авторитет.

страницы 21. В разделе 1.2 приведены некоторые причины, по которым я решил написать эту книгу об эволюционных алгоритмах, чего я надеюсь достичь с ее помощью и почему я думаю, что она отличается от всех других имеющихся превосходных книг по эволюционным алгоритмам. В разделе 1.3 рассматриваются предварительные условия, которым, как мы надеемся, читатели данной книги отвечают. В разделе 1.4 обсуждается общая организация домашних заданий, приводимых в данной книге, и наличие справочника с решениями задач. В разделе 1.5 резюмированы математические обозначения, которые мы используем в этой книге. Читателю рекомендуется регулярно обращаться к этому разделу при появлении незнакомых обозначений, а также самостоятельно применять их в домашних заданиях и в собственных исследованиях. В разделе 1.6 дается подробный план изложения в книге. Данный раздел переходит в раздел 1.7, в котором приводятся важные указания инструктору относительно некоторых приемов преподавания курса на основе этой книги. Данный раздел также дает преподавателю некоторые советы о том, какие главы являются более важными, чем другие.

1.1. Терминология

Некоторые авторы для обозначения эволюционных алгоритмов используют термин *эволюционные вычисления*. Этим подчеркивается тот факт, что эволюционные алгоритмы реализуются в компьютерах. Однако эволюционные вычисления могут относиться к алгоритмам, которые не используются для оптимизации; например, первые генетические алгоритмы не использовались для оптимизации как таковой, а были предназначены для изучения процесса естественного отбора (см. главу 3). Данная книга нацелена на алгоритмы эволюционной оптимизации, которые являются более специализированными, чем эволюционные вычисления.

Другие авторы для обозначения эволюционных алгоритмов используют термин *популяционно-ориентированная оптимизация*. Этим подчеркивается тот факт, что эволюционные алгоритмы обычно состоят из популяции кандидатных решений некоторой задачи, и по прошествии времени популяция эволюционирует к более качественному решению задачи. Однако многие эволюционные алгоритмы на каждой итерации могут состоять только из одного кандидатного решения (например, эволюционные стратегии и восхождения к вершине холма). Эволюционные алгоритмы являются более общими, чем популяционно-ориен-

тированная оптимизация, потому что эволюционные алгоритмы включают в себя алгоритмы с одной особью.

Отдельные авторы для обозначения эволюционных алгоритмов используют термин «компьютерный интеллект» или «вычислительный интеллект». Это часто делается для того, чтобы отличить эволюционные алгоритмы от экспертных систем, которые традиционно называются искусственным интеллектом. Экспертные системы моделируют дедуктивные рассуждения, в то время как эволюционные алгоритмы моделируют индуктивные рассуждения. Однако иногда эволюционные алгоритмы рассматриваются как разновидность искусственного интеллекта. Компьютерный интеллект – это более общий термин, чем эволюционный алгоритм, и включает в себя такие технологии, как нейронные сети, нечеткие системы и искусственная жизнь. Эти технологии могут использоваться для приложений, которые не являются оптимизационными. Поэтому, в зависимости от той или иной точки зрения, эволюционные алгоритмы вполне могут быть более общими или более конкретными, чем компьютерный интеллект.

Мягкие вычисления – это еще один термин, который связан с эволюционными алгоритмами. Мягкие вычисления противопоставляются жестким вычислениям. Жесткие вычисления относятся к точным, конкретным, численно строгим вычислениям. Под «мягкими» вычислениями понимаются менее точные вычисления, например те, которые люди выполняют в повседневной жизни. Алгоритмы мягких вычислений вычисляют, как правило, хорошие (но неточные) решения сложных, шумных, мультимодальных и многокритериальных задач. Поэтому эволюционные алгоритмы являются подмножеством мягких вычислений.

Другие авторы для обозначения эволюционных алгоритмов используют такие термины, как природообусловленные вычисления или биообусловленные вычисления. Вместе с тем некоторые эволюционные алгоритмы, такие как дифференциальная эволюция и алгоритмы оценивания вероятностных распределений, возможно, не являются природообусловлеными. Другие эволюционные алгоритмы, такие как эволюционные стратегии и оппозиционное самообучение, имеют очень слабую связь с естественными процессами. Эволюционные алгоритмы являются более общими, чем природообусловленные алгоритмы, потому что эволюционные алгоритмы включают в себя не биологически обусловленные алгоритмы.

Еще одним часто используемым термином для обозначения эволюционных алгоритмов является машинное (само)обучение. Машинное (само)обучение – это область исследования компьютерных алгоритмов,

которые автоматически обучаются на опыте. Однако эта область часто включает в себя множество алгоритмов, отличных от эволюционных алгоритмов. Машинное (само)обучение обычно считается более широкой областью, чем эволюционные алгоритмы, и включает в себя такие подобласти, как самообучение с подкреплением, нейронные сети, кластеризация, опорно-векторные машины и другие.

Некоторые авторы для обозначения эволюционных алгоритмов предпочитают использовать термин *эвристические алгоритмы*. Термин «эвристика» происходит от греческого слова *ηυρισκω*, которое транслитерируется как «*эврика*» на русском языке (*eurisko* на английском). Это слово означает «отыскиваю» или «открываю», а также является источником восклицания «эврика», которое мы используем, чтобы выразить радость, когда мы что-то обнаруживаем или решаем задачу. Эвристические алгоритмы – это методы, в которых для решения задачи используются эмпирические правила или подходы на основе здравого смысла. От эвристических алгоритмов, как правило, не рассчитывают получить лучший ответ на поставленную задачу, а ожидают только те решения, которые «достаточно близки» к самому лучшему. Термин метаэвристика используется для описания семейства эвристических алгоритмов. Большинство, если не все, эволюционных алгоритмов, которые мы обсуждаем в этой книге, могут быть реализованы по-разному и с целым рядом различных вариаций и параметров. Поэтому их все можно назвать метаэвристиками. Например, семейство всех алгоритмов оптимизации на основе муравьиной кучи можно назвать метаэвристикой муравьиной кучи.

Большинство авторов проводит различие между эволюционными алгоритмами и роевым интеллектом. Алгоритм роевого интеллекта основан на роях, встречающихся в природе (например, рои муравьев или стаи птиц). Оптимизация на основе муравьиной кучи (глава 10) и оптимизация на основе роя частиц (глава 11) – это два широко известных роевых алгоритма, и многие исследователи настаивают на том, что их не следует классифицировать как эволюционные алгоритмы. Однако некоторые авторы рассматривают роевой интеллект как подмножество эволюционных алгоритмов. Например, один из изобретателей оптимизации на основе роя частиц называет ее эволюционным алгоритмом [Shi и Eberhart, 1999]. Поскольку алгоритмы роевого интеллекта выполняются в том же общем виде, что и эволюционные алгоритмы, то есть путем эволюционного формирования популяции кандидатных решений задачи, которые улучшаются с каждой итерацией, мы рассматриваем роевой интеллект как эволюционный алгоритм.

Вся эта терминология неточна и контекстно зависима, однако в этой книге мы остановимся на термине *эволюционный алгоритм* и будем им обозначать алгоритм, который эволюционно формирует решение задачи в течение многочисленных итераций. Как правило, в соответствии со своей биологической основой одна итерация эволюционного алгоритма называется поколением. Однако это простое определение эволюционного алгоритма не является идеальным, потому что, например, оно подразумевает, что градиентный спуск – это эволюционный алгоритм, и никто не готов это признать. Таким образом, терминология эволюционных алгоритмов неоднородна и может вводить в заблуждение. Мы используем щекотливое определение, что алгоритм является эволюционным, если он в целом рассматривается как эволюционный алгоритм. Эта цикличность сначала надоедает, но те из нас, кто работает в данной области, через некоторое время к ней привыкают. В конце концов, естественный отбор определяется как выживание наиболее приспособленных, а под приспособленностью подразумеваются особи, которые с наибольшей вероятностью выживут.

1.2. Зачем нужна еще одна книга по эволюционным алгоритмам?

Существует ряд прекрасных книг по эволюционным алгоритмам, в которых поднимается вопрос, зачем нужен еще один учебник по теме эволюционных алгоритмов. Причина, почему эта книга была написана, состоит в том, чтобы предложить педагогический подход, перспективное видение и материал, которые недоступны ни в одной другой книге. В частности, мы надеемся, что эта книга предложит следующее.

- В книге дается простой восходящий подход, который помогает читателю получить четкое, но теоретически строгое понимание эволюционных алгоритмов. Во многих книгах обсуждаются различные эволюционные алгоритмы как алгоритмы из справочника без какой-либо теоретической поддержки. Другие книги больше похожи на научные монографии, чем на учебники, и не совсем доступны среднестатистическому студенту-инженеру. Эта книга пытается найти баланс, представляя простые в реализации алгоритмы наряду с небольшим объемом строгих теоретических выкладок и обсуждения компромиссов.
- В книге приведены простые примеры, которые дают читателю интуитивное понимание математики, уравнений и теории эво-

люционных алгоритмов. Во многих книгах представлена теория эволюционных алгоритмов, а затем приводятся примеры или задачи, которые не поддаются интуитивному пониманию. Вместе с тем вполне можно привести простые примеры и задачи, для решения которых нужны лишь бумага и карандаш. Эти простые задачи позволяют студенту непосредственнее увидеть то, как теория работает на практике.

- Исходный код на основе языка программирования MATLAB® для всех примеров в книге доступен на веб-сайте автора[2]. В ряде других учебников тоже предоставлен исходный код, но он часто является неполным или устаревшим, что расстраивает читателя. Адрес электронной почты автора также доступен на веб-сайте, и я с энтузиазмом приветствую отзывы, комментарии, предложения по улучшению исходного кода и его исправлениям. Конечно же, веб-адреса могут устареть, но эта книга содержит алгоритмические высокоуровневые листинги псевдокода, которые более постоянны, чем любые конкретные листинги программ. Обратите внимание, что примеры и программный код на MATLAB не предназначены в качестве эффективных или конкурентоспособных алгоритмов оптимизации; напротив, они предназначены только для того, чтобы дать читателю возможность получить общее представление о лежащих в их основе концепциях. Любое серьезное исследование или применение должно опираться на пример программного кода, который выступает только в качестве предварительной отправной точки.

- Данная книга включает в себя теорию и недавно разработанные эволюционные алгоритмы, которые недоступны в большинстве других учебников. Эти темы включают марковские теоретические модели эволюционных алгоритмов, системно-динамические модели эволюционных алгоритмов, алгоритмы на основе искусственной пчелиной семьи, биогеографическую оптимизацию, оппозиционное самообучение, алгоритмы на основе искусственных косяков рыб, перемешанных лягушачьих прыжков, оптимизацию на основе бактериальной кормодобычи и многие другие. Эти темы являются последними дополнениями в современное состояние дел в данной области, и их освещение в данной книге не соответствует уровню их освещения ни в одной другой книге.

[2] См. *http://academic.csuohio.edu/simond/EvolutionaryOptimization* – если адрес поменяется, то его легко можно отыскать в интернете.

Тем не менее эта книга не предназначена для того, чтобы быть высокоуровневым описанием текущего состояния дел в какой-либо конкретной области эволюционно-алгоритмических исследований. Вместо этого настоящая книга предназначена для высокоуровневого описания многих областей эволюционно-алгоритмических исследований, чтобы читатель мог получить широкое представление об эволюционных алгоритмах и иметь хорошие возможности для проведения дополнительных исследований в данной области в его текущем развитии.

1.3. Предварительные условия

В общем случае студент ничего не получит из такого курса без написания собственного программного продукта на основе эволюционного алгоритма. Поэтому в качестве предварительного условия можно было бы указать компетентные навыки программирования. В университете, где я преподаю этот курс студентам электротехники и вычислительной техники, конкретные предварительные условия для прохождения данного курса отсутствуют; предварительным условием для студентов старших курсов является статус старшекурсника, для студентов-выпускников предварительные условия отсутствуют. Однако я исхожу из того, что и старшекурсники, и студенты-выпускники являются хорошими программистами.

Используемые в книге обозначения предполагают, что читатель знаком со стандартной математической записью, которая используется в алгебре, геометрии, теории множеств и математическом анализе. Поэтому еще одним предварительным условием для понимания этой книги является уровень математической зрелости, характерный для продвинутого студента старших курсов. Математические обозначения описаны в разделе 1.5. Если читатель способен понять обозначения, описанные в указанном разделе, то существует неплохой шанс, что он сможет проследить за ходом обсуждения в оставшейся части книги.

Математика в теоретических разделах этой книги (глава 4, раздел 7.6, подавляющая часть главы 13 и несколько других разрозненных разделов) требует понимания теории вероятностей и линейных систем. Студенту будет трудно следить за логикой изложения этого материала, если он не закончил курс по этим двум предметам. В учебном курсе, ориентированном на старшекурсников, вероятно, следует этот материал пропустить.

1.4. Домашние задания

Задачи в конце каждой главы были написаны для того, чтобы обеспечить гибкость преподавателю и студенту. Задачи включают письменные и компьютерные упражнения. Письменные упражнения предназначены для укрепления студентом своего понимания теории, углубления своего интуитивного понимания понятий и развития своих аналитических навыков. Компьютерные упражнения призваны помочь студенту развить исследовательские навыки и научиться применять теорию к типам задач, которые обычно встречаются в промышленности. Оба типа задач важны для наращивания своей квалификации в работе с эволюционными алгоритмами. Разница между письменными и компьютерными упражнениями не является строгой, а скорее представляет собой нечеткое разделение. То есть некоторые письменные упражнения, возможно, потребуют некоторой работы с компьютером, а компьютерные упражнения потребуют некоторого анализа на бумаге. Инструктор может предусмотреть задания, связанные с эволюционными алгоритмами, исходя из собственных интересов. Семестровые, проектно-ориентированные задания часто являются наглядным примером таких тем, как эта. Например, студентам может быть поручено решить некоторую практическую оптимизационную задачу с использованием рассмотренных в данной книге эволюционных алгоритмов, применяя по одному эволюционному алгоритму на главу, а затем сравнивая результативность эволюционных алгоритмов и их вариации в конце семестра.

Справочник для преподавателей с решениями всех задач книги (как письменных, так и компьютерных упражнений) имеется в распоряжении у издателя. Преподавателям курсов рекомендуется обратиться к издателю за дополнительной информацией о получении справочника с решениями упомянутых задач. Для защиты целостности домашних заданий справочник с решениями задач будет предоставляться только преподавателям курса.

1.5. Обозначения

К сожалению, в английском языке нет гендерно-нейтрального местоимения третьего лица единственного числа. Поэтому мы используем слова «он» или «ему» для обозначения общего третьего лица, будь то мужчина или женщина. Такое именование может показаться неудобным как для писателей, так и для читателей, но оно представляется наиболее удовлетворительным выходом из трудного положения.

В приведенном ниже списке описаны некоторые математические обозначения, принятые в этой книге.

- $x \leftarrow y$ – вычислительное обозначение, указывающее, что переменной x присваивается значение y. Например, рассмотрим следующий ниже алгоритм:

  a = coefficient of x^2
  b = coefficient of x^1
  c = coefficient of x^0
  $x^* \leftarrow (-b + \sqrt{b^2 - 4ac})/(2a)$

  Первые три строки не являются инструкциями присваивания в алгоритме; они просто описывают или определяют значения a, b и c. Эти три параметра могуть быть заданы пользователем, другим алгоритмом или процессом. Последняя строка, однако, является инструкцией присваивания, которая обозначает, что значение в правой стороне от стрелки записывается в x^*.

- $df(\cdot)/dx$ – это полная производная от $f(\cdot)$ по x. Например, предположим, что $y = 2x$ и $f(x, y) = 2x + 3y$. Тогда $f(x, y) = 8x$ и $df(\cdot)/dx = 8$.

- $f_x(\cdot)$, также записываемое как $\partial f(\cdot)/\partial x$, является частной производной от $f(\cdot)$ по x. Например, предположим, что $y = 2x$ и $f(x, y) = 2x + 3y$. Тогда $f_x(x, y) = 2$.

- $\{x: x \in S\}$ – это множество всех x – таких, что x принадлежит множеству S. Аналогичная запись используется для обозначения тех значений x, которые удовлетворяют любому другому конкретному условию. Например, запись $\{x : x^2 = 4\}$ тождественна $\{x : x \in \{-2, +2\}\}$, которая тождественна $\{-2, +2\}$.

- $[a, b]$ – это закрытый интервал между a и b, который означает, что $\{x : a \le x \le b\}$. В зависимости от контекста он может представлять множество целых чисел и множество вещественных чисел.

- (a, b) – это открытый интервал между a и b, который означает, что $\{x : a < x < b\}$. В зависимости от контекста он может представлять множество целых чисел и множество вещественных чисел.

- Если из контекста понятно, что $i \in S$, то $\{x_i\}$ является сокращенной записью для $\{x_i : i \in S\}$. Например, если $i \in [1, N]$, то $\{x_i\} = \{x_1, x_2, \ldots, x_N\}$.

- $S_1 \cup S_2$ – это множество из всех x – таких, что x принадлежит либо множеству S_1, либо множеству S_2. Например, если $S_1 = \{1, 2, 3\}$ и $S_2 = \{7, 8\}$, то $S_1 \cup S_2 = \{1, 2, 3, 7, 8\}$.
- $|S|$ – это число элементов во множестве S. Например, если $S = \{i : i \in [4, 8]\}$, то $|S| = 5$. Если $S = \{3, 19, \pi, \sqrt{2}\}$, то $|S| = 4$. Если $S = \{\alpha : 1 < \alpha < 3\}$, то $|S| = \infty$.
- $\emptyset$ – это пустое множество. $|\emptyset| = 0$.
- x mod y – это остаток после деления x на y. Например, 8 mod 3 = 2.
- $\lceil x \rceil$ – это верхнее округление x; то есть наименьшее целое число, которое больше или равно x. Например, $\lceil 3.9 \rceil = 4$, а $\lceil 5 \rceil = 5$.
- $\lfloor x \rfloor$ – это нижнее округление x; то есть наибольшее целое число, которое меньше или равно x, например $\lfloor 3.9 \rfloor = 3$, и $\lfloor 5 \rfloor = 5$.
- Запись $\min_x f(x)$ обозначает задачу нахождения значения x, которое дает наименьшее значение $f(x)$. Кроме того, указанная запись может обозначать наименьшее значение $f(x)$. Например, предположим, что $f(x) = (x - 1)^2$. Тогда мы можем решить задачу $\min_x f(x)$, используя исчисление или построив график функции $f(x)$ и визуально отметив наименьшее значение $f(x)$. Для этого примера мы находим, что $\min_x f(x) = 0$. Аналогичное определение справедливо для $\max_x f(x)$.
- $\arg\min_x f(x)$ – это значение x, которое приводит к наименьшему значению $f(x)$. Например, предположим, что $f(x) = (x - 1)^2$. Наименьшее значение $f(x)$ равно 0, что происходит, когда $x = 1$, поэтому для данного примера $\arg\min_x f(x) = 1$. Аналогичное определение справедливо для $\arg\max_x f(x)$.
- R^s – это множество всех вещественных s-элементных векторов. В зависимости от контекста эта запись может обозначать либо векторы-столбцы, либо векторы-строки.
- $R^{s \times p}$ – это множество всех вещественных $s \times p$-матриц.
- $\{y_k\}_{k=L}^{U}$ – это множество всех y_k, где целое число k колеблется между L и U. Например, $\{y_k\}_{k=2}^{5} = \{y_2, y_3, y_4, y_5\}$.
- $\{y_k\}$ – это множество всех y_k, где целое число k колеблется между контекстно-зависимым нижним пределом и контекстно-зависимым верхним пределом. Например, предположим, что контекст указывает на наличие трех значений: y_1, y_2 и y_3. Тогда $\{y_k\} = \{y_1, y_2, y_3\}$.

- Знак $\exists$ означает «существует»; знак $\nexists$ означает «не существует». Например, если $Y = \{6,1,9\}$, то $\exists y < 2 : y \in Y$. Однако $\nexists y > 10 : y \in Y$.
- Запись $A \Rightarrow B$ означает, что A подразумевает B. Например, $(x > 10) \Rightarrow (x > 5)$.
- I – это единичная матрица. Ее размеры зависят от контекста.

Дополнительные обозначения см. в списке сокращений на стр. 21.

1.6. План изложения

Настоящая книга разделена на шесть частей.

1. *Часть I* состоит из этого введения и еще одной главы, которая охватывает вводный материал, связанный с оптимизацией. Она знакомит с различными типами оптимизационных задач, простым, но эффективным алгоритмом восхождения к вершине холма и завершается обсуждением признаков, которые делают алгоритм интеллектуальным.
2. *Часть II* посвящена четырем эволюционным алгоритмам, которые обычно считаются классикой:
   - генетический алгоритм;
   - эволюционное программирование;
   - эволюционные стратегии;
   - генетическое программирование.

   Часть II также включает главу, в которой рассматриваются методы математического анализа генетических алгоритмов. Часть II завершается главой, в которой обсуждаются некоторые из многих алгоритмических вариаций, которые могут использоваться в указанных классических алгоритмах. Эти же вариации можно использовать и в более поздних эволюционных алгоритмах, которые рассматриваются в следующей части.
3. В *части III* обсуждается несколько более поздних эволюционных алгоритмов. Некоторые из них не совсем новые и относятся к 1980-м годам, однако другие датируются только первым десятилетием XXI века.
4. В *части IV* обсуждаются специальные типы оптимизационных задач и показан способ модификации эволюционных алгоритмов

предыдущих глав для их решения. Эти специальные типы задач включают в себя:

- комбинаторные задачи, область решения которых состоит из целых чисел;
- ограниченные задачи, область решения которых ограничена известным множеством;
- многокритериальные задачи, в которых желательно минимизировать более одной целевой функции одновременно;
- задачи с шумными или дорогостоящими функциями приспособленности, для которых трудно получить точную результативность кандидатного решения или для которых вычислительно затратно оценить результативность кандидатного решения.

5. *Часть V* включает несколько приложений, в которых обсуждаются важные или интересные темы.
   - *Приложение A* содержит различные практические советы для студентов и исследователей эволюционных алгоритмов.
   - В *приложении B* обсуждается теорема об отсутствии бесплатных обедов, которая постулирует, что в среднем все оптимизационные алгоритмы работают одинаково. В нем также обсуждается вопрос о том, как использовать статистику для оценивания различий между эволюционными алгоритмами.
   - В *приложении C* приведено несколько стандартных эталонных функций, которые можно использовать для сравнения результативности разных эволюционных алгоритмов.

1.7. Учебный курс на основе данной книги

Любой курс на основе настоящей книги должен начинаться с глав 1 и 2, в которых дается краткий обзор оптимизационных задач. Остальные главы после них можно изучать практически в любом порядке, в зависимости от предпочтений и интересов инструктора. Очевидным исключением является то, что изучение генетических алгоритмов (глава 3) должно предшествовать изучению их математических моделей (глава 4).

Кроме того, по крайней мере, одна глава в частях II или III (то есть, по крайней мере, один конкретный эволюционный алгоритм) должна быть подробно рассмотрена перед любой из глав в части IV.

Большинство курсов, как минимум, охватывает главы 3 и 5–7, для того чтобы предоставить студенту общие сведения в области классических

эволюционных алгоритмов. Если студенты обладают достаточно высоким уровнем математических знаний и если есть время, то курс также в какой-то момент должен включить главу 4. Глава 4 важна для студентов-выпускников, потому что она помогает им понять, что эволюционные алгоритмы – это не только качественный предмет, но для них также может и должна иметься некая теоретическая основа. Слишком много исследовательских работ по эволюционным алгоритмам сегодня основано на незначительных алгоритмических корректировках без какой-либо математической поддержки. Многие практикующие специалисты в области эволюционных алгоритмов интересуются только получением результатов, что хорошо, однако академические исследователи должны быть вовлечены как в теорию, так и в практику.

Главы в частях III и IV могут быть рассмотрены с учетом конкретных интересов преподавателя или студентов.

Приложения не включены в основную часть книги, потому что они не об эволюционных алгоритмах как таковых, но важность приложений не следует недооценивать. В частности, материалы в приложениях B и C имеют решающее значение и должны быть включены в каждый курс по эволюционным алгоритмам. Я рекомендую обсудить некоторые детали этих двух приложений сразу же после первой главы в частях II или III.

Собрав все приведенные выше советы вместе, предлагаем схематичный план для одного семестра старшего курса.

- Главы 1 и 2.
- Глава 3.
- Приложения B и C.
- Главы 4–8. Я рекомендую пропустить главу 4 для большинства студентов старших курсов и в случае коротких курсов.
- Несколько глав в части III, исходя из предпочтений инструктора. Рискуя начать «эволюционно-алгоритмическую войну» с моими читателями, выскажу малопопулярную точку зрения и заявлю, что алгоритмы оптимизации на основе муравьиной кучи (ACO), оптимизации на основе роя частиц (PSO) и дифференциальной эволюции (DE) являются одними из наиважнейших «других» эволюционных алгоритмов, и поэтому инструктор должен как минимум охватить главы 10–12.
- Несколько глав в части IV, исходя из предпочтений инструктора и имеющегося времени.

Глава 2

Оптимизация

Оптимизацией пронизано все то, что мы делаем, и она управляет почти каждым аспектом инженерного дела.

– Деннис Бернштейн [Bernstein, 2006]

Как указано в приведенной выше цитате, оптимизация является частью почти всего, что мы делаем. Оптимизации подлежат штатные расписания, стили преподавания, экономические системы, игровые стратегии, биологические системы и системы здравоохранения. Оптимизация является увлекательной областью исследования не только из-за ее алгоритмического и теоретического содержания, но и из-за ее универсальной применимости.

Краткий обзор главы

В данной главе дается краткий обзор оптимизации (раздел 2.1), включая ограниченную оптимизацию с учетом ограничений (раздел 2.2), оптимизационные задачи, которые имеют несколько целей (раздел 2.3) и оптимизационные задачи, которые имеют несколько решений (раздел 2.4). Подавляющая часть нашей работы в этой книге сосредоточена на непрерывных оптимизационных задачах, то есть задачах, где независимая переменная может варьироваться непрерывно. Однако задачи, в которых независимая переменная ограничена конечным множеством, то есть так называемые комбинаторные задачи, также представляют большой интерес, и мы познакомимся с ними в разделе 2.5. В разделе 2.6 мы представим простой универсальный оптимизационный алгоритм под названием «восхождение к вершине холма», а также обсудим несколько его вариаций. Наконец, в разделе 2.7 мы обсудим несколько идей, связанных с природой интеллекта, и покажем, как они связаны с

алгоритмами эволюционной оптимизации, которые мы представим в последующих главах.

2.1. Неограниченная оптимизация

Оптимизация применима практически ко всем сферам жизни. Оптимизационные алгоритмы могут применяться ко всему, от разведения трубкозубов (африканских муравьедов) до исследования зигот (оплодотворенной яйцеклетки). Возможные применения эволюционных алгоритмов ограничены только воображением инженера, именно по этой причине в течение прошедших нескольких десятилетий эволюционные алгоритмы очень широко исследовались и применялись на практике.

Например, в инженерном деле эволюционные алгоритмы используются для поиска лучших траекторий робота для определенной задачи. Предположим, что на вашем производственном предприятии есть робот, и вы хотите, чтобы он выполнял свою задачу таким образом, чтобы он заканчивал ее как можно быстрее или использовал наименьшую возможную мощность. Как рассчитать лучший возможный путь для робота? Существует так много возможных путей, что задача найти лучшее решение является чрезвычайно сложной. Однако эволюционные алгоритмы способны сделать такую задачу управляемой (если вообще не легкой) и, по крайней мере, найти хорошее решение (если вообще не лучшее). Роботы очень нелинейны, поэтому, как мы видели в простом примере, представленном ранее в этой главе, поисковое пространство для задач робототехнической оптимизации сводится к большому числу пиков и впадин. Но в случае с робототехническими задачами ситуация еще хуже, потому что пики и впадины лежат в многомерном пространстве (а не в простом трехмерном пространстве, которое мы видели ранее). Сложность задач робототехнической оптимизации делает их естественной мишенью для эволюционных алгоритмов. Эта идея была применена как для стационарных роботов (то есть роботов-манипуляторов), так и для мобильных роботов.

Эволюционные алгоритмы также используются для тренировки нейронных сетей и систем нечеткой логики. В нейронных сетях мы должны выяснить архитектуру сети и нейронные веса, с тем чтобы получить лучшую результативность сети. Опять же, существует так много возможностей, что эта задача просто колоссальна. Вместе с тем можно применить эволюционные алгоритмы и найти лучшую конфигурацию и лучшие веса. Та же задача связана и с системами нечеткой логики. Какую базу правил мы должны использовать? Сколько функций принад-

лежности мы должны использовать? Какие формы функций принадлежности следует использовать? Эволюционные алгоритмы помогали и помогают решать эти сложные оптимизационные задачи.

Эволюционные алгоритмы также применялись в медицинской диагностике. Например, после биопсии специалисты-медики идентифицируют, какие клетки являются раковыми, а какие нет. Какие признаки следует искать, чтобы диагностировать рак? Какие признаки являются наиболее важными, а какие – неактуальными? Какие признаки важны только, если пациент относится к определенной демографической группе? Эволюционный алгоритм способен помогать принимать такие решения. Эволюционному алгоритму всегда нужен профессионал, который его запустит и натренирует, но после этого он фактически может превзойти своего учителя. Эволюционный алгоритм не только не устает и не истощается, но и может извлекать шаблоны из данных, которые могут быть слишком утонченными, чтобы их могли распознавать люди. Эволюционные алгоритмы применялись для диагностики нескольких разных видов рака.

После того как болезнь была диагностирована, следующий трудный вопрос включает в себя управление болезнью. Например, после обнаружения рака какое лечение для пациента является лучшим? Как часто следует проводить облучение, каким оно должно быть и в каких дозах? Как следует лечить побочные эффекты? Это еще одна сложная оптимизационная задача. Неправильное лечение может принести больше вреда, чем пользы. Определение правильного вида лечения является нетривиальной функцией таких факторов, как тип рака, локализация рака, демографические показатели, общее состояние здоровья и другие. Поэтому генетические алгоритмы применяются не только для диагностики заболеваний, но и для планирования лечения.

Эволюционные алгоритмы следует рассматривать всегда, когда вы хотите решить сложную задачу. Это не означает, что эволюционные алгоритмы всегда являются лучшим выбором для работы. В калькуляторах программа на основе эволюционного алгоритма для сложения чисел не используется, потому что для этого есть гораздо более простые и эффективные алгоритмы. Однако эволюционные алгоритмы, по крайней мере, следует рассматривать для любой нетривиальной задачи. Если вы хотите разработать проект объекта жилищного строительства или транспортной системы, то эволюционный алгоритм, возможно, будет ответом. Если вы хотите спроектировать сложную электрическую схему или компьютерную программу, то эволюционный алгоритм вполне сможет выполнить эту работу.

Оптимизационная задача может быть записана как минимизационная или как максимизационная задача. Иногда мы пытаемся минимизировать функцию, а иногда пытаемся ее максимизировать. Эти две задачи легко преобразуются в другую форму:

$$\begin{aligned} \min_x f(x) &\Leftrightarrow \max_x [-f(x)] \\ \max_x f(x) &\Leftrightarrow \min_x [-f(x)]. \end{aligned} \tag{2.1}$$

Функция $f(x)$ называется целевой функцией, а вектор x – независимой переменной, или переменной решения. Обратите внимание, что в зависимости от контекста термины независимая переменная и переменная решения иногда относятся ко всему вектору x, а иногда к определенным элементам в x. Элементы вектора x также называются признаками решения. Число элементов в x называется размерностью задачи. Как мы видим из уравнения (2.1), любой алгоритм, разработанный для минимизации функции, может быть легко использован для максимизации функции, и любой алгоритм, разработанный для максимизации функции, может быть легко использован для ее минимизации. Когда мы пытаемся минимизировать функцию, мы называем значение этой функции функцией стоимости, или просто стоимостью (от англ. *cost*). Когда мы пытаемся максимизировать функцию, то мы называем значение этой функции приспособленностью (от англ. *fitness*).

$$\begin{aligned} &\min_x f(x) \Rightarrow f(x) && \text{называется «стоимостью»,} \\ &&& \text{или «целевым критерием»} \\ &\max_x f(x) \Rightarrow f(x) && \text{называется «приспособленностью»,} \\ &&& \text{или «целевым критерием».} \end{aligned} \tag{2.2}$$

■ Пример 2.1

Данный пример иллюстрирует терминологию, которую мы используем в этой книге. Предположим, мы хотим минимизировать функцию

$$f(x, y, z) = (x - 1)^2 + (y + 2)^2 + (z - 5)^2 + 3. \tag{2.3}$$

Переменные x, y и z называются независимыми переменными, переменными решения или признаками решения; все три термина эквивалентны. Это трехмерная задача, $f(x, y, z)$ называется целевой функцией, или функцией стоимости. Мы можем изменить задачу на задачу максимизации, определив $g(x, y, z) = -f(x, y, z)$, и попытаться максимизировать $g(x, y, z)$. Функция $g(x, y, z)$ называется целевой функцией, или функцией приспособленности. Решение задачи $\min f(x, y, z)$ такое же, как и реше-

ние задачи max $y(x, y, z)$, и равно $x = 1$, $y = -2$ и $z = 5$. Однако оптимальное значение для $f(x, y, z)$ является противоположным оптимальному значению для $g(x, y, z)$.

■

Иногда оптимизация проста и может быть выполнена с помощью аналитических методов, как мы увидим в следующем далее примере.

■ Пример 2.2

Рассматрим задачу

$$\min_x f(x), \quad \text{где } f(x) = x^4 + 5x^3 + 4x^2 - 4x + 1. \tag{2.4}$$

График $f(x)$ показан на рис. 2.1. Поскольку функция $f(x)$ является квадратичным полиномом (так называемым многочленом четвертой степени, или четвертого порядка), мы знаем, что она имеет не более трех стационарных точек, то есть три значения x, при которых ее производная $f'(x) = 0$. Эти точки, как видно по рис. 2.1, встречаются при $x = -2.96$, $x = -1.10$ и $x = 0.31$. Мы можем подтвердить, что $f'(x)$, которая равна $4x^3 + 15x^2 + 8x - 4$, равна нулю при этих трех значениях x. Далее мы можем найти, что вторая производная от $f(x)$ в этих трех точках равняется

$$f''(x) = 12x^2 + 30x + 8 = \begin{cases} 24.33, & x = -2.96 \\ -10.48, & x = -1.10 \\ 18.45, & x = 0.31 \end{cases}. \tag{2.5}$$

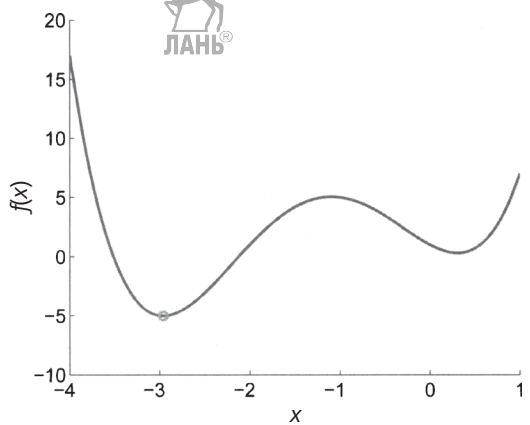


Рис. 2.1. Пример 2.2: простая задача минимизации. $f(x)$ имеет два локальных минимума и один глобальный минимум, который встречается при $x = -2.96$

Напомним, что вторая производная функции при локальном минимуме положительна, а вторая производная функции при локальном максимуме отрицательна. Таким образом, значения $f''(x)$ в стационарных точках подтверждают, что $x = -2.96$ является локальным минимумом, $x = -1.10$ – локальным максимумом, а $x = 0.31$ – еще одним локальным минимумом.

■

Функция примера 2.2 имеет два локальных минимума и один глобальный минимум. Обратите внимание, что глобальный минимум также является локальным минимумом. В некоторых функциях $\min_x f(x)$ встречается при более чем одном значении x; если это происходит, то $f(x)$ имеет несколько глобальных минимумов. Локальный минимум x^* может быть определен как

$$f(x^*) < f(x) \quad \text{для всех } x \text{ such that } \|x - x^*\| < \varepsilon, \tag{2.6}$$

где $\|\cdot\|$ – это некий метрический показатель расстояния, а $\varepsilon > 0$ – некий задаваемый пользователем размер окрестности. На рис. 2.1 видно, что $x = 0.31$ является локальным оптимумом, если, например, размер окрестности $\varepsilon = 1$, но не локальным оптимумом, если $\varepsilon = 4$. Глобальный минимум x^* может быть определен как

$$f(x^*) \leq f(x) \quad \text{для всех } x. \tag{2.7}$$

2.2. Ограниченная оптимизация

Довольно часто оптимизационная задача имеет ограничения. То есть перед нами поставлена задача минимизации некой функции $f(x)$ с ограничениями на допустимые значения x, как в следующем ниже примере.

■ Пример 2.3

Рассмотрим задачу

$$\min_x f(x), \quad \text{где} \quad f(x) = x^4 + 5x^3 + 4x^2 - 4x + 1 \quad \text{и} \quad x \geq -1.5. \tag{2.8}$$

Это та же задача, что и в примере 2.2, за исключением того, что x ограничено. График $f(x)$ и допустимые значения x показаны на рис. 2.2, и анализ графика показывает ограниченный минимум. Чтобы решить эту задачу аналитически, мы находим три стационарные точки $f(x)$, как в примере 2.2, игнорируя ограничение. Мы находим, что два локаль-

ных минимума встречаются при $x = -2.96$ и $x = 0.31$, как в примере 2.2. Мы видим, что только одно из этих значений, $x = 0.31$, удовлетворяет ограничению. Далее мы должны вычислить $f(x)$ на границе ограничения, чтобы убедиться, что она меньше, чем при локальном минимуме $x = 0.31$. Мы находим, что

$$f(x) = \begin{cases} 4.19, & \text{для } x = -1.50 \\ 0.30, & \text{для } x = 0.30 \end{cases}. \tag{2.9}$$

Мы видим, что $x = 0.31$ является минимизирующим значением x для задачи ограниченной минимизации.

Если бы граница ограничения была дальше влево, то минимизирующее значение x возникло бы на границе ограничения, а не на локальном минимуме $x = 0.31$. Если бы граница ограничения была слева от $x = -2.96$, то минимизирующее значение x для задачи ограниченной минимизации было бы таким же, как и для задачи неограниченной минимизации.

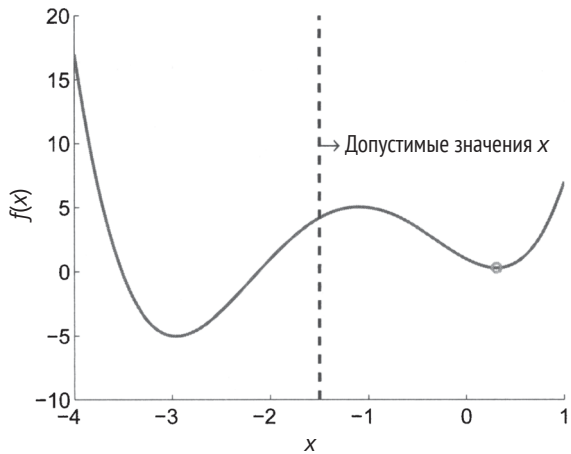


Рис. 2.2. Пример 2.3: простая задача ограниченной минимизации. Ограниченный минимум имеет место при $x = 0.31$

■

Реальные оптимизационные задачи почти всегда имеют ограничения. Кроме того, в реальных оптимизационных задачах оптимизирующее значение независимой переменной почти всегда возникает на границе ограничения. Это неудивительно, потому что мы обычно рассчитываем получить лучший инженерный проект, распределение ре-

сурсов или другую цель оптимизации, используя всю доступную энергию или силу или какой-либо другой ресурс [Bernstein, 2006]. Поэтому ограничения важны практически во всех реальных оптимизационных задачах. В главе 19 более подробно рассматривается ограниченная эволюционная оптимизация.

2.3. Многокритериальная оптимизация

Реальные оптимизационные задачи не только ограничены, но и являются многокритериальными[1]. Это означает, что мы заинтересованы в минимизации более чем одной меры одновременно. Например, в задаче управления двигателем мы, возможно, будем заинтересованы в минимизации ошибки слежения и одновременно минимизации энергопотребления. Мы можем получить очень небольшую ошибку слежения за счет высокого энергопотребления либо можем допустить большую ошибку слежения при малом энергопотреблении. В крайнем случае, мы можем выключить двигатель, для того чтобы достигнуть нулевого энергопотребления, но тогда наша ошибка слежения не будет очень хорошей.

■ Пример 2.4

Рассмотрим задачу

$$\min_x [f(x) \text{ и } g(x)], \quad \text{где} \quad f(x) = x^4 + 5x^3 + 4x^2 - 4x + 1 \quad \text{и} \quad g(x) = 2(x+1)^2. \tag{2.10}$$

Первый целевой критерий минимизации, $f(x)$, такой же, как и в примере 2.2. Однако теперь мы также хотим минимизировать $g(x)$. График $f(x)$, $g(x)$ и их минимумы показаны на рис. 2.3. Анализ графика показывает, что $x = -2.96$ минимизирует $f(x)$, в то время как $x = -1$ минимизирует $g(x)$. Не ясно, какое значение x будет для этой задачи наиболее предпочтительным, потому что у нас две противоречивые цели. Однако по рис. 2.3 должно быть очевидно, что мы никогда не захотим, чтобы $x < -2.96$ или $x > 0.31$. Если x уменьшается с -2.96 или увеличивается с 0.31, то целевые критерии $f(x)$ и $g(x)$ увеличиваются, что явно нежелательно.

Один из способов вычислить эту задачу – построить график $g(x)$ как функцию от $f(x)$. Это показано на рис. 2.4, где мы варьировали x от -3.4 до 0.8. Ниже мы рассмотрим каждый раздел графика.

[1] Такие оптимизационные задачи также называются многоцелевыми, многообъектными и многоатрибутными. – *Прим. перев.*

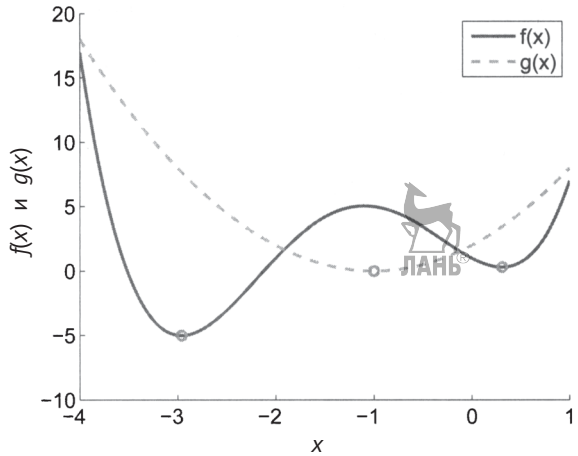


Рис. 2.3. Пример 2.4: простая задача многокритериальной минимизации. $f(x)$ имеет два минимума, а $g(x)$ – один минимум. Два целевых критерия конфликтуют

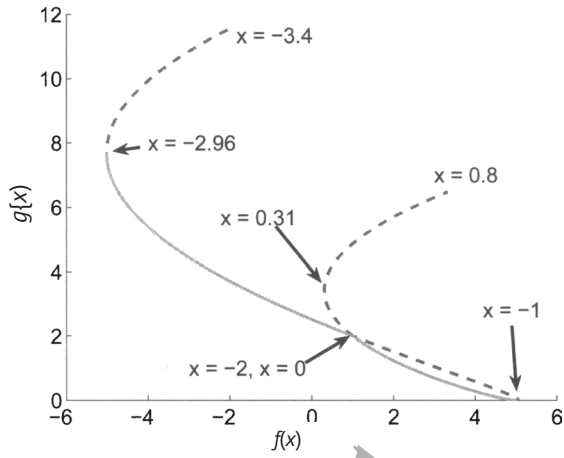


Рис. 2.4. Пример 2.4: на этом рисунке $g(x)$ показана как функция от $f(x)$ для простой задачи многокритериальной минимизации по мере того, как x варьируется от –3.4 до 0.8. Сплошная линия – это фронт Парето

- $x \in [-3.4, -2.96]$: когда x увеличивается от –3.4 до –2.96, $f(x)$ и $g(x)$ уменьшаются вместе. Поэтому мы никогда не выберем $x < -2.96$.
- $x \in [-2.96, -1]$: когда x увеличивается от –2.96 до –1, $g(x)$ уменьшается, а $f(x)$ увеличивается.

- $x \in [-1, 0]$: когда x увеличивается от -1 до 0, $g(x)$ увеличивается, а $f(x)$ уменьшается. Однако на этой части графика, несмотря на то что $g(x)$ увеличивается, он все еще меньше $g(x)$ для $x \in [-2, -1]$, поэтому $x \in [-1, 0]$ предпочтительнее $x \in [-2, -1]$.
- $x \in [0, 0.31]$: когда x увеличивается от 0 до 0.31, $g(x)$ увеличивается, а $f(x)$ уменьшается. По графику мы видим, что для $x \in [0, 0.31]$ $g(x)$ больше, чем она на части графика $x \in [-2.96, -2]$. Поэтому мы никогда не захотим выбрать $x \in [0, 0.31]$.
- $x \in [0.31, 0.8]$: наконец, когда x увеличивается от 0.31 до 0.8, $f(x)$ и $g(x)$ увеличиваются вместе. Поэтому мы никогда не выберем $x > 0.31$.

Резюмируя вышеизложенные результаты, мы начертим потенциально желательные значения $f(x)$ и $g(x)$ сплошной линией на рис. 2.4. Для значений x на сплошной линии мы не можем найти никаких других значений x, которые уменьшат одновременно $f(x)$ и $g(x)$. Сплошная линия называется фронтом Парето, а соответствующее множество значений x – множеством Парето.

$$\begin{aligned} &\text{множество Парето:} \quad x^* = \{x : x \in [-2.96, -2] \text{ или } x \in [-1, 0]\}; \\ &\text{фронт Парето:} \quad \{(f(x), g(x)) : x \in x^*\}. \end{aligned} \tag{2.11}$$

После того как мы получим множество Парето, мы больше ничего не можем сказать об оптимальном значении x. Выбор точки вдоль фронта Парето теперь является вопросом инженерного суждения как окончательного решения. Фронт Парето дает множество разумных вариантов на выбор, но любой вариант x из множества Парето по-прежнему влечет за собой компромисс между двумя целевыми критериями.

■

Пример 2.4 представляет собой довольно простую задачу многокритериальной оптимизации только с двумя целевыми критериями. Типичная оптимизационная задача в реальном мире включает в себя гораздо больше, чем просто две цели, и поэтому ее фронт Парето трудно получить. Даже если бы мы могли получить фронт Парето, мы не смогли бы визуализировать его из-за его высокой размерности. В главе 20 более подробно рассматривается эволюционная многокритериальная оптимизация.

2.4. Мультимодальная оптимизация

Задача мультимодальной оптимизации – это задача, имеющая более одного локального минимума. Мы видели пример мультимодальной задачи на рис. 2.1, но в этой задаче имелось только два локальных минимума, и поэтому с ней было довольно легко справиться. Некоторые задачи, однако, имеют много локальных минимумов, и бывает трудно обнаружить, какой минимум является глобальным минимумом.

■ Пример 2.5

Рассматрим задачу

$\min_{x,y} f(x, y)$, где

$$f(x,y) = e - 20\exp\left(-0.2\sqrt{\frac{x^2+y^2}{2}}\right) - \exp\left(\frac{\cos(2\pi x)+\cos(2\pi y)}{2}\right). \quad (2.12)$$

Это двумерная функция Экли. Ее график показан на рис. 2.5, а сама функция определена в приложении С.1.2. График, аналогичный тому, который приведен на рис. 2.5, часто называют ландшафтом приспособленности, поскольку он графически иллюстрирует, как функция приспособленности или функция стоимости варьируется вместе с независимыми переменными. Мы не можем проиллюстрировать такой график, как на рис. 2.5, более чем в двух размерностях, но даже если у нас задача с более чем двумя размерностями, приспособленность или стоимость как функция независимых переменных по-прежнему называется ландшафтом приспособленности. На рис. 2.5 показано, что даже в двух размерностях функция Экли имеет много локальных минимумов. Представьте, сколько минимумов она будет иметь в 20 и 30 размерностях. Для того чтобы отыскать локальные минимумы, мы могли бы атаковать эту задачу, взяв производную от $f(x, y)$ по x и y, а затем решив уравнение $f_x(x, y) = f_y(x, y) = 0$. Однако решить такую систему одновременных уравнений может оказаться трудно.

Предыдущий пример показывает, почему полезны эволюционные алгоритмы. Мы смогли решить примеры 2.2, 2.3 и 2.4 с помощью графических методов или исчисления, но многие реальные задачи больше похожи на пример 2.5, за тем исключением, что независимых переменных больше, имеется несколько целевых критериев и имеются ограничения. С такими типами задач методы на основе исчисления или гра-

фиков не дотягивают, а эволюционные алгоритмы могут давать более высокие результаты.

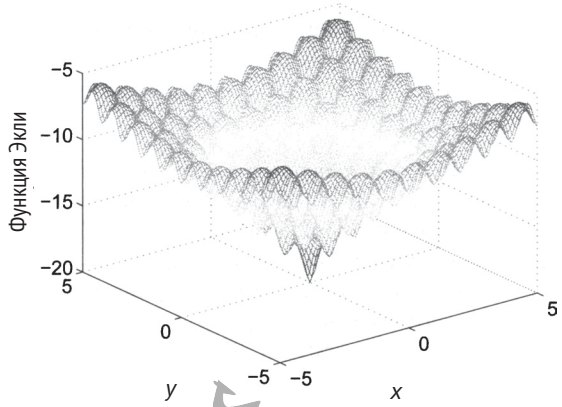


Рис. 2.5. Пример 2.5: двухмерная функция Экли

2.5. Комбинаторная оптимизация

До сих пор рассматривались непрерывные оптимизационные задачи, то есть где независимые переменные могли варьироваться непрерывно. Однако существует ряд оптимизационных задач, для которых независимые переменные ограничены множеством дискретных значений. Такие задачи называются комбинаторно-оптимизационными задачами.

■ Пример 2.6

Предположим, что предприниматель хочет посетить четыре филиала, начиная и заканчивая в своем домашнем офисе. Домашний офис находится в городе A, а филиалы – в городах B, C и D. Предприниматель хочет посетить филиалы в том порядке, который минимизирует его общее расстояние пути. У данной задачи существует шесть возможных решений:

$$\begin{aligned}
&\text{S1: } A \rightarrow +B \rightarrow +C \rightarrow +D \rightarrow +A \\
&\text{S2: } A \rightarrow +B \rightarrow +D \rightarrow +C \rightarrow +A \\
&\text{S3: } A \rightarrow +C \rightarrow +B \rightarrow +D \rightarrow +A \\
&\text{S4: } A \rightarrow +C \rightarrow +D \rightarrow +B \rightarrow +A \\
&\text{S5: } A \rightarrow +D \rightarrow +B \rightarrow +C \rightarrow +A \\
&\text{S5: } A \rightarrow +D \rightarrow +C \rightarrow +B \rightarrow +A
\end{aligned} \tag{2.13}$$

Предприниматель легко может решить эту задачу, рассчитав общее расстояние для каждого из шести возможных решений[2].

■

Задача примера 2.6 называется задачей коммивояжера (travelling salesman problem, TSP)[3]. Мы можем легко перечислить все возможные решения задачи коммивояжера с четырьмя городами. Поиск по всем возможным решениям комбинаторной задачи называется поиском методом грубой силы, или исчерпывающим поиском. Если у вас для этого есть время, то это часто лучший способ решить комбинаторную задачу, потому что он гарантирует решение.

Но сколько возможных решений существует для общей задачи коммивояжера для n городов? Немного поразмыслив, мы получаем $(n-1)!$ возможных решений. Это число растет очень быстро, и при скромных значениях n вычислить все возможные решения не представляется возможным. Предположим, предпринимателю необходимо посетить по одному городу в каждом из 50 штатов США. Число возможных решений равняется $49! = 6.1 \times 10^{62}$. Могут ли современные компьютеры рассчитать такое число возможных решений? Вселенной около 15 миллиардов лет, что составляет 4.7×10^{17} секунд. Предположим, что триллион компьютеров работает с начала возникновения Вселенной, и предположим, что каждую секунду каждый из этих триллионов компьютеров может рассчитать расстояние для триллиона возможных решений. Тогда мы рассчитали бы расстояние в общей сложности для 4.7×10^{41} возможных решений. Мы бы даже не поцарапали поверхность решения задачи коммивояжера для 50 городов.

Вот еще один способ взглянуть на вычислительную сложность задачи коммивояжера. На Земле имеется примерно от 10^{20} до 10^{24} песчинок [Welland, 2009]. Если бы каждая песчинка на Земле была планетой, похожей на Землю, с тем же количеством песка, что и Земля, то число возможных маршрутов в задаче коммивояжера для 50 городов все равно было бы намного больше, чем общее число мини-зерен песка. Совершенно очевидно, что невозможно решить такого рода крупную задачу методом исчерпывающего поиска.

Мы видим, что некоторые задачи настолько масштабны, что применение подхода на основе грубой силы просто невозможно. Кроме того, комбинаторные задачи, такие как задача коммивояжера, не имеют не-

[2] На самом деле задача проще, чем кажется на первый взгляд. S_1 является обратным для S_6, и поэтому S_1 и S_6 имеют одинаковое общее расстояние. То же самое можно сказать и об S_2 и S_4, S_3 и S_5.

[3] Незамкнутая (открытая) задача коммивояжера – это задача посещения всех городов ровно один раз, не возвращаясь в первоначальный город.

прерывных независимых переменных и поэтому не могут быть решены производными. Несмотря на это, нет уверенности в том, что у нас есть лучшее решение комбинаторной задачи, если мы не попробуем все возможные решения, эволюционные алгоритмы обеспечивают мощный способ отыскания хороших решений. Эволюционные алгоритмы не являются волшебством, но они способны помочь найти хотя бы хорошее решение (если вообще не лучшее решение) для таких типов крупных многомерных задач. Потенциальные решения в эволюционном алгоритме обмениваются информацией друг с другом и в итоге приходят к «консенсусу» в отношении лучшего решения. Мы не можем доказать, что это лучшее решение; нам пришлось бы перебрать все возможные решения, для того чтобы доказать, что мы нашли лучшее. Однако если сравнить решения эволюционного алгоритма с другими типами решений, то мы увидим, что эволюционные алгоритмы работают неплохо. В главе 18 более подробно рассматриваются эволюционные комбинаторно-оптимизационные задачи.

2.6. Восхождение к вершине холма

В этом разделе представлен простой оптимизационный алгоритм под названием «восхождение к вершине холма» (hill climbing). На самом деле восхождение к вершине холма — это семейство алгоритмов со многими вариациями. Некоторые исследователи считают восхождение к вершине холма простым эволюционным алгоритмом, в то время как другие считают его неэволюционным алгоритмом. Когда мы сталкиваемся с новой оптимизационной задачей, восхождение к вершине холма часто является хорошим первым вариантом ее решения, потому что этот алгоритм прост, удивительно эффективен, имеет ряд простых вариаций и обеспечивает хороший эталон, с которым могут сравниваться более сложные алгоритмы, в частности эволюционные. Идея восхождения к вершине холма настолько проста, что она, должно быть, была изобретена много раз, и очень давно, поэтому трудно определить ее происхождение.

Если вы хотите добраться до самой высокой точки ландшафта, то разумной стратегией будет просто сделать шаг в сторону самого крутого подъема. После этого шага вы заново оцениваете склон холма и снова делаете шаг в сторону самого крутого подъема. Этот процесс продолжается до тех пор, пока не будет никаких направлений, которые ведут вас выше, и, значит, в данной точке вы достигли вершины холма.

Эта стратегия является стратегией локального поиска, и называется она восхождением к вершине холма.

Более качественной стратегией будет осмотреться, оценить, где находится самая высокая точка, а затем рассчитать лучший способ туда добраться. Это устранит проблему зигзагов на пути к вершине или застревания на вершине небольшого холма, который находится ниже глобально самой высокой точки. Но если видимость низкая, то стратегия локального поиска может быть лучшим вариантом действий.

Восхождение к вершине холма может работать хорошо, а может и нет. Все зависит от формы холма, числа локальных максимумов и вашей исходной позиции. Восхождение к вершине холма может быть использовано само по себе в качестве оптимизационного алгоритма. Его также можно сочетать с эволюционным алгоритмом, и в этом случае будут сочетаться присущая эволюционному алгоритму возможность глобального поиска с возможностью локального поиска. Существует несколько разновидностей стратегий восхождения к вершине холма [Mitchell, 1998], некоторые из которых мы обсудим в этом разделе.

На рис. 2.6 показан алгоритм крутого восхождения к вершине холма (steepest ascent hill climbing). Этот алгоритм действует консервативно, за раз изменяя только один признак решения и заменяя текущее лучшее решение лучшим однопризнаковым изменением.

$x_0 \leftarrow$ случайно сгенерированная особь
While not (критерий останова)
    Рассчитать приспособленность $f(x_0)$ для x_0
    For each признак $q = 1, \cdots, n$ решения
        $x_q \leftarrow x_0$
        Заменить q-й признак решения в x_q случайной мутацией.
        Вычислить приспособленность $f(x_q)$ для x_q
    Next признак решения
    $x' \leftarrow \arg\max x_q(f(x_q) : q \in [0, n])$
    If $x_0 = x'$ then
        $x_0 \leftarrow$ случайно сгенерированная особь
    else
        $x_0 \leftarrow x'$
    End if
Next поколение

Рис. 2.6. Приведенный выше псевдокод дает схематичное описание алгоритма крутого восхождения к вершине холма для максимизации n-мерной функции $f(x)$. Обратите внимание, что x_q равен x_0, но при этом его g-й признак мутирован

На рис. 2.7 показан алгоритм следующего восхождения к вершине холма (next ascent hill climbing algorithm), также именуемый алгоритмом простого восхождения к вершине холма. Этот алгоритм, как и алгоритм крутого восхождения к вершине холма, за раз изменяет только один признак решения. Но алгоритм следующего восхождения к вершине холма более жадный, потому что текущее решение заменяется, как только найдено более качественное решение.

Следующие два алгоритма восхождения к вершине холма случайно отбирают, какой признак решения мутировать, и поэтому подпадают под общую классификацию стохастического восхождения к вершине холма. На рис. 2.8 показан алгоритм восхождения к вершине холма со случайной мутацией (random mutation hill climbing). Этот алгоритм очень похож на алгоритм следующего восхождения к вершине холма, за исключением того, что мутированный признак решения отбирается случайно.

$x_0 \leftarrow$ случайно сгенерированная особь
While not (критерий останова)
    Рассчитать приспособленность $f(x_0)$ для x_0
    ФлагЗамены ← false
    For each признак $q = 1, \cdots, n$ решения
        $x_q \leftarrow x_0$
        Заменить q-й признак решения для x_q на случайную мутацию.
        Рассчитать приспособленность $f(x_q)$ для x_q
        If $f(x_q) > f(x_0)$ **then**
            $x_q \leftarrow x_0$
            ФлагЗамены ← true
        End if
    Next признак решения
    If not (ФлагЗамены)
        $x_0 \leftarrow$ случайно сгенерированная особь
    End if
Next поколение

Рис. 2.7. Приведенный выше псевдокод дает схематичное описание алгоритма следующего восхождения к вершине холма для максимизации n-мерной функции $f(x)$. Обратите внимание, что x_q равен x_0, но при этом его g-й признак мутирован

$x_0 \leftarrow$ случайно сгенерированная особь
While not (критерий останова)
    Рассчитать приспособленность $f(x_0)$ для x_0
    $q \leftarrow$ индекс случайно отобранного признака решения $\in [1, n]$
    $x_1 \leftarrow x_0$

```
        Заменить g-й признак решения в x_1 на случайную мутацию
        Рассчитать приспособленность f(x_1) для x_1
        If f(x_1) > f(x_0)) then
            x_0 ← x_1
        End if
    Next поколение
```

Рис. 2.8. Приведенный выше псевдокод дает схематичное описание алгоритма восхождения к вершине холма со случайной мутацией для максимизации n-мерной функции $f(x)$. Обратите внимание, что x_i равно x_0, но с мутированным случайным признаком

На рис. 2.9 показан алгоритм адаптивного восхождения к вершине холма (adaptive hill climbing). Этот алгоритм похож на алгоритм восхождения к вершине холма со случайной мутацией, за исключением того, что каждый признак решения мутируется с некоторой вероятностью, прежде чем мутированное решение сравнивается с текущим лучшим решением.

```
Инициализировать p_m ∈ [0,1] в качестве вероятности мутации
x_0 ← случайно сгенерированная особь
While not (критерий останова)
    Рассчитать приспособленность f(x_0) для x_0
    x_1 ← x_0
    For each признак q = 1, ···, n решения
        Сгенерировать равномерно распределенное случайное число r ∈ [0,1]
        If r < p_m then
            Заменить q-й признак решения для x_1 на случайную мутацию
        End if
    Next признак решения
    Рассчитать приспособленность f(x_1) для x_1
    If f(x_1) > f(x_0) then
        x_0 ← x_1
    End if
Next поколение
```

Рис. 2.9. Приведенный выше псевдокод дает схематичное описание алгоритма адаптивного восхождения к вершине холма для максимизации n-мерной функции $f(x)$. Обратите внимание, что x_q равен x_0, но при этом его g-й признак мутирован

Результаты алгоритма восхождения к вершине холма могут сильно зависеть от исходного условия x_0 на рис. 2.6–2.9. Поэтому имеет смысл пробовать восхождение к вершине холма с несколькими разными случайно сгенерированными исходными условиями. Этот подход с раз-

мещением алгоритма восхождения к вершине холма внутрь цикла с исходным условием называется восхождением к вершине холма со случайным перезапуском.

■ Пример 2.7

Мы симулируем четыре алгоритма восхождения к вершине холма на множестве 20-мерных эталонных задач (см. приложение С.1). Обратите внимание, что эталонные задачи в приложении С.1 представляют собой минимизационные задачи, и поэтому мы адаптируем алгоритмы восхождения к вершине холма прямолинейным способом, получая алгоритмы спуска с вершины холма. Мы выполняем каждый алгоритм на каждом эталоне 50 раз, всякий раз с разным исходным условием. Для адаптивного восхождения к вершине холма мы используем $p_m = 0.1$. Мы завершаем каждый алгоритм восхождения к вершине холма после 1000 вычислений функции приспособленности.

Результаты представлены в табл. 2.1. Обратите внимание, что алгоритм крутого восхождения к вершине холма (рис. 2.6) в каждом поколении требует, по крайней мере, n оцениваний функции приспособленности (в наших примерах $n = 20$), в то время как восхождение к вершине холма со случайной мутацией (рис. 2.8) в каждом поколении требует только одного оценивания функции приспособленности. Подавляющее большинство вычислительных усилий эвристического алгоритма обычно расходуется на оценивание функций приспособленности (см. главу 21). Следовательно, в целях справедливого сравнения разных оптимизационных алгоритмов число оцениваний функции приспособленности должно быть для сравниваемых алгоритмов одинаковым. Если разные оптимизационные алгоритмы для каждого поколения имеют одинаковое число оцениваний функции приспособленности (например, крутое восхождение к вершине холма на рис. 2.6 и следующее восхождение к вершине холма на рис. 2.7), то было бы справедливо сравнивать эти алгоритмы на основе числа поколений.

Таблица 2.1 показывает, что восхождение к вершине холма со случайной мутацией лучше всего работает на 12 из 14 эталонов, и в среднем оно намного превосходит другие методы восхождения к вершине холма. Адаптивное восхождение к вершине холма и крутое восхождение к вершине холма на одном эталоне работают слегка лучше, чем восхождение к вершине холма со случайной мутацией. Вместе с тем результативность адаптивного восхождения к вершине холма может сильно зависеть от скорости мутации (см. задачу 2.11), и в данном примере мы не прилагали никаких усилий, чтобы найти хорошую скорость мутации.

Таблица 2.1. Пример 2.7: относительная результативность алгоритмов восхождения к вершине холма. В таблице показан нормализованный минимум, найденный четырьмя алгоритмами восхождения к вершине холма, усредненно по 50 симуляциям Монте-Карло. Поскольку восхождение к вершине холма является стохастическим, ваши результаты могут отличаться

Эталон	Крутое восхождение	Следующее восхождение	Случайная мутация	Адаптивный
Ackley	2.27	1.82 1.	00	1.70
Fletcher	2.62	1.87	1.00	1.68
Griewank	9.58	4.41	1.00	3.81
Penalty #1	26624	2160	1.00	281
Penalty #2	99347	5690	1.00	4178
Quartic	133.94	29.99	1.00	25.61
Rastrigin	3.76	2.52	1.00	2.10
Rosenbrock	2.68	1.50	1.00	1.72
Schwefel 1.2	1.63	1.37	1.24	1.00
Schwefel 2.21	1.00	1.75	1.02	1.12
Schwefel 2.22	3.65	2.73	1.00	2.30
Schwefel 2.26	5.05	3.63	1.00	2.91
Sphere	17.97	7.32	1.00	6.09
Step	16.58	6.78	1.00	6.52
Среднее значение	**9012**	**565**	**1.02**	**323**

■

2.6.1. Смещенные оптимизационные алгоритмы

Теперь мы упомянем один важный нюанс, связанный с эталонными функциями, который мы должны иметь в виду во время наших оптимизационных исследований. Этот нюанс включает в себя два связанных утверждения: во-первых, многие эталонные стоимостные функции имеют минимумы вблизи середины поисковой области; и во-вторых, многие оптимизационные алгоритмы смещены к середине поисковой

области[4]. Мы более подробно обсудим этот феномен смещенности в приложении С.7, которое призываем читателя внимательно изучить, прежде чем выполнять какие-либо серьезные исследования. Многие результаты симулирования в этой книге основаны на эталонах с минимумами в центре поисковой области. Эти результаты не претендуют на то, что будут точно отображать результативность оптимизационного алгоритма, а предназначены лишь для иллюстрации применения конкретного оптимизационного алгоритма. Прежде чем мы сможем смело делать выводы об оптимизационном алгоритме, мы должны реализовать несмещенный подход из приложения В.7.

2.6.2. Важность симуляций Монте-Карло

Обратите внимание, что в примере 2.7 мы усреднили 50 симуляционных результатов для построения графика результативности бинарного генетического алгоритма и непрерывного генетического алгоритма. Вывод результатов одиночной симуляции ничего не доказывает, поскольку результаты зависят от генератора случайных чисел. Из одиночной симуляции или из одиночного эксперимента мы можем получить очень мало обоснованных заключений. Мы обсудим эту тему более подробно в приложении В, но также упоминаем об этом здесь, поскольку это первое место в данной книге, где мы усреднили результаты нескольких симуляций.

Многократные симуляции, которые применются для анализа результативности, часто называются симуляциями Монте-Карло. Это название в 1940-х годах ввели Джон фон Нейман, Станислав Улам и Николас Метрополис во время своей работы над ядерным оружием. Подавляющая часть их работы включала анализ результатов многочисленных экспериментов; при этом была отмечена очевидная связь между статистическим анализом их экспериментов и статистическими свойствами азартных игр. Данная связь в сочетании с тем, что дядя Улама был задлым картежником в казино Монте-Карло в Монако, привела к названию «симуляции Монте-Карло» [Metropolis, 1987].

2.7. Интеллект

На заре вычислительной техники исследователи поняли, что компьютеры очень хороши в вещах, которые люди выполняют слабо, например

[4] Это не обязательно относится к алгоритмам восхождения к вершине холма данного раздела. Однако это относится ко многим эволюционным алгоритмам, которые мы обсудим в данной книге позже.

в вычислении траектории баллистической ракеты. Однако компьютеры были (и по-прежнему остаются) неэффективными в задачах, которые люди могут делать хорошо, например распознавать лицо. Это привело к попыткам имитировать биологическое поведение в надежде сделать компьютеры в таких задачах лучше. Результатом подобных усилий стали такие технологии, как нечеткие системы, нейронные сети, генетические алгоритмы и другие эволюционные алгоритмы. Поэтому эволюционные алгоритмы считаются частью общей категории компьютерного интеллекта.

При разработке эволюционного алгоритма мы стараемся создавать алгоритмы, которые обладают признаками разума. Но что значит быть разумным? Означает ли это, что наши эволюционные алгоритмы могут набрать высокий балл в тесте IQ? В этом разделе рассматривается значение интеллекта и некоторые его свойства: приспособляемость, случайность, общение, обратная связь, разведывание и эксплуатация. Эти свойства мы реализуем в эволюционных алгоритмах при поиске интеллектуальных алгоритмов.

2.7.1. Приспособляемость

Мы обычно рассматриваем приспособляемость к изменяющимся условиям как признак интеллекта. Предположим, вы учитесь собирать деталь, а затем ваш руководитель просит вас собрать приспособление, которое вы никогда раньше не видели. Если вы разумны, то сможете обобщить то, что знаете о деталях, и у вас получится собрать данное приспособление. Однако если вы не настолько разумны, то вас придется научить конкретным премудростям сборки приспособления.

Вместе с тем адаптивные регуляторы [Äström и Wittenmark, 2008] не считаются интеллектуальными. Вирус, который может выживать в экстремальных условиях, не считается интеллектуальным. Следовательно, мы приходим к выводу, что приспособляемость является необходимым, но не достаточным условием для интеллекта. Мы пытаемся конструировать эволюционные алгоритмы, которые можно приспосабливать к широкому классу задач. Наличие приспосабливаемости в эволюционном алгоритме является лишь одним из многих критериев успешного эволюционного алгоритма.

2.7.2. Случайность

Мы обычно думаем о случайности в отрицательных терминах. Нам не нравится непредсказуемость в нашей жизни, и поэтому мы стара-

емся ее избегать, и мы пытаемся контролировать нашу окружающую среду. Однако некоторая степень случайности является необходимым компонентом интеллекта. Подумайте о зебре, которая убегает ото льва. Если зебра будет бежать по прямой и с постоянной скоростью, то ее будет легко поймать. Но разумная зебра, для того чтобы уйти от хищника, будет двигаться зигзагообразно и непредсказуемо. И наоборот, подумайте обо льве, который пытается поймать зебру. Лев каждый день преследует стадо зебр. Если лев каждый день поджидает в одном и том же месте и в одно и то же время, то от него будет легко уйти. Но разумный лев будет наносить удар в разных местах и в разное время и непредсказуемым образом. Случайность – это свойство интеллекта.

Слишком много случайностей – это контрпродуктивно. Если зебра примет случайное решение лечь во время погони, то мы будем правы, подвергнув сомнению ее интеллект. Если же лев примет случайное решение выкопать яму в поисках зебры, то мы будем правы, подвергнув сомнению и его интеллект. Таким образом, случайность является свойством интеллекта, но только в пределах ограничений.

Наши проекты эволюционных алгоритмов будут включать в себя некоторые компоненты случайности. Если мы исключим случайность, то наши эволюционные алгоритмы не будут работать хорошо. Но если мы будем применять слишком много случайности, то они тоже не будут хорошо работать. В наших проектах эволюционных алгоритмов мы должны применять нужную величину случайности. Разумеется, как обсуждалось ранее, эволюционные алгоритмы могут приспосабливаться. Следовательно, хорошие эволюционные алгоритмы будут хорошо работать на целом ряде мер случайности. Мы не можем рассчитывать, что эволюционные алгоритмы будут настолько приспосабливаемы, что мы сможем использовать любой уровень случайности, но они будут приспосабливаться в достаточной мере, благодаря чему обязательное наличие точной меры случайности не станет критичным.

2.7.3. Общение

Общение – это свойство интеллекта. Рассмотрим гения, который проходит тест на IQ, за тем исключением, что гений не имеет никакого способа общаться. Он провалит тест IQ, даже если он гений. Многие глухонемые и аутисты проваливают тест на IQ, несмотря на то что они вполне разумны. Дети, которые растут без человеческого взаимодействия, не являются творческими, умными, веселыми и хорошо адаптирующимися личностями [Newton, 2004). Отсутствие общения с другими

людьми в годы их становления не позволяет им развивать интеллектуальные способности, выходящие за пределы маленького ребенка. Их годы изоляции невосполнимы, и они не могут научиться общаться или адаптироваться к обществу.

Интеллект не только включает в себя общение, но он также *эмергентен*, то есть проявляется неожиданно, стихийно. Иными словами, разум появляется из популяции особей. Одна особь не может быть разумной. Можно утверждать, что в мире есть много разумных особей, и даже если бы такая особь была изолирована, она все равно была бы разумной. Тем не менее такие особи получили свой интеллект только благодаря взаимодействию с другими. Один муравей бродит бесцельно и ничего не добивается, но муравьиная куча может найти кратчайший путь к пище, построить сложные сети туннелей и организовать себя как самоподдерживающееся сообщество. Точно так же одна особь никогда ничего не достигнет, если она никогда не будет взаимодействовать с сообществом. Сообщество же может отправить человека на Луну, соединить миллиарды людей через интернет и построить системы питания в пустыне.

Мы видим, что интеллект и общение образуют положительный цикл обратной связи. Общение необходимо для развития интеллекта, а интеллект необходим для общения. Но главное здесь в том, что общение – это свойство интеллекта. Вот почему большинство эволюционных алгоритмов включает в себя популяцию кандидатных решений той или иной задачи. Эти кандидатные решения, которые мы называем *особями*, общаются друг с другом и учатся на успехах и неудачах друг друга. Со временем популяция особей эволюционирует и становится хорошим решением оптимизационной задачи.

2.7.4. Обратная связь

Обратная связь является фундаментальным свойством интеллекта. Она предполагает приспособляемость, о которой говорилось выше. Система не может приспособиться, если она не может чувствовать и реагировать на окружающую среду. Однако обратная связь предусматривает не только приспособляемость, но и самообучение. Когда мы делаем ошибки, то меняемся, чтобы не повторять эти ошибки[5]. Однако, что еще более важно, когда ошибки совершают другие, мы корректируем свое собственное поведение, чтобы не повторять ошибки других. Неуспех обеспечивает отрицательную обратную связь. И наоборот, успех (наш и других) обеспечивает положительную обратную связь и оказывает на

[5] Альберт Эйнштейн, как считается, дал определение безумию как делание того же самого снова и снова, ожидая при этом получать разные результаты.

нас влияние тем, чтобы мы приняли те линии поведения, к которым относим успех. Мы часто видим других, которые, кажется, не учатся на ошибках и которые не принимают линии поведения, которые, как оказалось, приводят к успеху; мы не считаем таких людей очень разумными.

Обратная связь также является основой многих природных явлений. Круговорот воды состоит из бесконечной последовательности дождевых осадков и испарения. Больший объем осадков ведет к увеличению испарения, а испарения приводят к большему объему дождевых осадков. Поскольку это влечет за собой фиксированный объем воды, круговорот воды приводит к стабильному объему влаги на поверхности земли и в небе. Если бы этот механизм обратной связи был каким-то образом нарушен, то возникло бы много трудностей для жизни, включая наводнения и засухи.

Баланс сахара и инсулина в человеческом теле – это еще один механизм обратной связи. Чем больше сахара мы потребляем, тем больше инсулина вырабатывает поджелудочная железа; чем больше инсулина вырабатывает поджелудочная железа, тем больше сахара всасывается из крови. Слишком много сахара в крови приводит к гипергликемии, а слишком мало сахара в крови приводит к гипогликемии. Диабет является нарушением механизма обратной связи сахар/инсулин и может привести к серьезным и долгосрочным проблемам со здоровьем.

Такая характеристика обратной связи, как свойства интеллекта, часто признается в теории интеллектуального управления. Обратная связь не является достаточным условием для интеллекта. Никто не назовет пропорциональный контроллер разумным, так же как и механический термостат. Обратная связь является необходимым, но недостаточным условием для интеллекта.

Наши проекты эволюционных алгоритмов будут включать положительную и отрицательную обратную связь. Эволюционный алгоритм без обратной связи будет не очень эффективным, а вот эволюционный алгоритм с обратной связью будет удовлетворять этому необходимому условию наличия интеллекта.

2.7.5. Разведывание и эксплуатация

Разведывание – это поиск новых идей или новых стратегий. *Эксплуатация* – это использование существующих идей и стратегий, которые оказались успешными в прошлом. Разведывание сопряжено с высоким риском; многие новые идеи приводят к пустой трате времени и тупи-

кам. Тем не менее разведывание также может приносить высокую отдачу; много новых идей окупаются способами, которые мы не могли себе представить. Эксплуатация тесно связана с обсуждавшимися ранее стратегиями обратной связи. Разумный человек использует то, что он знает и что имеет, вместо того чтобы постоянно изобретать колесо. Но разумный человек также открыт для новых идей и готов пойти на просчитанный риск. Интеллект включает в себя надлежащий баланс разведывания и эксплуатации. Правильный баланс разведывания и эксплуатации зависит от того, насколько регулярна наша окружающая среда. Если наша окружающая среда быстро меняется, то наши знания быстро устаревают, и мы не можем во многом полагаться на эксплуатацию. Однако если наша окружающая среда весьма последовательна, то наше знание надежно и, возможно, не будет иметь смысла в том, чтобы испытывать слишком много новых идей.

В наших проектах эволюционных алгоритмов нам необходим надлежащий баланс между разведыванием и эксплуатацией. Слишком много разведывания похоже на слишком много случайности, о которой мы говорили ранее, и оно, вероятно, не даст хороших результатов оптимизации. Вместе с тем слишком много эксплуатации связано со слишком малой случайностью. Джон Холланд (John Holland), один из пионеров генетических алгоритмов, назвал правильный баланс между разведыванием и эксплуатацией в эволюционных алгоритмах «оптимальным распределением испытаний» [Holland, 1975].

2.8. Заключение

Ключевой момент данной главы состоит в том, что оптимизация является фундаментальным аспектом проектирования и решения задач. Когда мы пытаемся оптимизировать функцию, мы называем ее целевой функцией. Когда мы пытаемся минимизировать функцию, мы называем ее функцией стоимости. Когда мы пытаемся максимизировать функцию, мы называем ее функцией приспособленности. Любая оптимизационная задача может быть легко преобразована в обратную и перетекать между задачей минимизации и задачей максимизации. Некоторые специальные типы задач, с которыми мы познакомились в этой главе, являются ограниченными задачами, многокритериальными задачами и мультимодальными задачами. Почти все реальные оптимизационные задачи являются ограниченными, многокритериальными и мультимодальными. Еще один специальный класс задач – это ком-

бинаторные задачи, в которых независимые переменные принадлежат конечному множеству.

В этой главе мы представили простой, но эффективный оптимизационный алгоритм – восхождение к вершине холма. Существует много разных типов алгоритмов восхождения к вершине холма. Несмотря на то что они, как правило, довольно просты, они обеспечивают хороший эталон, с которым можно сравнивать более сложные оптимизационные алгоритмы. Наконец, мы упомянули несколько признаков естественного интеллекта и обсудили, как реализовывать эти признаки в наших эволюционных алгоритмах, для того чтобы оправдать обозначение *разумный*.

Задачи

Письменные упражнения

2.1. Рассмотрим задачу min $f(x)$, где

$$f(x) = 40 + \sum_{i=1}^{4} x_i^2 - 10 \cos(2\pi x_i).$$

Обратите внимание, что $f(x)$ является функцией Растригина – см. раздел C.1.11.

a) Каковы независимые переменные $f(x)$? Каковы переменные решения $f(x)$? Каковы признки решения $f(x)$?

b) Какова размерность этой задачи?

в) Каково решение этой задачи?

d) Перепишите эту задачу как задачу максимизации.

2.2. Рассмотрим функцию $f(x) = \sin x$.

a) Сколько у функции $f(x)$ локальных минимумов? Каковы значения функции в локальных минимумах, и каковы локально минимизирующие значения x?

b) Сколько у функции $f(x)$ глобальных минимумов? Каковы значения функции в глобальных минимумах, и каковы глобально минимизирующие значения x?

2.3. Рассмотрим функцию $f(x) = x^3 + 4x^2 - 4x + 1$.

a) Сколько у функции $f(x)$ локальных минимумов? Каковы значения функции в локальных минимумах, и каковы локально минимизирующие значения x?

b) Сколько у функции $f(x)$ локальных максимумов? Каковы значения функции в локальных максимумах, и каковы локально максимизирующие значения x?

c) Сколько у функции $f(x)$ глобальных минимумов?

d) Сколько у $f(x)$ глобальных максимумов?

2.4. Рассмотрим ту же функцию, что и в задаче 2.3, $f(x) = x^3 + 4x^2 - 4x + 1$, но с ограничением $x \in [-5, 3]$.

a) Сколько у функции $f(x)$ локальных минимумов? Каковы значения функции в локальных минимумах, и каковы локально минимизирующие значения x?

b) Сколько у функции $f(x)$ локальных максимумов? Каковы значения функции в локальных максимумах, и каковы локально максимизирующие значения x?

c) Сколько у функции $f(x)$ глобальных минимумов? Каково значение функции в глобальном минимуме, и каковы глобально минимизирующие значения x?

d) Сколько у функции $f(x)$ глобальных максимумов? Каково значение функции в глобальном максимуме, и каковы глобально максимизирующие значения x?

2.5. Напомним, что на рис. 2.4 показан фронт Парето для двукритериальной задачи, в которой поставлена задача минимизировать оба целевых критерия.

a) Нарисуйте возможное множество точек на плоскости (f, g) и фронте Парето для задачи, в которой целевой критерий состоит в максимизации $f(x)$ и минимизации $g(x)$.

b) Нарисуйте возможное множество точек на плоскости (f, g) и фронте Парето для задачи, в которой целевым критерием является минимизация $f(x)$ и максимизация $g(x)$.

c) Нарисуйте возможное множество точек на плоскости (f, g) и фронте Парето для задачи, в которой целевым критерием является максимизация как $f(x)$, так и $g(x)$.

2.6. Сколько уникальных замкнутых путей существует через N городов? Под уникальным мы подразумеваем, что первоначальный город и направление движения не имеют значения. Например,

в задаче для 4 городов с городами A, B, C и D мы рассматриваем маршрут $A \to B \to C \to D \to A$ как эквивалентный маршрутам $D \to C \to B \to A \to D$ и $B \to C \to D \to A \to B$.

2.7. Рассмотрим замкнутую задачу коммивояжера для городов в табл. 2.2.

a) Сколько существует замкнутых маршрутов через эти семь городов?

b) Легко ли увидеть решение, посмотрев на координаты в табл. 2.2?

Таблица 2.2. Координаты городов для задачи 2.7

Город	*x*	*y*
A	5	9
B	9	8
C	-6	-8
D	9	-2
E	-5	9
F	4	-7
G	-9	1

c) Постройте график координат. Легко ли увидеть решение из графика? Каково оптимальное решение? Эта задача показывает, что другой взгляд на задачу, возможно, поможет нам найти решение.

2.8. Дана произвольная задача максимизации $f(x)$ и случайное исходное кандидатное решение x_0. Какова вероятность того, что алгоритм крутого восхождения к вершине холма найдет значение x' – такое, что $f(x') > f(x_0)$ после первого поколения?

Компьютерные упражнения

2.9. Постройте график функции задачи 2.4 с четко обозначенными локальными и глобальными оптимумами.

2.10. Рассмотрим многокритериальную оптимизационную задачу $\min \{f_1, f_2\}$, где

$$f_1(x_1, x_2) = x_1^2 + x_2 \text{ и } f_2(x_1, x_2) = x_1 + x_2^2$$

x_1 и x_2 ограничены [–10, 10].

a) Вычислите $f_1(x_1, x_2)$ и $f_2(x_1, x_2)$ для всех допустимых целочисленных значений x_1 и x_2 и постройте точки в пространстве (f_1, f_2) (в общей сложности $21^2 = 441$ точка). Четко укажите фронт Парето на графике.

b) Дана разрешающая способность, которую вы использовали в части (a). Дайте математическое описание множества Парето. Постройте график множества Парето в пространстве (x_1, x_2).

2.11. Адаптивное восхождение к вершине холма.

a) Выполните 20 симуляций Монте-Карло алгоритма адаптивного восхождения к вершине холма с 1000 поколений на симуляцию Монте-Карло для двумерной функции Экли. Запишите минимальное значение, полученное при каждой симуляции Монте-Карло, и вычислите среднее значение. Сделайте это для 10 разных скоростей мутации, $p_m = k/10$ для $k \in [1, 10]$, и запишите свои результаты в табл. 2.3. Какова лучшая скорость мутации?

Таблица 2.3. Заполните эту таблицу для задачи 2.11.

P_m	Средний результат
0.1	
0.2	
0.3	
0.4	
0.5	
0.6	
0.7	
0.8	
0.9	
1.0	

b) Повторите часть (a). Получаете ли вы такие же или подобные результаты? Какой вывод вы делаете о числе симуляций

Монте-Карло, необходимых для получения воспроизводимых результатов для этой задачи?

c) Повторите часть (а) для 10-мерной функции Экли. Какой вывод вы делаете о взаимосвязи между оптимальной скоростью мутации и размерностью задачи?

Часть II

Классические эволюционные алгоритмы

Глава 3

Генетические алгоритмы

Генетические алгоритмы не являются оптимизаторами функций.

– Кеннет Де Йонг [De Jong, 1992]

Генетические алгоритмы (genetic algorithm, GA) – это самые ранние, наиболее известные и наиболее широко используемые эволюционные алгоритмы. Генетические алгоритмы представляют собой симуляции естественного отбора, которые могут решать оптимизационные задачи. Несмотря на приведенную выше цитату Кеннета Де Йонга, генетические алгоритмы часто служат эффективным инструментом оптимизации. Цитата Де Йонга подчеркивает тот факт, что генетические алгоритмы изначально разрабатывались для изучения адаптивных систем, а не для оптимизации функций. Генетические алгоритмы являются гораздо более широким классом систем, чем оптимизаторы функций. Мы можем применять генетические алгоритмы для изучения динамики адаптивных систем [Mitchell, 1998, глава 4], для предоставления консультаций модельерам-конструкторам [Kim и Cho, 2000], для обеспечения дизайнерских компромиссов с целью сопряжения дизайнеров [Furuta и соавт., 1995] и для многих других неоптимизационных приложений. Иногда разделительная линия между оптимизационными и неоптимизационными алгоритмами становится нечеткой, потому что все алгоритмы пытаются функционировать как можно лучше. В любом случае, наш основной интерес к генетическим алгоритмам в этой книге лежит в их специфическом применении в качестве оптимизационных алгоритмов. В самом начале нашего генетико-алгоритмического исследования давайте рассмотрим некоторые основные признаки естественного отбора.

1. Биологическая система содержит популяцию особей, многие из которых обладают способностью к размножению.
2. Продолжительность жизни особей ограничена.
3. В популяции существует изменчивость, то есть вариация.
4. Способность к выживанию положительно коррелирует со способностью к размножению.

Генетические алгоритмы симулируют каждый из этих признаков естественного отбора. При наличии оптимизационной задачи мы создаем популяцию кандидатных решений, которые называются особями или индивидуумами. Некоторые решения хороши, а некоторые не так хороши. Хорошие особи имеют относительно высокие шансы на размножение, в то время как шансы на размножение у слабых особей относительно низкие. Родители рождают детей, а затем родители выбывают из популяции, освобождая место для своих детей[1]. С приходом и уходом поколений популяция становится более приспособленной. Иногда один или несколько «суперменов» эволюционируют, становясь очень приспособленными особями, которые могут обеспечить почти оптимальные решения нашей инженерной задачи.

Краткий обзор главы

В этой главе дается обзор естественной генетики, а также искусственных генетических алгоритмов для оптимизационных задач. Поскольку здесь мы только начинаем работу с эволюционными алгоритмами, то потратим в ней больше времени на историю и биологические основания, чем в большинстве последующих глав. Читатель, который хочет сразу перейти к изучению генетических алгоритмов, может спокойно пропустить первые три раздела, не подвергая серьезной опасности свое понимание генетических алгоритмов. В разделе 3.1 кратко рассматривается история науки генетики, сосредоточено внимание на работе Чарльза Дарвина и Грегора Менделя в XIX веке. В разделе 3.2 рассматривается наука генетика, которая составляет основу генетических алгоритмов. В разделе 3.3 представлена история компьютерного моделирования генетики начиная с 1940-х годов биологами, которые были заинтересованы в изучении естественного отбора, и заканчивая

[1] В оригинале использован термин «offspring», который в толковании на английском языке звучит как «a person's child or children», т. е. «ребенок или дети человека», и обозначает близкого родственника по прямой линии первой ступени. Не путать с термином «потомок», который обозначает близкого родственника по прямой линии второй и последующих ступеней, т. е. внук, правнук и т. д. – *Прим. перев.*

взрывным ростом генетико-алгоритмических исследований в 1970-х и 1980-х годах.

В разделе 3.4 методично, шаг за шагом, разрабатывается простой бинарный генетический алгоритм. Генетический алгоритм основан на естественной генетике, и поэтому он представляет собой решения оптимизационных задач как хромосомы с бинарными аллелями. Бинарный генетический алгоритм естественным образом подходит для оптимизационных задач, чья область состоит из n-мерного двоичного поискового пространства, либо, по крайней мере, для задач, чьи области дискретны.

Для представления кандидатных решений оптимизационных задач с непрерывными областями мы можем использовать битовые строки, при условии что для получения требуемой разрешающей способности мы используем достаточно бит. Но более естественным является представление кандидатных решений задач с непрерывной областью как векторов вещественных чисел. Поэтому раздел 3.5 расширит генетические алгоритмы на задачи с непрерывной областью.

3.1. История генетики

Генетика – это наука о закономерностях наследственности и изменчивости живых организмов. Данный раздел посвящен краткой истории развития современной генетики с акцентом на работе Чарльза Дарвина, родоначальника теории эволюции, и Грегора Менделя, родоначальника генетики.

3.1.1. *Чарльз Дарвин*

Чарльз Дарвин родился в Англии в 1809 году в семье богатого доктора Роберта Дарвина. Привилегированное положение Чарльза в жизни позволило ему в молодости блуждать от одного интереса к другому, по-видимому, обрекая себя провести всю свою жизнь в ленивых блужданиях. Его отец был трудолюбивым человеком, но, как это часто бывает, тяжелый труд отца приводит к лености сына. «Ты не заботишься ни о чем, кроме стрельбы, собак и ловли крыс; ты станешь позором для себя и всей своей семьи», – сказал Роберт своему сыну [Darwin и соавт., 2002, p. 10]. Роберт попытался вовлечь сына в свою медицинскую практику, но Чарльзу это было неинтересно, и к тому же он ненавидел вид крови. Поэтому Роберт отправил сына учиться пастырской службе в Кембриджский университет.

Чарльз на самом деле не интересовался учебой в Кембридже; единственное, что его интересовало, – это природа. Он проводил все свое время, исследуя и изучая природу, читая книги великих естествоиспытателей и собирая жуков. Его жизнь начала принимать некие очертания, когда он стал более опытным как натуралист. Чарльз начал встречаться с профессорами и другими студентами, которые разделяли его интерес к природе. Он начал строить планы на то, чтобы оставить пастырскую учебу и продолжить преследовать свою настоящую страсть в жизни. Он наконец-то становится амбициозным.

В 1831 году, в возрасте 22 лет, Чарльз подал заявку на должность на Бигль, корабль, который был направлен для обследования южной оконечности Южной Америки для английского правительства. Чарльз был принят на должность, при условии что он оплатит поездку.

Что бы вы сделали, если бы были отцом Чарльза Дарвина? Вы отправили сына в школу и заплатили за три года подготовки к пастырской службе, и теперь он приходит к вам с просьбой о средствах для оплаты пятилетнего морского путешествия в качестве натуралиста. Конечно, вы ответите «нет». И это именно то, что поначалу Роберт сказал своему сыну. К счастью для Чарльза, Роберт понял, что его сын становится мужчиной и нашел страсть в жизни. В конце концов, Роберт убедил себя в этом и согласился финансировать путешествие.

Во время пятилетнего путешествия Чарльз жил на корабле 90 футов длиной, 25 футов шириной с 70 другими моряками. Корабль также содержал геодезическое оборудование и достаточно припасов, чтобы продержаться в море несколько месяцев. Для Чарльза это был трудный переход от вольготной жизни, но он воспользовался этим по максимуму. Он провел время в море, занимаясь чтением и учебой. Когда корабль останавливался на островах или на южноамериканском материке, он собирал животных и отправлял их обратно в Англию на следующем доступном корабле. Во время своих путешествий он собрал огромное многообразие видов. Схожие виды на соседних островах отличались друг от друга настолько, что каждый, казалось, адаптировался к своей конкретной среде обитания.

Чарльз вернулся домой в Англию в 1836 году в возрасте 27 лет. Почти сразу же после возвращения в Англию он начал работать над своей книгой «Происхождение видов» [Darwin, 1859], которая в конечном итоге станет десятилетним проектом. Он также принимал активное участие в написании журнальных статей и выступлениях на конференциях. Он продолжал свои исследования и учебу, когда занялся формированием последовательной теории естественного отбора. Естественный отбор

говорит о том, что наиболее приспособленная особь (особи) выживает и передает свои индивидуальные особенности детям. Именно так происходит адаптация – через «выживание наиболее приспособленных».

Чарльз продолжал работать над своей книгой, но он не решался обнародовать свою теорию эволюции. Проучившись пастырской службе в течение трех лет, он знал, что его теория может вызвать бурю споров из-за её возможного противоречия с Библией. Прежде чем опубликовать свои результаты, он хотел построить каркас без единого слабого места и настоящий опус магнум. Однако в 1858 году он получил бумагу от Альфреда Уоллеса, натуралиста, который путешествовал в южной части Тихого океана. Уоллес самостоятельно пришел ко многим из тех же идей, что и Дарвин, и послал Дарвину статью с просьбой о помощи в её публикации.

Дарвин оказался в некоторой степени затруднительном положении. Он должен был сделать выбор. Он мог «потерять» работу Уоллеса[2], опубликовать свои собственные результаты и претендовать на приоритет своей теории эволюции. Или же он мог представить статью Уоллеса для публикации и позволить тому получить приоритет. К его чести, Дарвин решил попытаться найти сбалансированное решение. Он быстро написал свою собственную работу, а затем представил свою и Уоллеса работы на следующей доступной конференции. Потом он внес последние штрихи в свою книгу, которая оказалась намного короче, чем он первоначально намеревался[3], из-за его спешки получить признание, которого он заслуживал. Его труд «Происхождение видов» был опубликован в 1859 году, а первый тираж в 1250 экземпляров был распродан за один день. Дарвин был близок к тому, чтобы стать самым известным и спорным ученым своего поколения.

Хотя теория эволюции Дарвина быстро завоевала научное доверие, как и все новые теории, она не обошлась без недоброжелателей. Во-первых, это противоречило библейскому учению об особом сотворении всех видов и, следовательно, подверглось нападкам со стороны религиозных лидеров. Во-вторых, Дарвин не имел ни малейшего представления, как индивидуальные особенности передаются от родителей к их детям. В некотором смысле удивительно, что его теория получила признание –такое, какое она получила, несмотря на отсутствие объяснения

[2] Никто бы не поставил под сомнение утверждение Дарвина о том, что письмо, отправленное из южной части Тихого океана в Англию в 1858 году, так и не прибыло.

[3] В конечном счете все получилось к лучшему. Книга Дарвина имела около 500 страниц. Если бы он закончил её так, как он хотел, она бы имела на несколько сотен страниц больше и была бы сложнее для восприятия потенциальными читателями. Сделав книгу короче, он увеличил свою читательскую аудиторию и последующий успех.

механизма наследственности. Он отметил её наличие и поэтому постулировал естественный отбор, но не смог сказать, как она реализуется.

Дарвин наряду с другими учеными своего времени имел два фундаментальных заблуждения о механизме наследственности. Во-первых, он полагал, что индивидуальные особенности родителей могут быть перемешаны в их детях; например, отпрыски черной мыши и белой мыши, возможно, будут серыми. Во-вторых, он считал, что приобретенные признаки могут передаваться детям. Например, человек, который поднимает тяжести и становится сильным, будет иметь сильных детей благодаря поднятию тяжестей.

Дарвин, дитя привилегий, разработал теорию естественного отбора, но она была оставлена в наследство дитю бедности Грегору Менделю, который её доказал.

3.1.2. *Грегор Мендель*

Грегор Мендель первым понял и объяснил, как реализуется механизм наследственности. Он родился в семье бедного фермера Иоганна Менделя в 1822 году в Чехословакии [Bankston, 2005]. Его отец нуждался в нем в качестве помощника на ферме, но молодой Иоганн был больше пригоден к академическому, чем физическому труду. Хотя его родители едва могли себе это позволить, они отправили его в школу, чтобы помочь ему получить возможности в жизни, которых им не хватало. В студенческие годы, несмотря на поддержку родителей и работу неполный рабочий день, он едва сводил концы с концами. Его финансовое положение было почти таким же, как и у многих студентов-выпускников сегодня, за исключением того, что было меньше возможностей для финансовой помощи.

В возрасте 21 года Мендель узнал, что в соседнем монастыре есть возможность продолжить свое образование без каких-либо финансовых забот, но для этого ему придется принять обет бедности и безбрачия. Однако финансовые выгоды были слишком хороши, чтобы отказаться. Мендель не был особенно религиозным, но он ухватился за эту возможность и вступил в августинский орден в монастырь св. Фомы.

Августин, живший в Римской империи около 400 г. н. э., был одним из величайших интеллектуальных лидеров в христианской истории. Его богословие подчеркивало способность Бога общаться с человеком через светское знание. Монастырь, к которому присоединился Мендель, в соответствии с философией своего тезки поощрял обучение во всех сферах жизни и, таким образом, идеально подходил Менделю. Он был

«мирским», по сравнению со многими другими монастырями. Монахи не должны были наказывать себя, или проводить весь день в молитве, или принимать обеты молчания. Они обязаны были только учиться, веря, что Бог говорил с ними через их учебу. Когда Иоганн Мендель присоединился к монастырю Св. Фомы, в соответствии с традицией он дал себе новое имя; его имя теперь было Грегор Мендель.

Будучи монахом, Мендель продолжал учиться в университетах, преподавал науку в близлежащих школах, читал и проводил собственные исследования в монастыре. Проводимые им исследования включали разведение растений, в частности гороха. Эта деятельность была подходящим направлением для реализации его творческих талантов благодаря опыту работы на ферме своего отца.

Мендель проводил эксперименты с горохом и заметил, что горох имеет разные свойства. Некоторые виды были гладкими, в то время как другие были грубыми; некоторые были зеленее, в то время как другие были желтее; у некоторых горошины были в одном месте, у других горошины были в другом месте. Когда Мендель экспериментировал, он понял, что эти свойства контролируются какой-то невидимой единицей наследственности, которую он назвал элементами. Некоторые из элементов были сильными и, как правило, имели больше контроля над свойствами гороха. Другие элементы были слабыми и имели меньше контроля над свойствами гороха. Сегодня вместо слова «элемент» мы используем термин «ген» и говорим, что гены либо доминантные, либо рецессивные, а не сильные или слабые. Но именно Мендель впервые понял генетический механизм наследственности и доминантности. Работа Менделя была недостающим звеном в теории Дарвина и объяснила, как работает естественный отбор.

Мендель представил свои выводы на конференции в 1865 году. Прошло всего шесть лет после публикации Дарвином книги «Происхождение видов», но по какой-то причине Мендель не осознавал масштабов того, что он обнаружил. Восприятие научного прорыва Менделя оказалось совершенно иным, чем у Дарвина. В то время как Дарвин сразу стал знаменитым, работы Менделя были проигнорированы. Он продолжал работать в безвестности в Сент-Томасе, изредка публикуя свои статьи, все из которых были, по существу, упущены из виду научным сообществом.

Мендель стал во главе Сент-Томаса в 1868 году, и после этого у него уже не оставалось времени на науку[4]. Мендель умер в 1884 году. Его работы по генетике были, наконец, заново открыты голландским био-

[4] Как часто бывает в науке, процветающая карьера в области научных исследований была трагически прервана продвижением по административной линии.

логом Гюго де Врисом, немецким ботаником Карлом Корренсом и австрийским агрономом Эрихом фон Чермаком примерно в 1900 году.

3.2. Генетика

Все наши индивидуальные особенности, или признаки, контролируются парой генов. Человеческая генетика поэтому называется диплоидной, имея в виду, что гены для каждого признака появляются в парах. Некоторые растения и животные гаплоидны, то есть каждый признак определяется одним набором генов. Другие организмы являются полиплоидными, то есть каждый признак определяется более чем двумя наборами генов.

В диплоидной генетике некоторые генетические значения доминируют, в то время как другие рецессивны. Если доминантный и рецессивный гены оба появляются в особи, то доминантный ген определит признак, который появится у особи. Рецессивный ген может определить признак только в том случае, если оба гена одинаковы.

■ Пример 3.1

Рассмотрим трех человек: Крис имеет два гена карих глаз, Ким – два гена зеленых глаз, и Терри – ген карих глаз и ген зеленых глаз. Поскольку у Крис два гена карих глаз, у Крис карие глаза. Поскольку у Ким два гена зеленых глаз, у Ким зеленые глаза. Поскольку у Терри один ген карих глаз, а другой ген зеленых глаз, при этом ген карих глаз является доминатным, у Терри карие глаза.

- Крис: карие / карие → карие глаза
- Ким: зеленые / зеленые → зеленые глаза
- Терри: карие / зеленые → карие глаза

Если спариваются Крис и Терри, то каждый из них вносит свой ген цвета глаз в своих детей. Поэтому их дети могут иметь либо два гена карих глаз, либо один ген карих глаз и один ген зеленых глаз. Все их дети будут иметь карие глаза, так как карий является доминантным.

Если спариваются Крис и Ким, то у их детей будет один ген карих глаз от Криса и один ген зеленых глаз от Ким. Все их дети будут иметь карие глаза, так как карий является доминантным.

Если спариваются Терри и Ким, то их дети будут иметь либо один ген карих глаз и один ген зеленых глаз, либо два гена зеленых глаз. У их детей могут быть либо карие, либо зеленые глаза.

■

Теперь предположим, что существует некоторое эволюционное преимущество в наличии зеленых глаз. Например, самки могут испытывать сильное влечение к самцам с зелеными глазами. Или же, возможно, зеленые глаза более восприимчивы к определенным частотам света, что позволяет зеленоглазым особям быть более успешными в охоте. В этом случае зеленоглазые особи с большей вероятностью выживут, чем кареглазые. Более того, зеленые глаза будут, как правило, сильнее и успешнее, чем карие глаза, что предоставит им больше возможностей для размножения. Это повлияет на генофонд человеческой расы, увеличив число генов зеленых глаз при одновременном уменьшении числа генов карих глаз. Такой процесс называется естественным отбором, или выживанием наиболее приспособленных, и это то, что Дарвин вывел во время своего путешествия на Бигле в 1830-х годах.

Иногда на детей родителей влияет мутация. В этом случае гены не передаются от родителя к детям нетронутыми, а вместо этого изменяются. Мутации обусловлены фундаментальным несовершенством жизни и биологических процессов, включая радиацию и болезни.

Подавляющее большинство мутаций нейтрально; они практически не влияют на детей из-за поразительной устойчивости и избыточности биологии. Эти нейтральные мутации важны в поисках биологической жизнью улучшенной приспособленности, и мы увидим позже в этой книге, что мутации также обычно важны в поисках эволюционным алгоритмом улучшенной приспособленности. Однако большинство мутаций, которые заметно влияют на биологическое потомство, вредно. На самом деле можно сказать, что почти все подобные мутации вредны. Существует более чем 6000 часто встречающихся моногенных мутаций, которые вызывают болезни, и они происходят в одном из каждых 200 рождений. Некоторые генетические расстройства появляются при рождении, в то время как другие могут проявиться только позже. Например, многие виды рака имеют генетическую составляющую.

Однако время от времени появляется мутация, которая на самом деле полезна. Например, предположим, что в примере 3.1 один из ребенков Терри и Ким имеет мутацию, которая приводит к гену фиолетовых глаз. Предположим, что этот ребенок с фиолетовыми глазами имеет более острое зрение, чем в среднем, из-за корреляции фиолетовой радужной оболочки с более высокой адаптивностью роговицы. Это позволяет фиолетовоглазому мутанту быть более успешным в охоте. Он становится сильным и успешным благодаря своим фиолетовым глазам, и это дает ему больше возможностей для спаривания. Если его ген фиолетовых глаз доминантен, то все его дети будут иметь фиолетовые глаза, и мута-

ция будет распространяться по всему виду. Если же его ген фиолетовых глаз рецессивен, то у него могут быть дети с фиолетовыми глазами в том случае, если он найдет другую мутацию фиолетовых глаз для спаривания. Мутация, за которой следует естественный отбор, способствует повышению живучести вида. Без мутации вид стал бы застойным. Мутация, как правило, вредна для особей, но по иронии судьбы она полезна для всего вида в целом.

3.3. История генетических алгоритмов

1903 год был хорошим годом для технологий. Компания Marconi начала первую регулярную трансатлантическую радиопередачу, братья Райт успешно завершили свой первый полет на самолете, а Янош Нейман родился в Будапеште, Венгрия. Гений Неймана проявил себя в молодом возрасте в его ненасытном чтении и математических способностях. Его родители, оба из которых были из образованных семей высшего класса, рано признали, что он был вундеркиндом, но они были осторожны в том, чтобы не заставлять его слишком сильно. К тому времени, когда ему было 23 года, он уже имел степень бакалавра в области химической инженерии и был доктором философии по математике. Он продолжил продуктивно работать в качестве университетского преподавателя и в 1929 году принял должность преподавателя в Принстонском университете в Нью-Джерси. Его имя теперь было Джон фон Нейман, и в 1933 году он стал одним из первых членов Института перспективных исследований Принстона.

Во время своей обширной карьеры в Принстоне фон Нейман сделал фундаментальный вклад в математику, физику и экономику. Он был одним из лидеров проекта по разработке атомной бомбы во время Второй мировой войны, а также одним из пионеров изобретения цифрового компьютера. В развитии цифровых вычислений были и другие, которые были столь же влиятельными (или, возможно, даже более влиятельными), например Алан Тьюринг (который работал с фон Нейманом в Принстоне), Джон Маучли и Джон Эккерт (который возглавил строительство ENIAC, первого компьютера, в 1940-х годах). Но именно фон Нейман впервые понял, что инструкции программы должны храниться в компьютере так же, как и данные программы. По сей день такие машины называются «машинами фон Неймана».

После войны фон Нейман заинтересовался искусственным интеллектом. В 1953 году он пригласил итальянско-норвежского математика Нильса Барричелли (Nils Barricelli) в Принстон для изучения

искусственной жизни. Барричелли использовал новые цифровые компьютеры для симуляции эволюционных процессов. Он не интересовался биологической эволюцией и решением оптимизационных задач. Он хотел создать искусственную жизнь внутри компьютера, используя процессы, которые находятся в природе (например, размножение и мутации). В 1953 году он писал: «проводится ряд численных экспериментов с целью проверить возможность эволюции, подобной той, которая имеется у живых организмов, происходящей в искусственно созданной вселенной» [Dyson, 1998, стр. 111]. Барричелли стал первым человеком, который написал программу на основе генетического алгоритма. Его первая работа на эту тему была опубликована на итальянском языке в 1954 году под названием «Esempi numerici di processi di evoluzione» («Численные модели эволюционных процессов») [Barricelli, 1954].

Александр Фрейзер (Alexander Eraser), родившийся в Лондоне в 1923 году, последовал за Барричелли и использовал компьютерные программы для симулирования эволюции. Его образование и карьера привели его в Гонконг, Новую Зеландию, Шотландию и, наконец, в 1950-х годах в Государственное объединение научных и прикладных исследований (Commonwealth Scientific and Industrial Research Organisation, CSIRO) в Сиднее, Австралия. Фрейзер не был инженером; он был биологом и интересовался эволюцией. Он не мог наблюдать эволюцию, происходящую в окружающем мире, потому что она была слишком медленной, требуя периодов времени порядка миллионов лет. Фрейзер решил изучать эволюцию, создав собственную вселенную внутри цифрового компьютера. Таким образом, он смог ускорить процесс и наблюдать, как эволюция действительно работает. В 1957 году Фрейзер написал статью под названием «Симуляция генетических систем с помощью автоматических цифровых компьютеров» [Fraser, 1957], став первым, кто использовал компьютерные симуляции с целью изучения биологической эволюции. Он опубликовал целый ряд статей о своей работе, в основном в биологических журналах. В конце 1950–1960-х годах многие другие биологи последовали его примеру и начали использовать компьютеры для симулирования биологической эволюции.

Ханс-Йоахим Бремерман (Hans-Joachim Bremermann), математик и физик, также выполнял ранние компьютерные симуляции биологической эволюции. Его первая работа по этому вопросу была опубликована в качестве технического доклада в 1958 году, когда он был профессором Вашингтонского университета, и называлась «The evolution of intelligence» (Эволюция интеллекта) [Fogel и Anderson, 2000]. Бремерман

работал большую часть своей карьеры в Калифорнийском университете в Беркли, где в 1960-х годах использовал компьютерные симуляции для изучения работы сложных систем, в особенности эволюции. Но его компьютерные программы не просто моделировали эволюцию – они также симулировали взаимодействие паразита с хозяином, распознавание образов человеческим мозгом и реакцию иммунной системы.

Джордж Бокс (George Box), родившийся в 1919 году в Англии, также интересовался искусственной эволюцией, но, в отличие от своих предшественников, не интересовался искусственной жизнью или эволюцией ради нее самой. Он хотел решать реальные задачи. Бокс использовал статистику для анализа проектов и результатов экспериментов, затем он стал промышленным инженером и использовал статистику для оптимизации производственных процессов. Каков лучший способ размещения машин в заводских цехах, который максимизирует производство деталей? Каков лучший способ организации материалопотока на заводе? В течение 1950-х годов Бокс разработал метод, который он назвал «эволюционной операцией» как способ оптимизации производственного процесса во время его функционирования. Его работа не была генетическим алгоритмом в чистом виде, но она использовала идею эволюции посредством накопления многочисленных инкрементных изменений для оптимизации инженерного проектирования. Его первая работа на эту тему была опубликована в 1957 году под названием «Evolutionary operation: A method for increasing industrial productivity» («Эволюционная операция: метод повышения производительности труда в промышленности») [Box, 1957].

Джордж Фридман (George Friedman), как и Джордж Бокс, был практичным человеком. Для своей магистерской диссертации 1956 года в Калифорнийском университете в Лос-Анджелесе он разработал робота, который мог учиться строить электрические цепи для управления своим собственным поведением. Тема диссертации: «Selective Feedback Computers for Engineering Synthesis and Nervous System Analogy» («Вычислители селективной обратной связи для инженерного синтеза и аналогии нервной системы») [Friedman, 1998], [Fogel, 2006]. Его работа была похожа на сегодняшние генетические алгоритмы, несмотря на то что для описания своего подхода он использовал термин «компьютер с селективной обратной связью». В последнем абзаце его заключения говорится: «концепции и схематичные иллюстрации в этой статье хоть и не убедительно, но демонстрируют полезность [генетических алгоритмов]… по крайней мере, обозначив возможную область для дальнейше-

го расследования». И в самом деле! Сейчас, спустя более полувека после диссертации Фридмана, ежегодно публикуются тысячи технических статей на тему генетических алгоритмов.

Еще одним пионером в области генетических алгоритмов был Лоуренс Фогель (Lawrence Fogel), который начал работать над генетическими алгоритмами в 1962 году. В 1966 году вместе с Элвином Оуэнсом (Alvin Owens) и Майклом Уолшем (Michael Walsh) он написал первую книгу о генетических алгоритмах: «Artificial Intelligence through Simulated Evolution» («Искусственный интеллект через симулированную эволюцию») [Fogel и соавт., 1966]. Ранняя работа Фогеля в области генетических алгоритмов была мотивирована инженерными задачами, такими как предсказание сигналов, моделирование боевых действий и управление инженерными системами. Сын Лоуренса Фогеля, Дэвид Фогель, выполнил редакцию важного тома, который содержит 31 фундаментальную работу о генетических алгоритмах и смежных темах [Fogel, 1998].

После плодотворной работы Барричелли, Фрейзера, Бремермана, Бокса и Фридмана в 1950-х годах другие исследователи начали использовать генетические алгоритмы для изучения биологической эволюции и решения инженерных задач. Некоторые важные достижения в области генетических алгоритмов были сделаны в 1960-х годах Джоном Холландом (John Holland), профессором психологии, электротехники и компьютерных наук в Мичиганском университете. В 1960-х Холланд интересовался адаптивными системами. Его интересовала не эволюция или оптимизация, а то, как системы адаптируются к окружающей среде. Он начал преподавать и проводить исследования в этих областях, а в 1975 году написал свою знаменитую книгу «Adaptation in Natural and Artificial Systems» («Адаптация в естественных и искусственных системах») [Holland, 1975]. Книга стала классикой благодаря демонстрации математики эволюции. Также в 1975 году студент Холланда Кеннет Де Йонг (Kenneth De Jong) защитил докторскую диссертацию на тему «An analysis of the behavior of a class of genetic adaptive systems» («Анализ поведения класса генетических адаптивных систем»). Диссертация Де Йонга стала первым систематическим и тщательным исследованием использования генетических алгоритмов для оптимизации. Де Йонг использовал набор примеров задач для изучения влияния различных параметров генетического алгоритма на результативность оптимизации. Его труд был настолько тщательным, что в течение длительного времени любая работа по оптимизации, которая не включала эталонные задачи Де Йонга, считалась неадекватной.

Именно в 1970-х и 1980-х годах объем исследований генетических алгоритмов стал увеличиваться в геометрической прогрессии. Вероятно, это объясняется несколькими факторами. Одним из факторов явилось увеличение вычислительных мощностей, которые стали доступны с популяризацией и коммерциализацией транзистора в 1950-х годах, который привел к экспоненциальному росту вычислительных возможностей. Другим фактором был повышенный интерес к биологически обусловленным алгоритмам, поскольку исследователи видели ограничения обычных вычислений. В 1970-х и 1980-х годах экспоненциальный рост также претерпевали два других биологически обусловленных вычислительных алгоритма, нечеткая логика и нейронные сети, хотя эти парадигмы не требуют много вычислительной мощности.

3.4. Простой бинарный генетический алгоритм

Предположим, у вас есть задача, которую вы хотите решить. Если вы можете представить каждое возможное решение задачи в виде битовой строки, то генетический алгоритм мог бы решить эту задачу. Каждое потенциальное решение называется «кандидатным решением», или «особью». Группа особей называется генетико-алгоритмической «популяцией». Это означает, что нам нужно закодировать каждый параметр задачи в виде битовой строки. В данном разделе генетические алгоритмы представлены на нескольких простых примерах. Мы не демонстрируем эти примеры как реальные задачи; они представлены как простые задачи, которые хорошо иллюстрируют существенные функциональные особенности генетических алгоритмов.

3.4.1. *Генетический алгоритм для проектирования роботов*

Предположим, что наша задача предполагает конструирование мобильного робота малой массы, который обладает достаточной мощностью для навигации по пересеченной местности и достаточной дальностью, чтобы ему не приходилось возвращаться на свою базу слишком часто. Параметры, которые нам нужно определить в нашей конструкции робота, включают тип и размер двигателя, тип и размер источника питания. Тип и размер двигателя можно было бы закодировать следующим образом:

000 = 5-вольтовый шаговый двигатель
001 = 9-вольтовый шаговый двигатель
010 = 12-вольтовый шаговый двигатель
011 = 24-вольтовый шаговый двигатель
100 = 5-вольтовый серводвигатель
101 = 9-вольтовый серводвигатель
110 = 12-вольтовый серводвигатель
111 = 24-вольтовый серводвигатель (3.1)

Тип и размер источника питания можно было бы закодировать следующим образом:

000 = 12-вольтовая никель-кадмиевая батарея
001 = 24-вольтовая никель-кадмиевая батарея
010 = 12-вольтовая литий-ионная батарея
011 = 24-вольтовая литий-ионная батарея
100 = 12-вольтовая солнечная панель
101 = 24-вольтовая солнечная панель
110 = 12-вольтовый термоядерный реактор
111 = 24-вольтовый термоядерный реактор (3.2)

Кодирование параметров системы является важным аспектом генетического алгоритма и будет иметь значительное влияние на то, будет генетический алгоритм работать в действительности или нет.

После того как мы определились со схемой кодирования, нам нужно решить, как оценить «приспособленность» каждого потенциального решения задачи. В нашем примере с роботом, возможно, будет формула, которая связывает вес робота с типоразмером двигателя и типоразмером источника питания. У нас, возможно, будут другие формулы, которые связывают мощность робота с двигателем и источником питания и которые связывают дальность хода робота с двигателем и источником питания. Мы не сможем просимулировать эволюцию, если у нас нет хорошего определения функции приспособленности. С другой стороны, у нас, возможно, будет компьютерная симуляция, в которой мы могли бы вводить типоразмер двигателя и типоразмер источника питания и выводить меру того, насколько хорошо работает наша проектная разработка. Именно здесь решающее значение имеет понимание задачи проектировщиком генетического алгоритма. В задаче с генетическим алгоритмом нет никаких жестких правил для определения функции приспособленности. Прерогативой проектировщика генетического алгоритма является достаточно хорошее понимание задачи. Благодаря

этому он сможет определить логичную функцию приспособленности. В нашем примере у нас, возможно, будет функция приспособленности, которая выглядит следующим образом:

Приспособленность = Дальность (часы) + Мощность (Вт) – Вес (кг). (3.3)

Дальность, мощность и вес – все эти параметры могут быть сложными функциями с участием типа двигателя и типа источника питания, либо они могут быть определены на выходе из симуляционного программного или аппаратного обеспечения[5].

Мы начинаем генетический алгоритм, случайно генерируя множество особей. Рассмотрим две особи в популяции генетического алгоритма. Первая особь – это конструкция робота с 12-вольтовым шаговым двигателем и 24-вольтовой солнечной панелью, и вторая особь – конструкция робота с 9-вольтовым серводвигателем и 24-вольтовой никель-кадмиевой батарей. Эти две особи указаны следующим образом:

$$\begin{aligned} \text{Особь 1} &= \underbrace{\text{12-вольтовый шаговый, двигатель}}_{010}\ \underbrace{\text{24-вольтовая солнечная панель}}_{101} \\ \text{Особь 2} &= \underbrace{\text{9-вольтовый, серводвигатель}}_{101}\ \underbrace{\text{24-вольтовая никель-кадмиева батарея}}_{001} \end{aligned} \quad (3.4)$$

Особь 1 кодируется битовой строкой 010101, особь 2 – битовой строкой 101001. Каждый бит называется аллелем. Последовательность бит в особи, которая содержит информацию о каком-то признаке этой особи, называется геном. Конкретные гены называются генотипами, а специфичный для конкретной задачи параметр, который представлен генотипом, называется фенотипом. В нашем примере с роботом у каждой особи есть два гена: один для типоразмера двигателя и один для типоразмера батареи. Особь 1 имеет генотип 010 для двигателя, что соответствует фенотипу «12-вольтовый шаговый двигатель», и генотип 101 для батареи, что соответствует фенотипу «24-вольтовая солнечная батарея». Совокупность всех генов особи называется хромосомой. Особь 1 имеет хромосому 010101.

[5] Из-за разницы в единицах измерения мы не можем складывать часы, ватты и килограммы между собой, поэтому нам понадобится несколько параметров шкалирования, которые позволяют взвешивать относительную важность каждого члена и преобразовывать количества в единицы, которые могут быть сложены. Тем не менее это уравнение дает общее представление о том, как сформулировать уравнение приспособленности.

3.4.2. *Отбор и скрещивание*

Генетический алгоритм может иметь много особей. Типичные генетические алгоритмы имеют десятки или сотни особей. Приведенные выше две особи могут спариваться, как спариваются особи в биологических популяциях. В целях спаривания мы даем им скрещиваться, или «пересечься»[6], что означает, что каждая особь делится частью своей генетической информации со своими детьми. Для того чтобы найти точку скрещивания, мы выбираем случайное число от 1 до 5. Предположим, что мы выбираем случайное число 2. Из этого следует, что две особи обмениваются всеми своими аллелями после второй битовой позиции в каждой хромосоме, как показано на рис. 3.1.

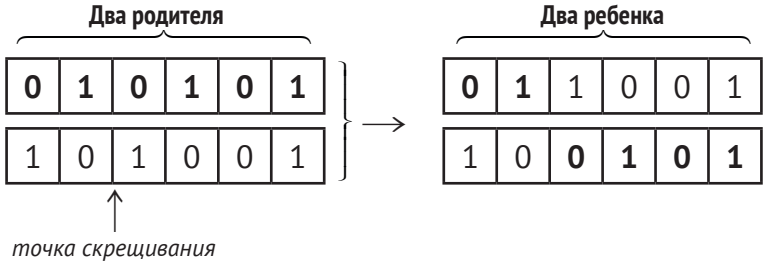


Рис. 3.1. Иллюстрация скрещивания в бинарном генетическом алгоритме. Точка скрещивания выбирается случайно. Два родителя рождают двух детей

Два родителя спарились (то есть пересеклись) и родили двух детей. Каждый ребенок получает некую генетическую информацию от одного родителя и другую – от другого. Родители умирают, а дети выживают и продолжают эволюционный процесс. Это событие называется одним генетико-алгоритмическим поколением.

Как и в биологии, часть детей будет иметь высокую приспособленность, а другие – низкую. Низко приспособленные особи имеют высокую вероятность умереть в своем поколении; то есть в генетическом алгоритме они из симуляции извлекаются. Высоко приспособленные особи выживают и скрещиваются с другими высоко приспособленными особями и тем самым производят новое поколение особей. Этот процесс продолжается до тех пор, пока генетический алгоритм не найдет приемлемое решение оптимизационной задачи.

В какой-то момент в нашей программе с генетическим алгоритмом нам придется решить, какие особи спариваются для рождения детей.

[6] Термины «скрещивание» и «пересечение» (от англ. *crossover*) часто используются как синонимы. – *Прим. перев.*

Это решение основано на приспособленности особей в популяции. Наиболее приспособленные особи, скорее всего, будут спариваться и произведут детей, в то время как наименее приспособленные особи вряд ли найдут себе пару и, следовательно, скорее всего, умрут, не произведя детей.

Одним из распространенных способов отбора родителей является отбор по принципу колеса рулетки, или рулеточный отбор, который еще также называется отбором пропорционально приспособленности. Предположим, в популяции есть четыре особи. (Реальный генетический алгоритм будет иметь гораздо больше четырех особей, но этот пример дан просто для иллюстрации.) Предположим, что индивидуальные приспособленности оцениваются следующим образом:

Особь 1: Приспособленность = 10
Особь 2: Приспособленность = 20
Особь 3: Приспособленность = 30
Особь 4: Приспособленность = 40 (3.5)

Особь 4 – наиболее приспособленная, особь 1 – наименее приспособленная. Мы создаем колесо рулетки, в котором каждый сектор соответствует приспособленности одной из особей. В нашем примере колесо рулетки показано на рис. 3.2.

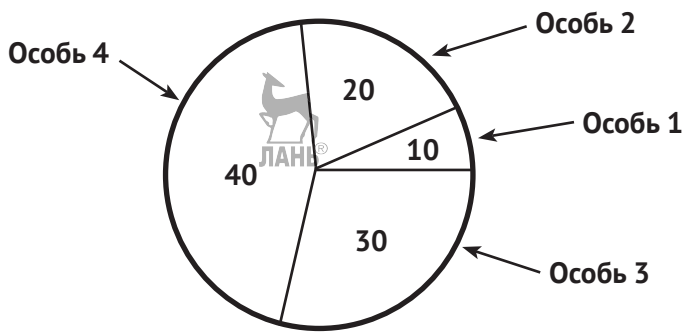


Рис. 3.2. Приведенная выше круговая диаграмма иллюстрирует рулеточный отбор в генетическом алгоритме для четырехчленной популяции. Каждой особи назначается доля, пропорциональная ее приспособленности. Отбор каждой особи в качестве родителя пропорционален его доле в колесе рулетки

Для того чтобы поддерживать размер популяции постоянным из поколения в поколение, мы отбираем две пары. В результате будет произведено четверо детей. Для того чтобы отобрать первую пару, мы вра-

щаем воображаемый (симулированный на компьютере) диск колеса, и где бы он ни оказался в колесе рулетки, он решает, кто является первым родителем. Из рулетки на рис. 3.2 мы видим, что особь 1 имеет шанс быть отобранной, равный 10%; особь 2 имеет шанс быть отобранной, равный 20%; особь 3 имеет шанс, равный 30%; и особь 4 имеет шанс, равный 40%. Другими словами, каждая особь имеет вероятность быть отобранной, пропорциональную ее приспособленности. Затем мы вращаем колесо рулетки во второй раз, для того чтобы отобрать второго родителя. Если колесо рулетки останавливается на той же самой особи, что и при первом вращении, то мы снова вращаем колесо – родитель не может спариваться сам с собой. После того как у нас будет два родителя, они скрещиваются и производят двух детей. Затем мы повторяем процесс и в результате получаем еще двух родителей, спариваем их и получаем еще двух детей. Этот процесс продолжается до тех пор, пока популяция детей не сравняется с размером популяции родителей. Эта идея проиллюстрирована на рис. 3.3.

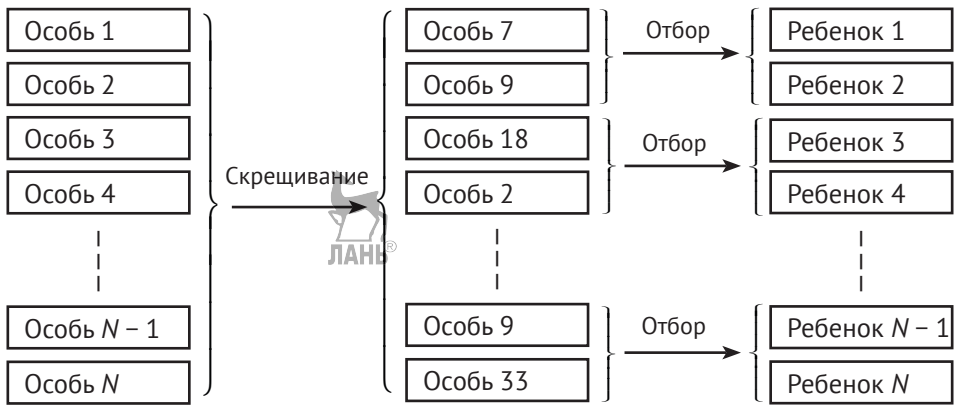


Рис. 3.3. Иллюстрация скрещивания популяции родителей для создания популяции детей. Исходная популяция из *N* особей слева проходит процесс отбора, возможно, рулеточный отбор, для того чтобы создать множество из *N* родителей. Некоторые особи, возможно, будут отобраны более одного раза, в то время как другие особи, возможно, не будут отобраны ни разу. Затем каждая пара родителей в середине скрещивается и создает пару детей. Взято из публикации [Whitley, 2001]

При наличии значений приспособленности, показанных на рис. 3.2, процесс вращения колеса рулетки для отбора родителя может быть выполнен, как показано на рис. 3.4. Мы повторяем процесс рис. 3.4 четыре раза, чтобы отобрать четырех родителей, которые создают четырех детей для следующего поколения.

В общем случае, при наличии популяции из N особей, рис. 3.5 демонстрирует псевдокод отбора родителя с помощью колеса рулетки. Мы повторяем процесс рис. 3.5 столько раз, сколько необходимо, для того чтобы отобрать родителей с целью создания детей следующего поколения.

```
Сгенерировать равномерно распределенное случайное число r ∈ [0, 1]
If r < 0.1 then
  Родитель = Особь 1
else if r < 0.3 then
    Родитель = Особь 2
else if r < 0.6 then
    Родитель = Особь 3
else
    Родитель = Особь 4
End if
```

Рис. 3.4. Приведенный выше псевдокод показывает, как выполняется отбор одного из родителей на основе колеса рулетки на рис. 3.2

```
x_i = i-я особь в популяции, i ∈ [1, N]
f_i ← приспособленность(x_i) для i ∈ [1, N]
f_sum = Σ_{i=1}^{N} f_i
Сгенерировать равномерно распределенное случайное число r ∈ [0, f_sum]
F ← f_1
k ← 1
While F < r
    k ← k + 1
    F ← F + f
End while
Родитель ← x_k
```

Рис. 3.5. Приведенный выше псевдокод показывает, как выполняется отбор одного родителя из N особей на основе рулеточного отбора. Данный псевдокод исходит из того, что значение приспособленности $f_i > 0$ для всех $i \in [1, N]$

3.4.3. *Мутации*

Последний шаг в генетическом алгоритме называется мутацией. Мутация в биологии встречается относительно редко, по крайней мере в той мере, насколько она заметно влияет на детей. В большинстве реализаций генетического алгоритма мутации также редки (порядка 2 %). Но мы не можем сказать в общем случае, какой должна быть правиль-

ная настройка скорости мутации[7] в генетическом алгоритме. Лучшая скорость мутации зависит от задачи, размера популяции, кодирования и других факторов. Вне зависимости от ее частоты мутация имеет важное значение, потому что позволяет эволюционному процессу разведывать новые потенциальные решения задачи. Если какая-то генетическая информация в популяции отсутствует, мутация обеспечивает возможность введения данной информации в популяцию. Это важно в биологической эволюции, но еще важнее в генетических алгоритмах. Это связано с тем, что генетические алгоритмы, как правило, имеют такие небольшие размеры популяции, что близкородственное скрещивание (инбридинг) может легко стать проблемой, а эволюционные тупики чаще встречаются в генетических алгоритмах, чем в биологической эволюции. В биологической эволюции мы обычно говорим о популяциях, состоящих из миллионов особей, в то время как в генетических алгоритмах мы говорим о популяциях всего лишь из десятков или сотен особей.

Для того чтобы реализовать мутацию, мы выбираем мутационную вероятность, скажем, 1%. Это означает, что, после того как процесс скрещивания производит детей, каждый бит в каждом ребенке имеет 1%-ную вероятность переключиться на противоположное значение (1 меняется на 0 или 0 меняется на 1). Мутация проста, но важно выбрать разумную мутационную вероятность. Слишком высокая мутационная вероятность приводит к тому, что генетический алгоритм ведет себя как случайный поиск, что обычно не является отличным способом решения задачи. Слишком низкая мутационная вероятность приводит к проблемам, связанным с близкородственным скрещиванием и эволюционными тупиками, что также не позволяет генетическому алгоритму найти хорошее решение.

Если у нас популяция из N особей x_j, где каждая особь имеет n бит, а скорость мутации равна p, то в конце каждого поколения мы переворачиваем каждый бит в каждой особи с вероятностью p:

$$\begin{aligned} r &\leftarrow U[0,1] \\ x_i(k) &\leftarrow \begin{cases} x_i(k) & \text{если } r \geq \rho \\ 0 & \text{если } r < \rho \text{ и } x_i(k) = 1 \\ 1 & \text{если } r < \rho \text{ и } x_i(k) = 0 \end{cases} \end{aligned} \tag{3.6}$$

[7] В данном переводе термин mutation rate переведен как «скорость мутации», то есть число мутаций в пересчете на ген за поколение. В генетике этот термин также переводится как частота мутаций и имеет более широкое значение, как число изменений в генетической информации с течением времени. – *Прим. перев.*

для $i \in [1, N]$ и $k \in [1, n]$, где $U[0, 1]$ – это случайное число, равномерно распределенное на [0, 1].

3.4.4. *Краткая формулировка генетического алгоритма*

В этом разделе был дан простой пример генетического алгоритма для проекта робота и рассмотрены операции отбора, скрещивания и мутации в генетических алгоритмах. Теперь мы объединим весь этот материал, для того чтобы на рис. 3.6 схематично сформулировать генетический алгоритм[8].

```
Родители ← {случайно сгенерированная популяция}
While not (критерий останова)
    Рассчитать приспособленность каждого родителя в популяции
    Дети ← ∅
    While | Дети | < | Родители |
        Применить приспособленности для вероятностного отбора пары
        родителей с целью их спаривания
        Спарить родителей для создания детей c1 и c2
        Дети ← Дети ∪ {c1, c2}
    Loop
    Случайно мутировать нескольких детей
    Родители ← Дети
Next поколение
```

Рис. 3.6. Приведенный выше псевдокод иллюстрирует простой генетический алгоритм

3.4.5. *Регулировочные параметры и примеры генетического алгоритма*

На рис. 3.6 схематично представлен простой генетический алгоритм, причем мы видим большую гибкость в реализации алгоритма рис. 3.6. Например, критерий останова генетического алгоритма может включать несколько разных параметров – те же самые параметры, что и в любом другом итеративном оптимизационном алгоритме. Одна из возможностей заключается в том, что генетический алгоритм может работать в течение заданного числа поколений. Другая возможность за-

[8] В данном примере и в остальных примерах псевдокода инструкции языка программирования намеренно оставлены на английском, для того чтобы подчеркнуть логику алгоритмов. – *Прим. перев.*

ключается в том, что он будет работать до тех пор, пока приспособленность лучшей особи не станет лучше, чем определяемый пользователем порог. Если наша задача заключается в том, чтобы найти решение, которое является «достаточно хорошим», то это может быть разумным критерием останова. Другая возможность генетического алгоритма – выполняться до тех пор, пока приспособленность лучшей особи перестанет улучшаться из поколения в поколение. Это будет означать, что эволюционный процесс достиг плато и не может улучшаться дальше.

Существует ряд параметров, которые проектировщик генетического алгоритма должен указать, для того чтобы получить хорошие результаты. Выбор этих параметров часто может означать разницу между успехом и неудачей. Некоторые из этих параметров включают следующие:

1) схема кодирования, которая увязывает решения задачи с битовыми строками. Приведенные ниже несколько примеров иллюстрируют двоичное кодирование вещественных чисел, в разделе 3.5 рассматриваются генетические алгоритмы с вещественными параметрами, а в разделе 8.3 затрагивается тема кодирования с помощью кодов Грея в бинарных генетических алгоритмах;
2) функция приспособленности, которая увязывает решения задачи со значениями приспособленности;
3) размер популяции;
4) метод отбора. Выше мы говорили о рулеточном отборе, но возможны и другие типы отбора, включая турнирный отбор, ранговый отбор и многие другие вариации. В разделе 8.7 обсуждаются некоторые из этих вариаций;
5) скорость мутации. Генетический алгоритм, в котором используется слишком высокая скорость мутации, выродится в случайный поиск. Вместе с тем генетический алгоритм, в котором используется слишком низкая скорость мутации, не сможет в достаточной мере разведать поисковое пространство;
6) шкалирование приспособленности. Оно определяет, как реализуется функция приспособленности. Иногда функция приспособленности слабо определена, и в этом случае все особи имеют значения приспособленности, которые очень близки друг к другу. Если значения приспособленности образовывают скопления, то процесс отбора не сможет хорошо отличать высоко приспособленные особи от низко приспособленных. А это помешает более приспособленным особям размножиться в следующем поколе-

нии. Иногда возникает противоположная проблема; значения приспособленности разбросаны слишком широко, в результате чего низко приспособленные особи не будут иметь никакого шанса быть отобранным для размножения. В разделе 8.7 обсуждается вопрос шкалирования приспособленности;

7) тип скрещивания. Выше мы говорили о скрещивании в одной точке в каждой паре хромосом, но мы также можем выполнять скрещивание в нескольких точках. В разделе 8.8 рассматриваются разные типы скрещивания;
8) видообразование/инцест. Некоторые исследователи в области генетических алгоритмов позволяют особям спариваться только в том случае, если они достаточно похожи друг на друга; то есть только если они принадлежат к одному и тому же «виду». Другие исследователи позволяют особям спариваться только в том случае, если они достаточно отличаются друг от друга; то есть только если они принадлежат к разным «семьям». В разделе 8.6 обсуждаются некоторые из этих идей.

Данные вопросы также относятся к другим эволюционным алгоритмам, отличным от генетических алгоритмов, и поэтому в главе 8 обсуждаются эти и некоторые другие вопросы.

■ Пример 3.2

Рассмотрим задачу минимизации примера 2.2:

$$\min_x f(x), \quad \text{где } f(x) = x^4 + 5x^3 + 4x^2 - 4x + 1. \tag{3.7}$$

Предположим, мы каким-то образом знаем заранее, что минимум функции $f(x)$ происходит в области $x \in [-4, -1]$. Мы решили кодировать x четырьмя битами:

$$\begin{array}{ll} 0000 = -4.0, & 0001 = -3.8, \\ 0010 = -3.6, & 0011 = -3.4, \\ 0100 = -3.2, & 0101 = -3.0, \\ 0110 = -2.8, & 0111 = -2.6, \\ 1000 = -2.4, & 1001 = -2.2, \\ 1010 = -2.0, & 1011 = -1.8, \\ 1100 = -1.6, & 1101 = -1.4, \\ 1110 = -1.2, & 1111 = -1.0. \end{array} \tag{3.8}$$

Схема кодирования представляет собой компромисс между точностью и сложностью. Больше бит дадут нам большую разрешающую спо-

собность, но также усложнят генетический алгоритм. Рассмотрим исходную популяцию из четырех особей, сгенерированную случайно:

$$\begin{aligned} x_1 &= 1100, \\ x_2 &= 1011, \\ x_3 &= 0010, \\ x_4 &= 1001. \end{aligned} \tag{3.9}$$

Мы хотим минимизировать $f(x)$, но генетические алгоритмы предназначены для максимизации приспособленности[9]. Поэтому, для того чтобы задача соответствовала структуре генетического алгоритма, нам нужно конвертировать задачу минимизации в задачу максимизации. Мы можем сделать это за счет максимизации величины, противоположной $f(x)$. Значения приспособленности, следовательно, будут получены путем декодирования особей с использованием комбинаций генотипа/фенотипа, показанных в уравнении (3.8), а затем оценивания $f(x)$:

$$\begin{aligned} \text{fitness}(x_1) &= -f(-1.6) = -3.71 \\ \text{fitness}(x_2) &= -f(-1.8) = -2.50 \\ \text{fitness}(x_3) &= -f(-3.6) = -1.92 \\ \text{fitness}(x_4) &= -f(-2.2) = +0.65 \end{aligned} \tag{3.10}$$

Теперь в каждое значение приспособленности мы произвольно добавим некоторый сдвиг, с тем чтобы все они были больше 0. Это необходимо для того, чтобы позже мы могли назначить каждой особи процентные значения приспособленности:

$$\begin{aligned} f_1 &= -3.71 + 10 = 6.29 \\ f_2 &= -2.50 + 10 = 7.50 \\ f_3 &= -1.92 + 10 = 8.08 \\ f_4 &= +0.65 + 10 = 10.65 \end{aligned} \tag{3.11}$$

Теперь мы вычисляем относительные значения приспособленности каждой особи. Относительной приспособленностью каждой особи является ее вероятность отбора при вращении колеса рулетки:

$$\begin{aligned} p_1 &= f_1/(f_1 + f_2 + f_3 + f_4) = 0.19 \\ p_2 &= f_2/(f_1 + f_2 + f_3 + f_4) = 0.23 \\ p_3 &= f_3/(f_1 + f_2 + f_3 + f_4) = 0.25 \\ p_4 &= f_4/(f_1 + f_2 + f_3 + f_4) = 0.33 \end{aligned} \tag{3.12}$$

[9] Это утверждение исходит из того, что мы используем рулеточный отбор. Некоторые методы отбора, которые мы будем обсуждать в разделе 8.7, исходят из того, что лежащая в основе задача не является задачей максимизации.

Исходная популяция кратко сформулирована в табл. 3.1.

Таблица 3.1. Пример 3.2: исходная популяция простого генетического алгоритма

Номер особи	Генотип	Фенотип	Приспособленность	Вероятность отбора
x_1	1100	−1.4	−4.56	0.19
x_2	1011	−1.8	−2.50	0.23
x_3	0010	−3.6	−1.92	0.25
x_4	1001	−2.2	+0.65	0.33

Таблица 3.1 показывает нам, что x_2 и x_3 при каждом вращении колеса рулетки имеют шанс отбора порядка 25 %, в то время как x_4 имеет вероятность отбора почти в два раза выше, чем у x_1. Для того чтобы начать первое поколение генетического алгоритма, мы генерируем четыре равномерно распределенных случайных числа в области [0, 1] и используем их для отбора четырех родителей. Предположим, что этот процесс приводит к отбору x_3, x_4, x_4 и x_1. Из этого следует, что мы хотим скрестить x_3 и x_4, для того чтобы получить двух детей, и x_4 и x_1, чтобы получить еще двух детей. Помните, что точка скрещивания для каждой пары родителей отбирается случайно. Это показано в табл. 3.2.

Таблица 3.2. Пример 3.2: скрещивание в простом генетическом алгоритме. Случайно отобранные точки скрещивания выделены **жирным** шрифтом. Точка скрещивания находится между первыми двумя битами для первых родителей и в середине хромосомы для второго множества родителей

Родители		Дети	
Особь	**Генотип**	**Генотип**	**Приспособленность**
x_3	0010	0001	−8.11
x_4	1**001**	1010	−1.00
x_4	**10**01	1000	+2.30
x_1	11**00**	1101	−4.56

Из табл. 3.2 видно, что лучший ребенок имеет приспособленность 2.30, то есть лучше, чем лучший ребенок исходого поколения (0.65). Генетический алгоритм сделал значительный шаг к оптимизации $f(x)$. Нет никаких гарантий, что дети будут лучше, чем родители, но этот простой пример иллюстрирует, как генетический алгоритм может нацеливаться на решение оптимизационной задачи.

■ Пример 3.3

Рассмотрим задачу минимизации примера 2.5:

$$\min_{x,y} f(x, y), \quad \text{где}$$

$$f(x,y) = e - 20\exp\left(-0.2\sqrt{\frac{x^2+y^2}{2}}\right) - \exp\left(\frac{\cos(2\pi x)+\cos(2\pi y)}{2}\right). \tag{3.13}$$

Предположим, как в примере 2.5, что x и y могут находиться в диапазоне от –5 до +5. Нам нужно принять решение о разрешающей способности, которую мы хотим использовать для x и y в нашем генетическом алгоритме. Если для каждой независимой переменной x и y мы хотим получить разрешающую способность 0.25 или лучше, то нам нужно использовать шесть бит как для x, так и для y.

$$\begin{aligned} x \text{ генотип} &= x_g \in [0, 63] \\ x \text{ фенотип} &= -5 + \frac{10x_g}{63} \in [-5, 5] \\ y \text{ генотип} &= y_g \in [0, 63] \\ y \text{ фенотип} &= -5 + \frac{10y_g}{63} \in [-5, 5] \end{aligned} \tag{3.14}$$

В результате получаем разрешающую способность 10/63 = 0.159 для каждого бита x_g и y_g. Давайте выполним генетический алгоритм 10 поколений, для того чтобы попытаться минимизировать $f(x, y)$. Нам нужно определиться с размером популяции и скоростью мутации. Давайте применим размер популяции 20 и скорость мутации 2 % на бит. Типичный прогон генетического алгоритма дает результаты, показанные на рис. 3.7. Каждый прогон будет отличаться из-за используемых в генетическом алгоритме случайных чисел, но рис. 3.7 показывает типичные результаты. По мере увеличения числа поколений мы видим снижение как минимальной стоимости (то есть стоимости лучшей особи), так и средней стоимости популяции.

На рис. 3.8 показан контурный график $f(x, y)$, а также расположение на графике особей генетического алгоритма в первом, четвертом, седьмом и десятом поколениях. По рис. 3.8 видно, что популяция исходно разабросана по всей области вследствие нашей случайной инициализации. По мере поступательного развития генетического алгоритма популяция начинает кластеризоваться, и особи устремляются к минимуму, который находится в центре графика.

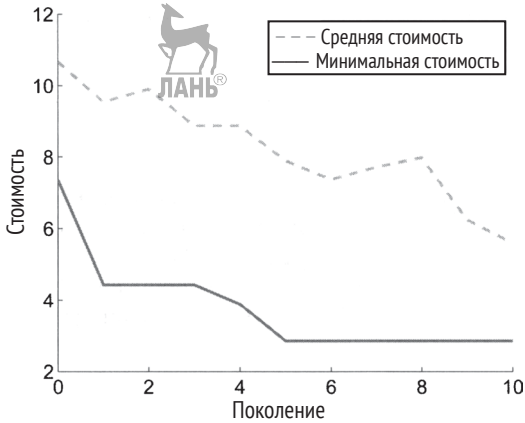


Рис. 3.7. Пример 3.3: типичные результаты симуляции генетическим алгоритмом с целью минимизации двумерной функции Экли

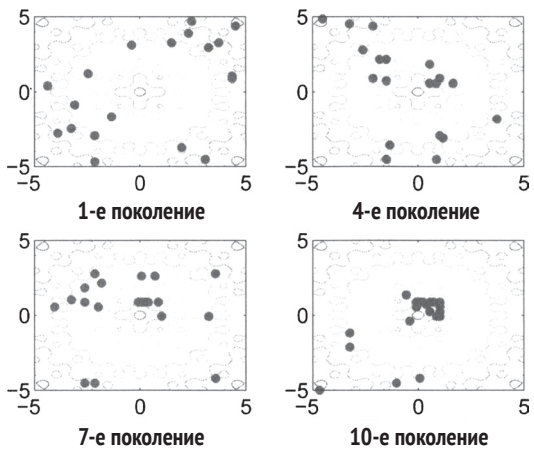


Рис. 3.8. Пример 3.3: типичные результаты симуляции генетическим алгоритмом с целью минимизации двумерной функции Экли. По мере поступательного развития генетического алгоритма особи в популяции постепенно начинают кластеризоваться в группы и двигаться к минимуму, который находится в центре контурного графика

Контурный график на рис. 3.8 показывает, насколько сложно минимизировать мультимодальную многомерную функцию. Вы можете вообразить, что стоите среди ландшафта с холмами и долинами рис. 3.8 и хотите найти самую низкую точку ландшафта. Это будет трудно сделать, потому что существует очень много пиков и долин. Вместе с тем больше шансов найти самую низкую точку будет у группы людей, раз-

бросанных по всему ландшафту. Люди могут учиться друг у друга: «Эта долина выглядит довольно низкой; давайте ее разведуем», – говорит один. «Нет, вот эта выглядит ниже; идем сюда!» Люди сотрудничают друг с другом и вместе находят самую низкую точку в долине. Это похоже на то, как работают генетические и другие эволюционные алгоритмы. Особи в популяции кооперируются, для того чтобы найти хорошее решение поставленной задачи.

3.5. Простой непрерывный генетический алгоритм

На рис. 3.6 схематично представлен простой бинарный генетический алгоритм, и это именно тот алгоритм, который мы использовали в примерах 3.2 и 3.3. Однако задачи в этих примерах определены на непрерывной области, и поэтому для применения бинарного генетического алгоритма нам пришлось дискретизировать область. Было бы проще и естественнее, если бы мы могли применять генетический алгоритм непосредственно к непрерывной области задачи. Мы используем термин *непрерывные генетические алгоритмы* или вещественно кодированные генетические алгоритмы для обозначения генетических алгоритмов, которые работают непосредственно с непрерывными величинами.

Расширить генетические алгоритмы с бинарных областей на непрерывные области довольно просто. Фактически мы по-прежнему можем использовать алгоритм, показанный на рис. 3.6, – нам просто нужно изменить несколько шагов этого алгоритма. Взглянем на операции на рис. 3.6 и рассмотрим, как они могли бы работать с оптимизационной задачей с непрерывной областью.

1. На рис. 3.6 сначала генерируется случайная исходная популяция. Это легко делается в непрерывной области. Предположим, что в нашем генетическом алгоритме мы хотим сгенерировать N особей. Тогда i-я особь будет обозначена как x_i для $i \in [1, N]$. Также предположим, что мы хотим минимизировать n-мерную функцию в непрерывной области. Тогда мы применим $i \in [1, N]$ и $k \in [1, n]$ для обозначения k-го элемента x_i:

$$x_i = [\, x_i(1) \quad x_i(2) \quad \cdots \quad x_i(n) \,]. \tag{3.15}$$

Предположим, что поисковая область k-й размерности равняется $[x_{\min}(k), x_{\max}(k)]$:

$$x_i(k) \in [x_{\min}(k),\ x_{\max}(k)] \tag{3.16}$$

для $i \in [1, N]$ и $k \in [1, n]$. Мы можем сгенерировать случайную исходную популяцию, как в первой строке рис. 3.6, следующим образом:

```
For i = 1 to N
    For k = 1 to n
        x_i(k) ← U [x_min(k), x_max(k)]
    Next k
Next i
```

То есть мы просто принимаем каждый $x_i(k)$ равным реализации случайной величины, равномерно распределенной между $x_{\min}(k)$ и $x_{\max}(k)$.

2. Далее мы начинаем цикл «while not (критерий останова)» на рис. 3.6. Первым шагом в этом цикле является расчет приспособленности каждой особи. Если мы пытаемся максимизировать $f(x)$, то вычисляем приспособленность каждой x_1, вычисляя $f(x_i)$. Если мы пытаемся минимизировать $f(x)$, то вычисляем приспособленность каждой x_i, вычисляя значение, противоположное результату функции $f(x_i)$.
3. Далее мы начинаем цикл «while |Дети| < |Родители|» на рис. 3.6. Первым шагом в этом цикле является «применение приспособленности для вероятностного отбора пары родителей с целью спаривания». Мы выполняем этот шаг, используя рулеточный отбор, как обсуждали в разделе 3.4.2. Мы рассмотрим другие варианты этого шага в разделе 8.7, но пока просто применяем рулеточный отбор.
4. Далее мы выполняем шаг «спаривания родителей» на рис. 3.6 для создания двух детей. Мы выполняем этот шаг, используя одноточечное скрещивание, как показано на рис. 3.1. Единственное различие состоит в том, что мы объединяем особей не с двоичной, а с непрерывной областью. Мы иллюстрируем одноточечное скрещивание для особей с непрерывной областью на рис. 3.9. В разделе 8.8 мы рассмотрим другие типы скрещивания для непрерывных генетических алгоритмов.
5. Далее мы выполняем шаг «случайной мутации» на рис. 3.6. В бинарных эволюционных алгоритмах мутация является прямолинейной операцией, как показано в уравнении (3.6). В непрерыв-

ном генетическом алгоритме мы мутируем особь $x_i(k)$, присваивая ей случайное число, которое генерируется из равномерного распределения в области поиска решений:

$$r \leftarrow U[0,1]$$
$$x_i(k) \leftarrow \begin{cases} x_i(k) & \text{если } r \geq \rho \\ U[x_{\min}(k),\ x_{\max}(k)] & \text{если } r < \rho \end{cases} \quad (3.17)$$

для $i \in [1, N]$ и $k \in [1, n]$, где ρ – это скорость мутации. Другие возможности мутации в генетических алгоритмах с непрерывной областью мы рассмотрим в разделе 8.9.

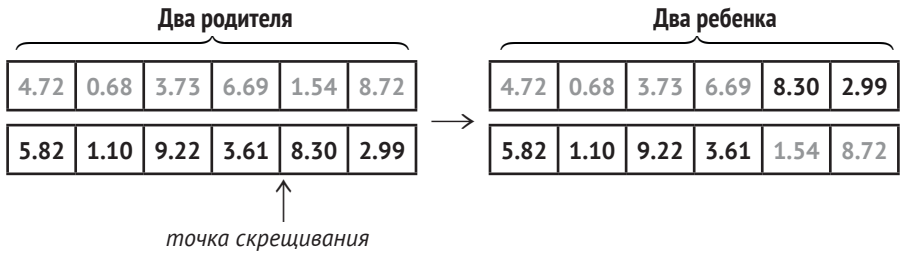


Рис. 3.9. Иллюстрация скрещивания в непрерывной области генетического алгоритма. Точка скрещивания отбирается случайно. Два родителя производят двух детей

Мутация в непрерывных генетических алгоритмах

Обратите внимание, что заданная скорость мутации имеет другой эффект в бинарном генетическом алгоритме, чем в непрерывном генетическом алгоритме. Если у нас задача с непрерывной областью с n размерностями и скоростью мутации ρ_c, то каждый признак решения у каждого ребенка имеет мутационную вероятность ρ_c. Например, на рис. 3.9 каждый из шести компонентов обоих детей имеет мутационную вероятность рс. Кроме того, мутация в непрерывном генетическом алгоритме приводит к тому, что признак решения берется из равномерного распределения между его минимальным и максимальным возможными значениями, как показано в уравнении (3.17)[10].

Однако в бинарном генетическом алгоритме каждая размерность каждой особи дискретизируется. Если мы дискретизируем непрерывную

[10] Равномерная мутация, наверное, является самым классическим типом мутации в непрерывных генетических алгоритмах. Однако, как описано в разделе 8.9, мы также можем выбрать из многих других типов мутации.

размерность в m бит и используем скорость мутации pb, то каждый бит имеет вероятность того, что ρ_b мутирует. Из этого следует, что каждый бит имеет вероятность не быть мутированным, равную $1 - \rho_b$. Следовательно, вероятность каждой размерности не быть мутированной равна вероятности того, что все m ее бит не мутированы, которая равна $(1 - \rho_b)^m$. Следовательно, вероятность мутации m-й размерности составляет $1 - (1 - \rho_b)^m$. Более того, если мутация все же произойдет, то мутированная размерность *не* является равномерно распределенной между ее минимальным и максимальным значениями; вместо этого ее распределение зависит от того, какой бит мутировал.

Мы можем получить скорость мутации pc для задачи с непрерывной областью, имеющую эффект, приблизительно равный скорости мутации ρ_b для дискретной задачи. Как мы обсуждали выше, если бинарный генетический алгоритм для дискретной задачи с m битами в расчете на размерность имеет скорость мутации ρ_b, то вероятность того, что любая заданная размерность не мутирует, равна $(1 - \rho_b)^m$. Это можно аппроксимировать путем разложения в ряд Тейлора первого порядка:

$$\begin{aligned} \Pr(\text{без мутации в бинарном ГА}) &= (1 - \rho_b)^m \\ &\approx 1 - m\rho_b \end{aligned}, \tag{3.18}$$

где аппроксимация справедлива для малых ρ_b. Если генетический алгоритм для непрерывной задачи имеет скорость мутации pc, то вероятность того, что любая заданная размерность не мутирует, равна $1 - \rho_c$. Приравнивание этой вероятности к уравнению (3.18) дает:

$$\begin{aligned} 1 - \rho_c &= 1 - m\rho_b \\ \rho_c &= m\rho_b \end{aligned} \tag{3.19}$$

Поэтому мутационный процесс в бинарном генетическом алгоритме с m бит на размерность и скоростью мутации ρ_b *приближенно* эквивалентен мутационному процессу в непрерывном генетическом алгоритме со скоростью мутации $m\rho_b$. В предыдущем предложении мы подчеркиваем слово *приближенно*, потому что пока не ясно, что эквивалентные скорости мутации в бинарном и непрерывном генетических алгоритмах дают эквивалентные результаты. Это вызвано тем, что распределение величины мутации бинарного генетического алгоритма отличается от величины мутации непрерывного генетического алгоритма. Интересной темой для дальнейшей исследовательской работы было бы тщательное изучение эквивалентности мутаций в бинарном и непрерывном генетических алгоритмах.

■ Пример 3.4

Рассмотрим задачу минимизации примера 3.3:

$$\min_{x,y} f(x, y), \quad \text{где}$$

$$f(x, y) = e - 20\exp\left(-0.2\sqrt{\frac{x^2 + y^2}{2}}\right) - \exp\left(\frac{\cos(2\pi x) + \cos(2\pi y)}{2}\right). \tag{3.20}$$

Предположим, что x и y могут находиться в диапазоне от –1 до +1. В примере 3.3 мы дискретизировали поисковые области с целью применения бинарного генетического алгоритма. Однако поскольку задача определена в непрерывной области, более естественным будет использовать непрерывный генетический алгоритм. В данном примере мы выполняем как бинарный, так и непрерывный генетические алгоритмы в течение 20 поколений с популяцией размером 10. В случае бинарного генетического алгоритма мы используем четыре бита на размерность и скорость мутации 2 % на бит, как в примере 3.3. Для того чтобы обеспечить непрерывному генетическому алгоритму примерно такой же эффект мутации, как и у бинарного, в непрерывном генетическом алгоритме мы используем скорость мутации 8 %. Мы также используем фактор элитарности 1, то есть оставляем лучшую особь в популяции из поколения в поколение (см. раздел 8.4).

На рис. 3.10 показаны лучшие особи, найденные в каждом поколении, усредненно по 50 симуляциям. Мы видим, что непрерывный генетический алгоритм значительно лучше бинарного генетического алгоритма. Для задач с непрерывной областью мы обычно (но не всегда) получаем более высокую результативность с бинарным генетическим алгоритмом, когда используем больше бит, и мы получаем лучшую результативность, если используем непрерывный генетический алгоритм.

■

Интересно отметить, что непрерывный генетический алгоритм имеет несколько противоречивую историю. Поскольку генетические алгоритмы изначально разрабатывались для бинарных представлений и вся ранняя теория генетических алгоритмов была ориентирована на бинарные генетические алгоритмы, в 1980-х годах исследователи скептически относились к распространению непрерывных генетических

алгоритмов [Goldberg, 1991]. Однако трудно поспорить с успехом непрерывных генетических алгоритмов, их простотой в использовании и их относительно недавней теоретической поддержкой.

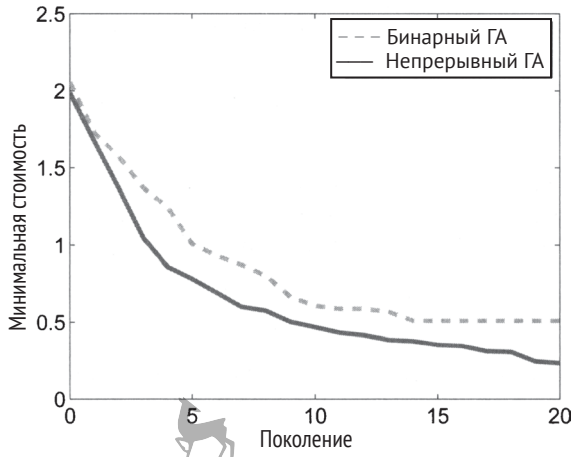


Рис. 3.10. Пример 3.4: сопоставление результативности бинарного и непрерывного генетических алгоритмов для двумерной функции Экли. На графике показана стоимость лучшей особи в каждом поколении, усредненно по 50 симуляциям

3.6. Заключение

Генетический алгоритм был одним из первых эволюционных алгоритмов, и сегодня он, пожалуй, является самым популярным. В последние годы появился ряд конкурирующих эволюционных алгоритмов, но генетические алгоритмы остаются популярными из-за хорошей о них осведомленности, простоты реализации, интуитивной привлекательности и хорошей результативности на целом ряде задач.

За прошедшие годы генетическим алгоритмам был посвящен целый ряд книг и обзорных работ. Книга Дэвида Голдберга [Goldberg, 1989a] была одной из первых книг о генетических алгоритмах, но, как и ранние книги по многим другим темам, она хоть и довольно-таки устарела, по-прежнему популярна из-за своего ясного изложения материала. Генетическим алгоритмам посвящен ряд других хороших книг, в том числе книги [Митчелл, 1998], [Michalewicz, 1996], [Haupt и Haupt, 2004] и [Reeves и Rowe, 2003], что примечательно из-за сильного акцента на последней теории. Некоторые популярные учебные статьи включают публикации [Bäck и Schwefel, 1993], [Whitley, 1994] и [Whitley, 2001].

Ввиду огромного числа книг и статей о генетических алгоритмах эта глава является обязательным кратким введением в данную тему. В этой главе мы пренебрегли многими вопросами, связанными с генетическими алгоритмами, не потому, что считаем, что они неважны, а просто потому, что наша точка зрения ограничена. Некоторые из этих вопросов включают запутанные генетические алгоритмы, которые имеют хромосомы переменной длины [Goldberg, 1989b], [Mitchell, 1998]; гендерные генетические алгоритмы, которые симулируют несколько полов в популяции генетического алгоритма и часто используются для многокритериальной оптимизации (глава 20) [Lis и Eiben, 1997]; островные генетические алгоритмы, включающие субпопуляции [Whitley и соавт., 1998]; сотовые генетические алгоритмы, которые накладывают определенные пространственные связи между особями в популяции [Whitley, 1994]; и ковариационно-матричную адаптацию, являющуюся локальной поисковой стратегией, которая может дополнить любой эволюционный алгоритм [Hansen и соавт., 2003].

Существует также ряд вариаций на основе базового генетического алгоритма, который мы представили в этой главе. Некоторые из этих вариаций чрезвычайно важны и могут стать решающим фактором между успехом и неудачей в приложении с участием генетического алгоритма. В главе 8 рассматриваются различные вариации, применимые к генетическим и другим эволюционным алгоритмам.

Задачи

Письменные упражнения

3.1. В разделе 3.4.1 приведен простой пример того, как мы можем представить параметры робототехнического проекта в генетическом алгоритме. Предположим, что у нас есть генетико-алгоритмическая особь, задаваемая битовой строкой 110010.

a) Какова хромосома для этой генетико-алгоритмической особи?

b) Каковы генотипы и фенотипы у этой особи?

3.2. Мы хотим применить бинарный генетический алгоритм для нахождения x с разрешающейспособностью 0.1, для того чтобы минимизировать двумерную функцию Растригина (см. раздел C.1.11) на области $[-5, 5]$.

a) Сколько генов нам нужно для каждой хромосомы?

b) Сколько бит нам нужно в каждом гене?

c) С учетом вашего ответа на вопрос в части (b) какова разрешающая способность каждого элемента x?

3.3. Дан генетический алгоритм с 10 особями $\{x_i\}$, и приспособленность x_i равняется $f(x_i) = i$ для $i \in [1,10]$. Для отбора 10 родителей с целью скрещивания мы применяем рулеточный отбор. Первые два родителя спариваются и создают двух детей, следующие два спариваются и создают еще двух детей и так далее.

a) Какова вероятность того, что наиболее приспособленная особь будет спариваться сама с собой хотя бы один раз и создаст двух клонированных детей?

b) Повторите часть (a) для наименее приспособленной особи.

3.4. Дан генетический алгоритм с 10 особями $\{x_i\}$, и приспособленность x_i равняется $f(x_i) = i$ для $i \in [1,10]$. Для отбора 10 родителей с целью скрещивания мы применяем рулеточный отбор.

а) Какова вероятность того, что особь x_{10} вообще не будет отобрана после 10 вращений колеса рулетки?

б) Какова вероятность того, что особь x_{10} будет отобрана ровно один раз после 10 вращений колеса рулетки?

в) Какова вероятность того, что особь x_{10} будет отобрана более одного раза после 10 вращений колеса рулетки?

3.5. Рулеточный отбор исходит из того, что значения приспособленности популяции удовлетворяют $f(x_i) \geq 0$ для $i \in [1,N]$. Предположим, у вас есть популяция со значениями приспособленности $\{-10, -5, 0, 2, 3\}$. Как бы вы предложили модифицировать эти значения приспособленности, для того чтобы вы могли применить рулеточный отбор?

3.6. Рулеточный отбор исходит из того, что популяция характеризуется значениями приспособленности $\{f(x_i)\}$, где более высокие значения приспособленности лучше, чем более низкие значения. Предположим, что у нас есть задача, популяция которой характеризуется значениями стоимости $\{c(x_i)\}$, где меньшие значения стоимости лучше, чем более высокие значения стоимости, и $c(x_i) > 0$ для всех i. Как бы вы модифицировали значения стоимости для применения рулеточного отбора?

3.7. У нас есть два родителя в бинарном генетическом алгоритме с n битами каждый. i-й бит в родителе 1 отличается от i-го бита в родителе 2 для $i \in [1, n]$. Мы случайно отбираем точку скрещивания $c \in [1, n]$. Какова вероятность того, что дети будут клонами родителей (то есть идентичными им)?

3.8. Предположим, что у нас есть N случайно инициализированных особей в бинарном генетическом алгоритме, где каждая особь состоит из n бит.

a) Какова вероятность того, что i-й бит каждой особи одинаков для заданного i?

b) Какова вероятность того, что i-й бит каждой особи не одинаков для всех $i \in [1, n]$?

c) Напомним, что $\exp(-am) \approx 1 - am$ для малых значений am и $(1 - a)^m \approx 1 - am$ для малых значений a. Используйте эти факты, для того чтобы аппроксимировать свой ответ на вопрос в части (b) как экспоненциальную функцию.

d) Используйте свой ответ на вопрос в части (c), для того чтобы найти размер популяции N, необходимый для получения вероятности p, что обе аллели встречаются в каждой битовой позиции случайно инициализированной популяции.

e) Предположим, что мы хотим случайно инициализировать популяцию, состоящую из особей, каждую из 100 бит, так чтобы имелся 99.9%-ный или больший шанс, что обе аллели будут встречаться в каждой битовой позиции. Используйте свой ответ на вопрос в части (d), для того чтобы получить минимальный размер популяции.

3.9. Дан бинарный генетический алгоритм с размером популяции N и скоростью мутации ρ, и каждая особь состоит из n бит.

a) Какова вероятность того, что мы не мутируем какие-либо биты во всей популяции в течение одного поколения?

b) Используйте свой ответ на вопрос в части (a), для того чтобы найти минимальную скорость мутации ρ для заданного размера популяции N и битовой длины n – такую, что вероятность отсутствия мутаций в течение каждого поколения не превышает P_{none}.

c) Используйте свой ответ на вопрос в части (b), для того чтобы найти минимальную скорость мутации ρ, такую что вероят-

ность отсутствия мутации каких-либо бит составляет 0.01 %, когда $N = 100$ и $n = 100$.

Компьютерные упражнения

3.10. Напишите компьютерную симуляцию для подтверждения ваших ответов на задачу 3.3.

3.11. Напишите компьютерную симуляцию для подтверждения вашего ответа на задачу 3.8.

3.12. Задача нахождения подстроки one-max заключается в поиске подстроки из n бит с максимально возможным числом единиц. Приспособленность битовой строки – это число единиц. Конечно, мы можем легко решить эту задачу, просто записав n последовательных единиц, но в данной задаче нам интересно посмотреть, сможет ли генетический алгоритм решить задачу нахождения подстроки one-max. Напишите генетический алгоритм для решения задачи нахождения подстроки one-max. Используйте $n = 30$, лимит поколений = 100, размер популяции = 20 и скорость мутаций = 1 %.

a) Постройте график приспособленности лучших особей и средней приспособленности популяции как функции от номера поколения.

b) Выполните 50 симуляций Монте-Карло для вашего генетического алгоритма. В результате получится 50 графиков приспособленности лучшей особи как функции номера поколения. Постройте график среднего значения этих 50 графиков. Обозначьте среднее значение 50 лучших значений приспособленности в 100-м поколении как $\bar{f}(x^*)$. Каким будет $\bar{f}(x^*)$?

c) Повторите часть (b) с размером популяции 40. Как будет меняться $\bar{f}(x^*)$, по сравнению с вашим ответом на вопрос части (b)? Почему?

d) Верните размер популяции обратно в 20 и поменяйте скорость мутации на 5 %. Как изменится $\bar{f}(x^*)$, по сравнению с вашим ответом на вопрос части (b)? Почему?

e) Установите скорость мутации равной 0 %. Как изменится $\bar{f}(x^*)$, по сравнению с вашим ответом на вопрос части (b)? Почему?

f) Вместо того чтобы принимать приспособленность равной числу единиц, установите приспособленность равной числу единиц плюс 50. Теперь повторите часть (b). Как изменится $\overline{f}(x^*)$, по сравнению с вашим ответом на вопрос части (b)? Почему?

g) Как и в части (b), установите приспособленность равной числу единиц; но затем для всех особей с приспособленностью меньше средней установите приспособленность равной 0. Как изменится $\overline{f}(x^*)$, по сравнению с вашим ответом на вопрос части (b)? Почему?

3.13. Напишите непрерывный генетический алгоритм для минимизации сферической функции (см. раздел C.1.1). Установите поисковую область в каждой размерности равной [−5, +5], размерность задачи равной 20, лимит поколений равным 100, размер популяции равным 20 и скорость мутации равной 1 %. Для того чтобы выполнить рулеточный отбор, нам нужно увязать значения стоимости $c(x_i)$ со значениями приспособленности $f(x_i)$. Сделайте это следующим образом: $f(x_i) = 1/c(x_i)$.

a) Постройте график стоимости лучшей особи и средней стоимости популяции как функции номера поколения.

b) Выполните 50 симуляций Монте-Карло для вашего генетического алгоритма. В результате получится 50 графиков стоимости лучшей особи как функции номера поколения. Постройте график среднего значения этих 50 графиков. Обозначьте среднее значение 50 лучших значений стоимости в 100-м поколении как $\overline{c}(x^*)$. Каким будет $\overline{c}(x^*)$?

c) Повторите часть (b) со скоростью мутации 2 %. Как изменится $\overline{c}(x^*)$, по сравнению с вашим ответом на вопрос части (b)? Повторите со скоростью мутации 5 %.

Глава 4

Математические модели генетических алгоритмов

Но исходный код программы не обязательно является самым проницательным описанием.

– Майкл Вос (Michael Vose)
[неопубликованные заметки по курсу, 2010]

Изучение эволюционных алгоритмов (evolutionary algorithm, EA) часто было ситуативным, симуляционным, эвристическим и неаналитическим. Исторически сложилось так, что инженеры были больше озабочены вопросом о том, работают или нет эволюционные алгоритмы, а не о том, как и почему они работают. Тем не менее с развитием эволюционно-алгоритмических исследований в последние пару десятилетий XX века инженеры стали больше фокусироваться на вопросах «как и почему». В этой главе мы обсудим некоторые варианты ответа на эти вопросы относительно генетических алгоритмов. Данная глава является самой технической и математической в этой книге. Студент, который хочет получить об эволюционных алгоритмах лишь рабочее знание, может пропустить эту главу. Вместе с тем для тех студентов, которые хотят стать хорошо информированными и всесторонне развитыми в области эволюционно-алгоритмических исследований, важно усвоить идеи данной главы. Студент, который потратит свои время и усилия, для того чтобы понять этот материал, возможно, найдет неожиданные и совершенно новые направления исследований.

Краткий обзор главы

Одним из первых ответов на вопросы «как и почему» была теория схем, которая анализирует рост и распад с течением времени различных ком-

бинаций бит в генетических алгоритмах, и поэтому мы обсудим этот вопрос в разделе 4.1. Некоторые из последних математических генетико-алгоритмических анализов основывались на марковских моделях и системно-динамических моделях, и мы также рассмотрим эти подходы в данной главе. Эти модели имеют свои недостатки, но их недостатки относятся к области реализации и вычислительных ресурсов, а не к теории. Раздел 4.2 дает обзор теории Маркова, которая была разработана российским математиком Андреем Марковым[1] в 1906 году [Seneta, 1966]. Теория Маркова стала фундаментальной областью математики с приложениями в физике, химии, информатике, социальных науках, технике, биологии, музыке, легкой атлетике и других удивительных областях. В этой главе мы увидим, что теория Маркова также может давать более глубокое представление о поведении генетического алгоритма. В разделе 4.3 представлены некоторые обозначения и предварительные результаты, которые мы будем использовать в последующих разделах при разработке марковских моделей и системно-динамическмх моделей.

Раздел 4.4 разворачивает модель марковских цепей для генетического алгоритма, в котором применяется отбор на основе приспособленности, затем мутация, а потом одноточечное скрещивание. К сожалению, размерность марковской модели растет факториально (то есть быстрее, чем экспоненциально) вместе с ростом размера популяции и размера поискового пространства. Это ограничивает его применение очень небольшими задачами. Однако марковские модели по-прежнему полезны для получения точных результатов, не требуя необходимости полагаться на случайный характер стохастических симуляций.

В разделе 4.5 разрабатывается системно-динамическая модель для генетического алгоритма. Системно-динамическая модель основана на марковской модели, но ее приложение совсем другое. Марковская модель дает стационарную вероятность по мере приближения числа поколений к бесконечности каждой возможной популяции. Системно-динамическая модель дает варьирующуюся со временем долю каждой особи в поисковом пространстве по мере приближения размера популяции к бесконечности.

4.1. Теория схем

Рассмотрим простую задачу $\max_x f(x)$, где $f(x) = x^2$. Предположим, что мы кодируем x как 5-битное целое число, где битовая строка 00000 пред-

[1] Сын Маркова Андрей Марков-младший также был опытным математиком.

ставляет десятичное число 0, а битовая строка 11111 представляет десятичное число 31. Максимум от $f(x)$ происходит, когда $x = 11111$. Не только эта, но и любая битовая строка, начинающаяся с 1, лучше, чем каждая битовая строка, начинающаяся с 0. Эти рассуждения подводят к концепции схемы. Схема – это битовый шаблон, описывающий множество особей, где * используется для представления «все равно какого» бита. Например, битовые строки 11000 и 10011 принадлежат схеме 1****. Эта схема является «высоко приспособленной» схемой для функции x^2. Любая битовая строка, принадлежащая этой схеме, лучше любой битовой строки, не принадлежащей ей. Генетические алгоритмы комбинируют схемы таким образом, который приводит к высоко приспособленным особям.

Рассмотрим битовые строки длины 2. Схемами с длиной 2 будут **, 0*, 1*, *0, *1, 00, 01, 10 и 11. Всего в общей сложности девять уникальных схем длины 2. В общем случае имеется в общей сложности 3^l схем длины l.

Теперь рассмотрим число схем, к которым принадлежит битовая строка. Например, обратите внимание, что 01 принадлежит к четырем схемам: 01, *1, 0* и **. В общем случае битовая строка длины l принадлежит к 2^l схемам.

Теперь рассмотрим популяцию из N битовых строк, каждая длиной l. Каждая битовая строка в популяции принадлежит к определенному множеству схем. Мы говорим, что объединение этих N множеств схем есть множество, к которому принадлежит вся популяция в целом. Если все битовые строки идентичны, то каждая битовая строка принадлежит к одним и тем же 2^l схемам, а вся популяция в целом принадлежит к 2^l схемам. С другой стороны, все битовые строки могут быть уникальными и не принадлежать к одним и тем же схемам, за исключением универсальной схемы **...**. В этом случае вся популяция в целом принадлежит к $N2^l - (N - 1)$ схемам. Мы видим, что популяция из N битовых строк, каждая длиной l, принадлежит где-то между к 2^l и $(N(2^l - 1) + 1)$ схемам.

Число определенных бит (то есть не звездочек) в схеме называется порядком o схемы. Например, $o(1**0) = 2$ и $o(0*11*) = 3$.

Число бит от самого левого определенного бита до самого правого определенного бита в схеме называется определяющей длиной σ. Например, $\sigma(1***0) = 4$, $\sigma(0*11*) = 3$ и $\sigma(1****) = 0$.

Битовая строка, принадлежащая схеме, называется экземпляром схемы. Например, схема 0*11* имеет четыре экземпляра: 00110, 00111, 01110 и 01111. В общем случае число экземпляров схемы равно 2^A, где A – это число звездочек в схеме. Обратите внимание, что $A = l - o$.

Мы используем запись $m(h, t)$ для обозначения в генетическом алгоритме числа экземпляров схемы h в поколении t. Мы используем $f(x)$ для обозначения приспособленности битовой строки x. Мы используем $f(h, t)$ для обозначения средней приспособленности экземпляров схемы h в популяции в поколении t:

$$f(h,t) = \frac{\sum_{x \in h} f(x)}{m(h,t)}. \tag{4.1}$$

Мы используем $\overline{f}(t)$ для обозначения средней приспособленности всей популяции в поколении t. Если мы применяем рулеточный отбор для отбора родителей следующего поколения, то видим, что ожидаемое число экземпляров h после выбора равняется

$$E\left[m(h,t+1)\right] = \frac{\sum_{x \in h} f(x)}{\overline{f}(t)} = \frac{f(h,t)m(h,t)}{\overline{f}(t)}. \tag{4.2}$$

Далее мы выполняем скрещивание с вероятностью pc. Мы исходим из того, что точка скрещивания находится между битами, и никогда в конце битовой строки. От каждой пары родителей мы получаем по паре детей. Какова вероятность того, что скрещивание разрушит схему? Давайте рассмотрим несколько примеров.

- Рассмотрим схему h = 1****. Скрещивание никогда не разрушит эту схему. Если экземпляр этой схемы пересекается с другой битовой строкой, по крайней мере, один ребенок будет экземпляром h.
- Рассмотрим схему h = 11***. Если экземпляр этой схемы пересекается с битовой строкой x, то точкой скрещивания может быть одно из четырех мест. Если точка скрещивания находится между двумя наиболее значащими битами, то схема, возможно, будет разрушена, в зависимости от значения x. Однако если точка скрещивания находится справа от этой точки (три другие возможные точки скрещивания), то схема никогда не будет разрушена; по крайней мере, один ребенок будет экземпляром h. Мы видим, что вероятность разрушения схемы h меньше или равна 1/4, в зависимости от того, где происходит скрещивание.
- Рассмотрим схему h = 1*1**. Если экземпляр этой схемы пересекается с битовой строкой x, то точкой скрещивания может быть одно из четырех мест. Если точка скрещивания находится меж-

ду двумя единичными битами (две возможные точки скрещивания), то схема, возможно, будет разрушена, в зависимости от значения *x*. Однако если точка скрещивания находится справа от самого правого единичного бита (две другие возможные точки скрещивания), то схема никогда не будет разрушена; по крайней мере, один ребенок будет экземпляром *h*. Мы видим, что вероятность разрушения схемы *h* меньше или равна 1/2, в зависимости от того, где происходит скрещивание.

Обобщая вышесказанное, мы видим, что вероятность того, что скрещивание разрушит схему, если оно произойдет, меньше или равна $\sigma / (l - 1)$. Вероятность того, что скрещивание вообще произойдет, равна p_c, поэтому общая вероятность того, что скрещивание разрушит схему, меньше или равна $p_c \sigma / (l - 1)$. Следовательно, вероятность того, что схема скрещивания выживет, равна

$$p_s \geq 1 - p_c \left(\sigma / l - 1\right). \tag{4.3}$$

Далее мы выполняем мутацию с вероятностью p_m на бит. Число определенных (не звездочек) бит в *h* является порядком *h* и обозначается как $o(h)$. Вероятность того, что определенный бит мутирует, равна p_m, а вероятность того, что он не мутирует, равна $1 - p_m$. Поэтому вероятность того, что ни один из заданных битов не мутирует, равна $(1 - p_m)^{o(h)}$.

Эта вероятность имеет вид $g(x) = (1 - x)^y$. Разложение в ряд Тейлора $g(x)$ вокруг x_0 равно

$$g(x) = \sum_{n=0}^{\infty} g^{(n)}(x_0) \frac{(x - x_0)^n}{n!}. \tag{4.4}$$

Принятие $x_0 = 0$ дает

$$\begin{aligned} g(x) &= \sum_{n=0}^{\infty} g^{(n)}(x_0) \frac{x^n}{n!} \\ &= 1 - xy + \frac{x^2 y(y-1)}{2!} - \frac{x^3 y(y-1)(y-2)}{3!} + \cdots \\ &\approx 1 - xy \text{ для } xy \ll 1. \end{aligned} \tag{4.5}$$

Поэтому если $p_m o(h) \ll 1$, то $(1 - p_m)^{o(h)} \approx 1 - p_m o(h)$. Связав это с уравнениями (4.2) и (4.3), в результате получим

$$E\left[m(h,t+1)\right] \geq \frac{f(h,t)m(h,t)}{\overline{f}(t)}\left(1-p_c\left(\frac{\sigma}{l-1}\right)\right)\left(1-p_m o(h)\right)$$
$$\approx \frac{f(h,t)m(h,t)}{\overline{f}(t)}\left(1-p_c\left(\frac{\sigma}{l-1}\right)-p_m o(h)\right). \qquad (4.6)$$

Предположим, что схема короткая, то есть ее определяющая длина σ мала. Тогда $\sigma/(l-1) \ll 1$. Предположим, что мы используем низкую скорость мутации, а схема имеет низкий порядок, то есть существует не так много определенных бит. Тогда $p_m o(h) \ll 1$. Предположим, что схема имеет приспособленность выше среднего, то есть $f(h)/\overline{f}(t) = k > 1$, где k – некоторая константа. Наконец, предположим, что у нас большая популяция, такая что $E[m(h, t+1)] \approx m(h, t+1)$. Тогда можно приближенно записать

$$m(h, t+1) \geq km(h, t) = k^t m(h, 0). \qquad (4.7)$$

В результате приходим к следующей ниже теореме, которая называется теоремой схем.

Теорема 4.1. Короткие низкопорядковые схемы со значениями приспособленности выше среднего получают экспоненциально увеличивающееся число представителей в популяции генетического алгоритма.

Теорема схем часто записывается как уравнение (4.6) или (4.7).

Теория схем возникла благодаря Джону Холланду в 1970-х годах [Holland, 1975] и быстро закрепилась в генетико-алгоритмических исследованиях. Теория схем была настолько доминирующей в 1980-х годах, что реализации генетических алгоритмов были подозрительными, если они нарушали ее допущения (например, если они использовали ранговый отбор, а не отбор на основе приспособленности [Whitley, 1989]). Описание того, как теория схем работает на простом примере, приведено в публикации [Goldberg, 1989a, глава 2].

Однако некоторые контрпримеры теории схем приведены в публикации [Reeves и Rowe, 2003, раздел 3.2]. То есть теорема схем не всегда полезна. Это объясняется следующим.

- Теория схем применяется к произвольным подмножествам поискового пространства. Рассмотрим табл. 3.2 в примере 3.2. Мы видим, что x_1 и x_4 принадлежат схеме $h = 1*0*$. Но это наименее при-

способленные и наиболее приспособленные особи в популяции, и поэтому эти две особи на самом деле не имеют ничего общего друг с другом, кроме того факта, что они обе являются членами h. В h нет ничего особенного, поэтому теорема схем не дает полезной информации об h.

- Теория схем не распознает, что похожие битовые строки, возможно, не принадлежат к одной схеме. В примере 3.2 мы видим, что 0111 и 1000 являются соседями в поисковом пространстве, но не принадлежат к какой-либо общей схеме, кроме универсальной схемы ****. Эта задача может быть решена с помощью кодов Грея, но даже тогда, в зависимости от поискового пространства соседи в поисковом пространстве могут не иметь тесной связи в пространстве приспособленности.
- Теория схем говорит нам о числе экземпляров схемы, которые выживают из поколения в поколение, но что более важно, какие экземпляры схемы выживают. Это тесно связано с пунктом 1 выше. Опять же, глядя на пример 3.2, мы видим, что x_1 и x_4 принадлежат схеме h = 1*0*. Теория схем говорит нам, если h выживает из поколения в поколение, но мы гораздо больше заинтересованы в выживании x_4, чем x_1.
- Теория схем дает нам ожидаемое число экземпляров схемы. Но стохастическая природа генетических алгоритмов приводит к разному поведению в каждом прогоне генетического алгоритма. Ожидаемое число экземпляров схемы равно фактическому числу экземпляров схемы только по мере приближения размера популяции к бесконечности.
- Ни одна схема не может одновременно увеличиваться в геометрической прогрессии и иметь приспособленность выше среднего. Если схема увеличивается в геометрической прогрессии, то вскоре она будет доминировать в популяции, и в это время средняя приспособленность популяции будет примерно равна приспособленности схемы. Поэтому аппроксимация $f(h) / \bar{f}(t) = k$ в абзаце перед приведенной выше теоремой 4.1, где k является константой, неверна. К этой идее относится тот факт, что бóльшая часть генетических алгоритмов работает с популяцией 100 или менее особей. Такие малые размеры популяции не могут поддерживать экспоненциальный рост любой схемы более чем на пару поколений.

К 1990-м годам чрезмерный акцент на недостатках теории схем привел к крайним утверждениям, таким как: «поскольку уже никто не спорит, что теорема схем практически ничего не объясняет о поведении SGA [simple genetic algorithm, простого генетического алгоритма]» [Vose, 1999, стр. xi]. Маятник качнулся с одной стороны (чрезмерная зависимость от теории схем) в другую (полное отклонение теории схем). Такая высокая изменчивость характерна для многих новых теорий. Теория схем верна, но у нее есть ограничения. Сбалансированное представление о преимуществах и недостатках теории схем дается в публикации [Reeves и Rowe, 2003, глава 3].

4.2. Цепи Маркова

Предположим, что мы имеем дискретно-временную систему, которая может быть описана множеством дискретных состояний $S = \{S_1, \ldots, S_n\}$. Например, погода вполне может быть описана множеством состояний S = {дождливая, хорошая, снежная}. Мы используем обозначение $S(t)$ для обозначения состояния на временном шаге t. Исходное состояние равняется $S(0)$, состояние на следующем шаге времени – $S(1)$ и так далее. Состояние системы может из одного временного шага в другой изменяться либо из одного временного шага в другой оставаться в том же состоянии. Переход из одного состояния в другое является полностью вероятностным. В марковском процессе первого порядка, так называемой марковской цепи первого порядка, вероятность того, что система перейдет в какое-либо заданное состояние на следующем временном шаге, зависит только от текущего состояния, то есть вероятность не зависит от всех предыдущих состояний. Вероятность того, что система перейдет из состояния i в состояние j с одного временного шага на следующий, обозначается p_{ij}. Следовательно,

$$\sum_{j=1}^{n} p_{ij} = 1 \tag{4.8}$$

для всех i. Мы формируем $n \times n$-матрицу P, где p_{ij} – это элемент в i-й строке и j-м столбце. P называется транзитной матрицей, матрицей переходов, матрицей вероятностей или стохастической матрицей марковского процесса[2]. Сумма элементов каждой строки в матрице P равна 1.

[2] Если быть точным, то матрица, которую мы определили, называется правой транзитной матрицей. Некоторые книги и статьи обозначают вероятность перехода как pji и определяют марковскую транзитную матрицу как транспонирование той, которую мы определили. Их матрица называется левой транзитной матрицей, и сумма элементов каждого столбца равна 1.

■ Пример 4.1

В стране Оз никогда не бывает двух прекрасных дней подряд [Kemeny и соавт., 1974]. Если сегодня – прекрасный день, то на следующий день есть 50%-ный шанс, что будет дождь, и 50%-ный шанс, что будет снег. Если сегодня идет дождь, то на следующий день есть 50%-ный шанс, что снова будет идти дождь, 25%-ный шанс, что пойдет снег, 25%-ный шанс, что будет прекрасная погода. Если сегодня идет снег, то на следующий день есть 50%-ный шанс, что снова будет идти снег, 25%-ный шанс, что пойдет дождь, и 25%-ный шанс, что будет прекрасная погода. Мы видим, что прогноз погоды на данный день зависит исключительно от погоды предыдущего дня. Если присвоить состояния *R*, *N* и *S* соответственно дождю, хорошей погоде и снегу, то можно сформировать марковскую матрицу, представляющую вероятность различных погодных переходов:

$$P = \begin{matrix} \\ R \\ N \\ S \end{matrix} \begin{matrix} \begin{matrix} R & N & S \end{matrix} \\ \begin{bmatrix} 1/2 & 1/4 & 1/4 \\ 1/2 & 0 & 1/2 \\ 1/4 & 1/4 & 1/2 \end{bmatrix} \end{matrix} \tag{4.9}$$

■

Предположим, что марковский процесс начинается в состоянии *i* в момент времени 0. Из нашего предыдущего обсуждения мы знаем, что вероятность того, что процесс находится в состоянии *j* в момент времени 1, при условии что процесс находился в состоянии *i* в момент времени 0, задается формулой $\Pr(S(1) = S_j \mid S(0) = S_i) = p_{ij}$. Далее рассмотрим следующий временной шаг. Мы можем применить теорему о полной вероятности [Mitzenmacher и Upfal, 2005] и найти вероятность того, что процесс находится в состоянии 1 в момент времени 2, следующим образом:

$$\begin{aligned} \Pr(S(2) = S_1 \,|\, S(0) = S_i) &= \Pr(S(1) = S_1 \,|\, S(0) = S_i)\, p_{11} + \\ &\quad \Pr(S(1) = S_2 \,|\, S(0) = S_i)\, p_{21} + \cdots + \\ &\quad \Pr(S(1) = S_n \,|\, S(0) = S_i)\, p_{n1} \\ &= \sum_{k=1}^{n} \Pr(S(1) = S_k \,|\, S(0) = S_i)\, p_{k1} \\ &= \sum_{k=1}^{n} p_{ik}\, p_{k1}. \end{aligned} \tag{4.10}$$

Обобщая вышеизложенное преобразование, мы находим, что вероятность того, что процесс находится в состоянии j в момент времени 2, задается формулой

$$\Pr(S(2) = S_j \,|\, S(0) = S_i) = \sum_{k=1}^{n} p_{ik} p_{kj}. \tag{4.11}$$

Но это равно элементу в i-й строке и j-м столбце квадрата P, то есть

$$\Pr(S(2) = S_j \,|\, S(0) = S_i) = \left[P^2 \right]_{ij}. \tag{4.12}$$

Продолжая эту линию рассуждений индуктивным образом, мы находим, что

$$\Pr(S(t) = S_j \,|\, S(0) = S_i) = \left[P^t \right]_{ij}. \tag{4.13}$$

То есть вероятность того, что марковский процесс перейдет из состояния i в состояние j после t временных шагов, равна элементу в i-й строке и j-м столбце матрицы P^t. В примере 4.1 мы можем вычислить P^t для разных значений t и получим:

$$\begin{aligned}
P &= \begin{bmatrix} 0.5000 & 0.2500 & 0.2500 \\ 0.5000 & 0.0000 & 0.5000 \\ 0.2500 & 0.2500 & 0.5000 \end{bmatrix} \\
P^2 &= \begin{bmatrix} 0.4375 & 0.1875 & 0.3750 \\ 0.3750 & 0.2500 & 0.3750 \\ 0.3750 & 0.1875 & 0.4375 \end{bmatrix} \\
P^4 &= \begin{bmatrix} 0.4023 & 0.1992 & 0.3984 \\ 0.3984 & 0.2031 & 0.3984 \\ 0.3984 & 0.1992 & 0.4023 \end{bmatrix} \\
P^8 &= \begin{bmatrix} 0.4000 & 0.2000 & 0.4000 \\ 0.4000 & 0.2000 & 0.4000 \\ 0.4000 & 0.2000 & 0.4000 \end{bmatrix}
\end{aligned} \tag{4.14}$$

Интересно, что P^t сходится как $t \to \infty$ к матрице с идентичными строками. Это не происходит для всех транзитных матриц, но это происходит для некоторого их подмножества, как указано в следующей ниже теореме.

Теорема 4.2. Регулярная транзитная $n \times n$-матрица P, так называемая примитивная транзитная матрица, есть матрица, для которой все элементы Pl ненулевые для некоторого t. Если P – это регулярная транзитная матрица, то

1) $\lim_{t \to \infty} P^t = P_\infty$;
2) все строки P_∞ идентичны и обозначаются как p_{ss};
3) каждый элемент p_{ss} положителен;
4) вероятность того, что марковский процесс находится в i-м состоянии после бесконечного числа переходов, равна i-му элементу p_{ss};
5) P_{js}^T является собственным вектором P^T, соответствующим собственному значению 1, нормализованному так, что его элементы в сумме составляют 1;
6) если мы образуем матрицы P_i, $i \in [1, n]$, заменив i-й столбец P на нули, то i-й элемент p_{ss} выражается следующей формулой:

$$p_{ss,i} = \frac{\left|P_i - I\right|}{\sum_{j=1}^{n} \left|P_i - I\right|}, \tag{4.16}$$

где I – это единичная $n \times n$-матрица и $|\cdot|$ – детерминантный оператор.

Доказательство: первые пять вышеприведенных свойств составляют основную предельную теорему регулярных цепей Маркова и доказаны в книге [Grinstead и Snell, 1997, глава 11] и других книгах по цепям Маркова. Для получения дополнительной информации о таких понятиях, как детерминанты, собственные значения и собственные векторы, прочитайте любое пособие по линейным системам [Simon, 2006, глава 1]. Последнее свойство теоремы 4.2 доказано в публикации [Davis и Principe, 1993].

■

■ Пример 4.2

Применив уравнение (4.14) и теорему 4.2 к примеру 4.1, мы видим, что любой день в отдаленном будущем имеет 40%-ную вероятность дождя, 20%-ную вероятность солнца и 40%-ную вероятность снега. Поэтому 40 % дней в стране Оз – дождливые, 20 % – солнечные и 40 % – снежные. Кроме того, мы можем найти собственные значения PT как 1,

–0.25 и 0.25. Собственный вектор, соответствующий собственному значению 1, равен $[0.4 \;\; 0.2 \;\; 0.4]^T$.

■

Теперь предположим, что мы не знаем исходного состояния марковского процесса, но знаем вероятности для каждого состояния; вероятность того, что исходное состояние $S(0)$ равно S_k, задается выражением $p_k(0), k \in [1, n]$. Затем мы можем применить теорему о полной вероятности [Mitzenmacher и Upfal, 2005] и получить

$$\begin{aligned} \Pr(S(1) = S_i) &= \Pr(S(0) = S_1) p_{1i} + \Pr(S(0) = S_2) p_{2i} + \cdots + \\ &\quad \Pr(S(0) = S_n) p_{ni} \\ &= \sum_{k=1}^{n} \Pr(S(0) = S_k) p_{ki} \\ &= \sum_{k=1}^{n} p_{ki} p_k(0). \end{aligned} \tag{4.16}$$

Обобщая приведенное выше уравнение, получаем

$$\begin{bmatrix} \Pr(S(1) = S_1) \\ \cdots \\ \Pr(S(1) = S_n) \end{bmatrix}^T = p^T(0)P, \tag{4.17}$$

где $p(0)$ – это вектор-столбец, состоящий из $p_k(0), k \in [1, n]$. Обобщая это разложение на несколько временных шагов, получаем

$$p^T(t) = \begin{bmatrix} \Pr(S(t) = S_1) \\ \cdots \\ \Pr(S(t) = S_n) \end{bmatrix}^T = p^T(0)P^t. \tag{4.18}$$

■ Пример 4.3

Прогноз погоды на сегодня в стране Оз: 80 % солнце и 20 % снег. Каким будет прогноз погоды на ближайшие два дня?

Из уравнения (4.18), $p^T(2) = p^T(0)P^2$, где P задается в примере 4.1 и $p(0) = [0.0 \;\; 0.8 \;\; 0.2]^T$. Это дает $p(2) = [0.3750 \;\; 0.2375 \;\; 0.3875]^T$. То есть через два дня есть 37.5%-ный шанс дождливой погоды, 23.75%-ный шанс солнечной погоды и 38.75%-ный шанс снежной погоды.

■

■ Пример 4.4

Рассмотрим простой эволюционный алгоритм на основе восхождения к вершине холма, состоящий из одной особи [Reeves и Rowe, 2003, стр. 112]. Цель эволюционного алгоритма – минимизировать $f(x)$. Мы используем x_i для обозначения кандидатного решения в i-м поколении. Каждое поколение мы случайно мутируем x_i и получаем x'_i. Если $f(x'_i) < f(x_i)$, то мы принимаем $x_{i+1} = x'_i$.

Если $f(x'_i) > f(x_i)$, то для определения x_{i+1} используется следующая далее логика. Если мы приняли $x_{k+1} = x'_k$ в предыдущем поколении k, в котором $f(x'_k) > f(x_k)$, то мы принимаем $x_{i+1} = x'_i$ с вероятностью 10 % и $x_{i+1} = x'_i$ с вероятностью 90 %. Если, однако, мы приняли $x_{k+1} = x'_k$ в предыдущем поколении k, в котором $f(x'_k) > f(x_k)$, то мы принимаем $x_{i+1} = x'_i$ с вероятностью 50 % и $x_{i+1} = x'_i$ с вероятностью 50 %. Этот эволюционный алгоритм является жадным в том, что он всегда принимает полезные мутации. Вместе с тем он также включает в себя некоторое разведывание в том, что иногда принимает вредные мутации. Вероятность принятия вредной мутации варьируется в зависимости от того, была ли принята предыдущая вредная мутация. Алгоритм этого эволюционного алгоритма показан на рис. 4.1.

```
Инициализировать x_i случайным кандидатным решением
Инициализировать флагПринятия значением false
For i = 1, 2, ...
    Мутировать x_i, получив x'_i
    If f(x'_i) < f(x_i)
        x_{i+1} ← x'_i
    else
        If флагПринятия
                Pr(x_{i+1} ← x'_i) = 0.1, и Pr(x_{i+1} ← x_i) = 0.9
        else
                Pr(x_{i+1} ← x'_i) = 0.5, и Pr(x_{i+1} ← x_i) = 0.5
        end if
        флагПринятия ← (x_{i+1} = x'_i)
    end if
Next i
```

Рис. 4.1. Приведенный выше псевдокод дает схематичное описание эволюционного алгоритма на основе восхождения к вершине холма с одной особью примера 4.4. ФлагПринятия сигнализирует о том, заменила предыдущая вредоносная мутация кандидатное решение или нет

Мы можем проанализировать этот эволюционный алгоритм, рассматривая, что произойдет, если x'_i хуже x_i. Мы используем Z_k для обозначе-

ния состояния k-й раз, когда $f(x_i') > f(x_i)$. Мы определяем Y_1 как состояние «принять», то есть $x_{i+1} \leftarrow x_i'$. Мы определяем Y_2 как состояние «отклонить», то есть $x_{i+1} \leftarrow x_i$. Затем, изучив алгоритм на рис. 4.1, мы можем записать:

$$\begin{aligned} &\Pr(Z_k = Y_1 \mid Z_{k-1} = Y_1) = 0.1 \\ &\Pr(Z_k = Y_2 \mid Z_{k-1} = Y_1) = 0.9 \\ &\Pr(Z_k = Y_1 \mid Z_{k-1} = Y_2) = 0.5 \\ &\Pr(Z_k = Y_2 \mid Z_{k-1} = Y_2) = 0.5 \end{aligned} \tag{4.19}$$

Это уравнение показывает, что транзитная матрица равняется

$$P = \begin{bmatrix} 0.1 & 0.9 \\ 0.5 & 0.5 \end{bmatrix}. \tag{4.20}$$

Обратите внимание, что строки P в сумме составляют 1. Мы также видим, что все элементы P^t ненулевые для некоторого t (в данном случае фактически для всех t), поэтому P является регулярной транзитной матрицей. Теорема 4.2 уверяет нас, что: (1) P^t сходится при $t \to \infty$; (2) все строки P^∞ идентичны; (3) каждый элемент P^∞ является положительным; (4) вероятность того, что марковский процесс в состоянии Y_i после бесконечного числа переходов равен i-му элементу каждой строки P^∞; и (5) каждая строка P^∞ равна транспонированному собственному вектору, соответствующему единичному собственному значению P^T.

Мы применяем численное вычисление и находим

$$P^\infty = \frac{1}{14}\begin{bmatrix} 5 & 9 \\ 5 & 9 \end{bmatrix}. \tag{4.21}$$

Мы также находим, что собственные значения P^T равны –0.4 и 1, а собственный вектор, соответствующий единичному собственному значению, равен $[5/14 \;\; 9/14]^T$. Эти результаты говорят нам о том, что в долгосрочной перспективе соотношение принятий вредных мутаций к их отклонениям равно 5/9.

■

4.3. Обозначения марковской модели для эволюционных алгоритмов

В этом разделе мы определим обозначения, которые будем использовать позже для получения марковской модели и системно-динамиче-

ской модели для эволюционных алгоритмов. Марковские модели могут быть ценным инструментом для анализа эволюционных алгоритмов, поскольку они дают нам точные результаты. Мы можем выполнять симуляции для оценивания результативности эволюционных алгоритмов, но она может вводить в заблуждение. Например, нередко набор симуляций Монте-Карло вполне может приводить к ошибочным результатам из-за специфической последовательности случайных чисел, генерируемых во время симуляции. Кроме того, генератор случайных чисел, используемый в эволюционно-алгоритмической симуляции, может быть некорректным, что случается чаще, чем хотелось бы думать, и будет давать вводящие в заблуждение результаты [Savicky и Robnik-Sikonja, 2008]. Наконец, число симуляций Монте-Карло для оценивания крайне маловероятных исходов вполне может быть настолько велико, что не будет достигнуто за разумное вычислительное время. Результаты марковской модели, которые мы получаем, избегают всех этих ловушек и дают точные результаты. Недостатком марковских моделей является большой объем вычислительных затрат, необходимых для их реализации.

Мы сосредоточимся на эволюционных алгоритмах с размером популяции N, оперирующих в дискретном поисковом пространстве с мощностью n. Будем считать, что поисковое пространство состоит из всех q-битных двоичных строк, так что $n = 2^q$. Мы используем x_i для обозначения i-й битовой строки в указанном поисковом пространстве. Мы используем v для обозначения популяционного вектора, то есть v – это число x_i особей в популяции. Мы видим, что

$$\sum_{i=1}^{n} v_i = N. \tag{4.22}$$

Это уравнение просто означает, что общее число особей в популяции равно N. Мы используем y_k для обозначения k-й особи в популяции. Эволюционно-алгоритмическая популяция Y может быть представлена как

$$\begin{aligned} Y &= \{y_1, \ldots, y_N\} \\ &= \{\underbrace{x_1, x_1, \ldots, x_1}_{v_1 \text{ копий}}, \underbrace{x_2, x_2, \ldots, x_2}_{v_3 \text{ копий}}, \cdots \underbrace{x_n, x_n, \ldots, x_n}_{v_n \text{ копий}}\}, \end{aligned} \tag{4.23}$$

где y_i были упорядочены с целью группирования идентичных особей. Мы используем T для обозначения общего числа возможных популя-

ций Y. То есть T – это число целочисленных $n\times1$-векторов v – таких, что $\sum_{i=1}^{n} v_i = N$ и $v_i \in [0, N]$.

■ Пример 4.5

Предположим, что $N = 2$ и $n = 4$, то есть поисковое пространство состоит из битовых строк {00, 01, 10, 11}, и в эволюционном алгоритме есть две особи. Особями поискового пространства являются

$$\begin{aligned} x_1 &= 00, & x_2 &= 01, \\ x_3 &= 10, & x_4 &= 11. \end{aligned} \tag{4.24}$$

Возможные популяции включают следующие:

$$\begin{aligned} &\{00, 00\}, && \{00, 01\}, \\ &\{00, 10\}, && \{00, 11\}, \\ &\{01, 01\}, && \{01, 10\}, \\ &\{01, 11\}, && \{10, 10\}, \\ &\{10, 11\}, && \{11, 11\}. \end{aligned} \tag{4.25}$$

Мы видим, что для этого примера $T = 10$.

■

Сколько возможных популяций эволюционного алгоритма существует для популяции размером N в поисковом пространстве мощности n? Можно продемонстрировать [Nix и Vose, 1992], что T задается следующим биномиальным коэффициентом, который также называется функцией выбора (choose function):

$$T = \binom{n+N-1}{N}. \tag{4.26}$$

Для того чтобы найти T, мы также можем применить полиномиальную теорему [Chuan-Chong и Khee-Meng, 1992], [Simon и соавт., 2011a]. Полиномиальная теорема может быть сформулирована несколькими способами, в том числе следующим: если дано K классов объектов, то числом разных способов отбора N объектов независимо от порядка, при этом выбирая из каждого класса не более M раз, является коэффициент q_N в многочлене

$$\begin{aligned} q(x) &= (1 + x + x^2 + \cdots + x^M)^K \\ &= 1 + q_1 x + q_2 x^2 + \cdots + q_N x^N + \cdots + x^{MK}. \end{aligned} \tag{4.27}$$

Наш популяционный вектор v эволюционного алгоритма является N-элементным вектором, где каждый элемент представляет собой целое число от 0 до n включительно и элементы которого в сумме составляют N. T – это число уникальных популяционных векторов v. Таким образом, T – это число способов, которыми N объектов может быть выбрано независимо от порядка из N классов объектов, выбирая из каждого класса не более N раз. Применение полиномиальной теоремы (4.27) к этой задаче дает

$$\begin{aligned} T &= q^N, \\ \text{где } q(x) &= (1 + x + x^2 + \cdots + x^N)^n \\ &= 1 + q_1 x + q_2 x^2 + \cdots + q_N x^N + \cdots + x^{Nn}. \end{aligned} \tag{4.28}$$

Для того чтобы найти T, мы также можем применить другую форму полиномиальной теоремы [Chuan-Chong и Khee-Meng, 1992], [Simon и соавт., 2011a]. Эту полиномиальную теорему можно сформулировать следующим образом:

$$\begin{aligned} (x_1 + x_2 + \cdots + x_N)^n &= \sum_{S(k)} \frac{n!}{\prod_{j=0}^{N} k_j!} \prod_{j=0}^{N} x_j^{k_j} \\ &= \sum_{S(k)} \prod_{j=0}^{N} \binom{\sum_{j=0}^{i} k_j}{k_i} \prod_{j=0}^{N} x_j^{k_j}, \\ \text{где } S(k) &= \left\{ k \in \mathbb{R}^N : k_j \in \{0, 1, \ldots, n\}, \sum_{j=0}^{N} k_j = n \right\}. \end{aligned} \tag{4.29}$$

Теперь рассмотрим многочлен $(x^0 + x^1 + x^2 + \ldots + x^N)^n$. Из полиномиальной теоремы уравнения (4.29) мы видим, что коэффициент $[(x^0)^{k_0} + (x^1)^{k_1} + \ldots + (x^N)^{k_N}]$ задается формулой

$$\prod_{j=0}^{N} \binom{\sum_{j=0}^{i} k_j}{k_i}. \tag{4.30}$$

Если сложить эти члены для всех k_j так, что

$$\sum_{j=0}^{N} j k_j = N, \tag{4.31}$$

тогда мы получим коэффициент x^N. Но уравнение (4.28) показывает, что T равно коэффициенту x^N. Следовательно,

$$T = \sum_{S'(k)} \prod_{i=0}^{N} \binom{\sum_{j=0}^{i} k_j}{k_i}, \tag{4.32}$$

$$\text{где } S'(k) = \left\{ k \in \mathbb{R}^{N+1} : k_j \in \{0, 1, \ldots, n\}, \sum_{j=0}^{N} k_j = n, \sum_{j=0}^{N} j k_j = N \right\}.$$

Уравнения (4.26), (4.28) и (4.32) дают эквивалентные выражения для T.

■ Пример 4.6

Этот пример взят из книги [Simon и соавт., 2011a]. Предположим, что дано двухбитное поисковое пространство ($q = 2$, $n = 4$) и размер эволюционно-алгоритмической популяции $N = 4$. Уравнение (4.26) дает

$$T = \binom{7}{4} = 35. \tag{4.33}$$

Уравнение (4.28) дает

$$\begin{aligned} q(x) &= (1 + x + x^2 + x^3 + x^4)^4 \\ &= 1 + \cdots + 35x^4 + \cdots + x^{16}. \end{aligned} \tag{4.34}$$

Из этого следует, что $T = 35$. Уравнение (4.32) дает следующее:

$$T = \sum_{S'(k)} \prod_{i=0}^{4} \binom{\sum_{j=0}^{i} k_j}{k_i},$$

$$\begin{aligned} \text{где } S'(k) &= \left\{ k \in \mathbb{R}^5 : k_j \in \{0, 1, \ldots, 4\}, \sum_{j=0}^{4} k_j = 4, \sum_{j=0}^{4} j k_j = 4 \right\} \\ &= \{(3\,0\,0\,0\,1), (2\,1\,0\,1\,0), (2\,0\,2\,0\,0), (1\,2\,1\,0\,0), (0\,4\,0\,0\,0)\}. \end{aligned} \tag{4.35}$$

Из этого следует, что $T = 4 + 12 + 6 + 12 + 1 = 35$. Мы видим, что все три метода расчета дают одинаковый результат.

■

4.4. Марковские модели генетических алгоритмов

Марковские модели были впервые применены для моделирования генетических алгоритмов в публикациях [Nix и Vose, 1992], [Davis и Principe, 1991] и citeDavis93 и далее объясняются в публикациях [Reeves и Rowe, 2003] и [Vose, 1999]. Как мы видели в главе 3, генетический алгоритм состоит из отбора, скрещивания и мутации. Для целей марковского моделирования поменяем порядок следования скрещивания и мутации, поэтому мы рассмотрим генетический алгоритм, который состоит из отбора, мутации и скрещивания в таком порядке.

4.4.1. *Отбор*

Сначала рассмотрим отбор пропорционально приспособленности (то есть по принципу колеса рулетки). Вероятность отбора индиувидуума x_i с помощью одного вращения колеса рулетки пропорциональна приспособленности особи x_i, умноженной на число особей x_i в популяции. Эта вероятность нормализована так, что все вероятности в сумме составляют 1. Как определено в предыдущем разделе, v_i – это число особей x_i в популяции. Следовательно, вероятность отбора особи x_i с помощью одного вращения колеса рулетки равна

$$P_s(x_i|v) = \frac{v_i f_i}{\sum_{j=1}^{n} v_j f_j}, \tag{4.36}$$

для $i \in [1, n]$, где n – это мощность поискового пространства, а f_j – приспособленность x_j. Мы используем запись $P_s(x_i|v)$, для того чтобы показать, что вероятность отбора особи x_i зависит от популяционного вектора v. С учетом популяции из N особей предположим, что мы вращаем колесо рулетки N раз, чтобы отобрать N родителей. Каждое вращение колеса рулетки имеет n возможных исходов $\{x_1, \ldots, x_n\}$. Вероятность получения исхода x_i при каждом вращении равна $P_s(x_i|v)$. Пусть $U = [U_1 \ldots U_n]$ есть вектор случайных величин, где U_i обозначает общее число раз, когда x_i случается в N вращениях колеса рулетки, и пусть $u = [u_1 \ldots u_n]$ есть реализация U. Теория мультиномиального распределения [Evans и соавт., 2000] говорит нам, что

$$\Pr_s = (u|v) = N! \prod_{i=1}^{n} \frac{\left[P_s(x_i|v)\right]^{u_i}}{u_i!}. \tag{4.37}$$

Это дает нам вероятность получения популяционного вектора u после N вращений колеса рулетки, если мы начнем с популяционного вектора v. Мы используем нижний индекс s в $\Pr_s(u|v)$ для обозначения того, что рассматриваем только отбор (а не мутацию или скрещивание).

Теперь вспомним, что марковская транзитная матрица содержит все вероятности перехода из одного состояния в другое. Уравнение (4.37) дает вероятность перехода из одного популяционного вектора v к другому популяционному вектору u. Как обсуждалось в предыдущем разделе, существует T возможных популяционных векторов. Следовательно, если мы вычислим уравнение (4.37) для каждого возможного u и каждого возможного v, то получим транзитную $T \times T$-матрицу Маркова, которая дает точную вероятностную модель только для отбора в генетическом алгоритме. Каждая запись транзитной матрицы содержит вероятность перехода из некоторого конкретного популяционного вектора к некоторому другому популяционному вектору.

4.4.2. *Мутации*

Теперь предположим, что после отбора мы реализуем мутацию на отобранных особях. Определим M_{ji} как вероятность того, что x_j мутирует в x_i. Тогда вероятность получения индиидуума x_i после одного вращения колеса рулетки, вслед за которым идет единственный шанс мутации, равняется

$$P_{sm}(x_i|v) = \sum_{j=1}^{n} M_{ji} P_s(x_j|v), \tag{4.38}$$

для $i \in [1, n]$. Из этого следует, что мы можем записать n-элементный вектор, i-й элемент которого равен $P_{sm}(x_i|v)$, следующим образом:

$$P_{sm}(x|v) = M^T P_s(x|v), \tag{4.39}$$

где M – это матрица, содержащая M_{ji} в j-й строке и i-м столбце, а $P_s(x|v)$ – n-элементный вектор, j-й элемент которого равен $P_s(x_j|v)$. Теперь мы снова применим теорию полиномиального распределения и найдем, что

$$\Pr_{sm} = (u|v) = N! \prod_{i=1}^{n} \frac{\left[P_{sm}(x_i|v)\right]^{u_i}}{u_i!}. \tag{4.40}$$

Это дает нам вероятность получения популяционного вектора u, если мы начнем с популяционного вектора v, после того как произой-

дут отбор и мутация. Если мы вычислим уравнение (4.40) для каждого из T возможных популяционных векторов u и v, то у нас будет транзитная $T \times T$-матрица Маркова, которая дает точную вероятностную модель генетического алгоритма, состоящую как из выбора, так и из мутации.

Если мутация определена так, что $M_{ji} > 0$ для всех i и j, тогда $\Pr_{sm}(u|v) > 0$ для всех u и v. Из этого следует, что транзитная матрица Маркова будет содержать только положительные записи, а это значит, что транзитная матрица будет регулярной. Теорема 4.2 говорит нам, что будет иметься уникальная ненулевая вероятность получения каждого возможного популяционного распределения. Из этого следует, что в долгосрочной перспективе каждое возможное популяционное распределение будет случаться в течение ненулевого процента времени. Данные проценты могут быть рассчитаны с использованием теоремы 4.2 и транзитной матрицы, полученной из уравнения (4.40). Генетический алгоритм не будет сходиться к какой-либо конкретной популяции, а будет бесконечно блуждать по всему поисковому пространству, попадая в каждую возможную популяцию в течение процента времени, указанного в теореме 4.2.

■ Пример 4.7

Предположим, что у нас есть четырехэлементное поисковое пространство с особями $x = \{00, 01, 10, 11\}$. Предположим, что каждый бит в каждой особи имеет 10%-ный шанс мутации. Вероятность того, что 00 останется равным 00 после шанса мутации, равного вероятности, что первый 0-й бит останется неизменным (90%), умноженной на вероятность того, что второй 0-й бит останется неизменным (90%), давая вероятность 0.81. В итоге получаем M_{11}, то есть вероятность того, что x_i остается неизменным после шанса мутации. Вероятность того, что 00 изменится на 01, равна вероятности того, что первый 0-й бит останется неизменным (90%), умноженной на вероятность того, что второй 0-й бит изменится на 1 (10%), что в итоге дает вероятность $M_{12} = 0.09$. Продолжая в этом направлении, мы обнаруживаем, что

$$M = \begin{bmatrix} 0.81 & 0.09 & 0.09 & 0.01 \\ 0.09 & 0.81 & 0.01 & 0.09 \\ 0.09 & 0.01 & 0.81 & 0.09 \\ 0.01 & 0.09 & 0.09 & 0.81 \end{bmatrix}. \tag{4.41}$$

Заметим, что M симметрична (то есть M равна ее транспонированию M^T). Это обычно (но не всегда) тот случай, который означает, что x_i с одинаковой вероятностью мутирует в x_j, так же как x_j мутирует в x_i.

■

4.4.3. *Скрещивание*

Теперь предположим, что после отбора и мутации мы реализуем скрещивание. Пусть r_{jki} обозначает вероятность того, что x_j и x_k скрещиваются, формируя x_i. Тогда вероятность получения x, после двух вращений колеса рулетки, за которыми следует единственный шанс мутации для каждой отобранной особи, за которым следует скрещивание, равна

$$P_{smc}(x_i \mid v) = \sum_{j=1}^{n}\sum_{k=1}^{n} r_{jki} P_{sm}(x_j \mid v) P_{sm}(x_k \mid v). \tag{4.42}$$

Теперь мы снова воспользуемся теорией полиномиального распределения и найдем, что

$$\Pr_{smc} = (u \mid v) = N! \prod_{i=1}^{n} \frac{\left[P_{smc}(x_i \mid v)\right]^{u_i}}{u_i!}. \tag{4.43}$$

Это дает нам вероятность получения популяционного вектора u, если мы начнем с популяционного вектора v после отбора, мутации и скрещивания.

■ Пример 4.8

Предположим, что у нас есть четырехэлементное поисковое пространство с особями $\{x_1, x_2, x_3, x_4\} = \{00, 01, 10, 11\}$. Предположим, что мы реализуем скрещивание, случайно назначая $b = 1$ или $b = 2$ с равной вероятностью, а затем сцепляя биты $1 \rightarrow b$ от первого родителя с битами $(b + 1) \rightarrow 2$ от второго родителя. Некоторые возможности скрещивания можно записать следующим образом:

$$\begin{aligned} &00 \times 00 \rightarrow 00 \\ &00 \times 01 \rightarrow 01 \text{ или } 00 \\ &00 \times 10 \rightarrow 00 \\ &00 \times 11 \rightarrow 01 \text{ или } 00 \end{aligned} \tag{4.44}$$

В результате получим вероятности скрещивания:

$$\begin{array}{llll} r_{111} = 1.0, & r_{112} = 0.0, & r_{113} = 0.0, & r_{114} = 0.0 \\ r_{121} = 0.5, & r_{122} = 0.5, & r_{123} = 0.0, & r_{124} = 0.0 \\ r_{131} = 1.0, & r_{132} = 0.0, & r_{133} = 0.0, & r_{134} = 0.0 \\ r_{141} = 0.5, & r_{142} = 0.5, & r_{143} = 0.0, & r_{144} = 0.0 \end{array} \quad (4.45)$$

Другие значения r_{jki} могут быть вычислены аналогичным образом.

■

■ Пример 4.9

В данном примере мы рассмотрим трехбитную задачу *one-max* нахождения подстроки с максимально возможным числом единиц. Значение приспособленности каждой особи пропорционально числу единиц в особи:

$$\begin{array}{llll} f(000) = 1, & f(100) = 2, & f(001) = 2, & f(101) = 3, \\ f(010) = 2, & f(110) = 3, & f(011) = 3, & f(111) = 4. \end{array} \quad (4.46)$$

Предположим, что каждый бит имеет 10%-ную вероятность мутации. Это дает матрицу мутации, полученную в примере 4.7. После отбора и мутации мы выполняем скрещивание с вероятностью 90 %. Если скрещивание отобрано, то оно выполняется путем отбора случайной битовой позиции $b \in [1, q - 1]$, где q – это число бит в каждой особи. Затем мы объединяем биты $1 \rightarrow b$ от первого родителя с битами $(b + 1) \rightarrow q$ от второго родителя.

Воспользуемся размером популяции $N = 3$. Существует $(n + N - 1)$ по $N = 10$ по $3 = 120$ возможных популяционных распределений. Мы можем применить уравнение (4.43) для вычисления вероятности перехода между каждым из 120 популяционных распределений, в результате получив транзитную 120×120-матрицу P. Затем мы можем рассчитать вероятность каждого возможного популяционного распределения тремя различными способами:

1) мы можем применить результат уравнения Дэвиса-Принсипа (4.15);
2) из теоремы 4.2 мы можем численно возводить P до все возрастающих более высоких степеней до тех пор, пока она не сойдется, а затем применить любую из строк P^{∞} для наблюдения вероятности каждой возможной популяции;
3) мы можем рассчитать собственные данные P^T и найти собственный вектор, соответствующий единичному собственному значению.

Каждый из этих подходов дает нам одно и то же множество из 120 вероятностей для 120 популяционных распределений. Мы находим, что вероятность того, что популяция содержит все оптимальные особи, то есть каждая особь равна битовой строке 111, составляет 6.1 %. Вероятность того, что популяция не содержит оптимальных особей, составляет 51.1 %. На рис. 4.2 показаны результаты симуляции 20 000 поколений и показано, что результаты симуляции близко соответствуют марковским результатам. Результаты симуляции приближенные, будут варьироваться из одного прогона в другой и будут равны марковским результатам только по мере приближения числа поколений к бесконечности.

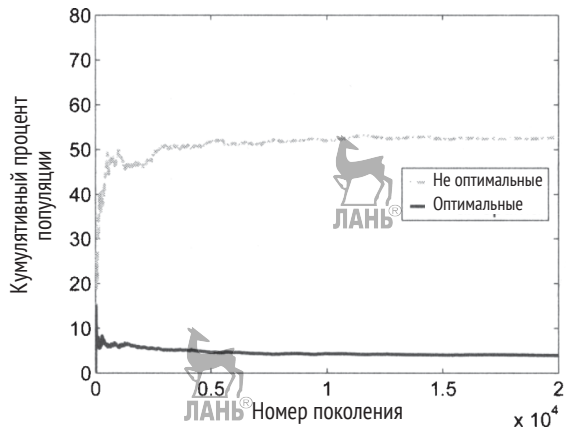


Рис. 4.2. Пример 4.9: результаты симулирования нахождения трехбитных подстрок задачи one-max. Теория Маркова предсказывает, что процент неоптимальных особей составляет 51.1 %, а процент всех оптимальных равняется 6.1 %

■

■ Пример 4.10

Здесь мы повторим пример 4.9, за исключением следующих ниже значений приспособленности:

$$f(000) = 5,\ \ f(100) = 2,\ \ f(001) = 2,\ \ f(101) = 3,$$
$$f(010) = 2,\ \ f(110) = 3,\ \ f(011) = 3,\ \ f(111) = 4. \qquad (4.47)$$

Эти значения приспособленности такие же, как и в уравнении (4.46), за исключением того, что мы сделали строку из 000 бит наиболее приспособленной особью. Такая задача называется обманной, потому что обычно, когда мы добавляем 1 бит к одной из вышеуказанных особей, ее приспособленность увеличивается. Исключение состоит в том, что

111 не является самой приспособленной особью. Самой приспособленной особью становится 000.

Как и в примере 4.9, мы вычисляем множество из 120 вероятностей для 120 популяционных распределений. Мы находим, что вероятность того, что популяция содержит все оптимальные особи, то есть каждая особь равна битной строке 000, равна 5.9 %. Это меньше, чем вероятность всех оптимальных особей в примере 4.9, которая составила 6.1 %. Вероятность того, что популяция не содержит оптимальных особей, составляет 65.2 %. Это больше, чем вероятность отсутствия оптимальных особей в примере 4.9, которая составила 51.1 %. Данный пример иллюстрирует, что обманчивые задачи решать сложнее, чем задачи с более регулярной структурой. На рис. 4.3 показаны результаты симуляции 20 000 поколений и показано, что результаты симуляции близко соответствуют марковским результатам.

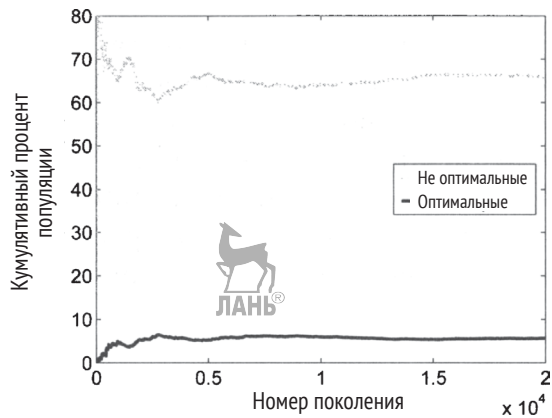


Рис. 4.3. Пример 4.10: результаты симуляции трехбитной обманчивой задачи. Теория Маркова предсказывает, что процент неоптимальных особей составляет 65.2 %, а процент всех оптимальных особей равняется 5.9 %

■

Проклятие размерности: проклятие размерности – это фраза, которая изначально использовалась в контексте динамического программирования [Bellman, 1961]. Тем не менее она еще более применима к марковским моделям для генетических алгоритмов. Размер транзитной матрицы марковской модели эволюционного алгоритма равняется $T \times T$, где $T = (N + n - 1)$ по N. Размерности транзитной матрицы для некоторых сочетаний размера популяции N и мощности поискового пространства n, равной 2^q для g-битных поисковых пространств, при-

ведены в табл. 4.1. Мы видим, что размерность транзитной матрицы становится абсурдно большой даже для задач скромной размерности. Это, по-видимому, указывает на то, что марковское моделирование представляет интерес только с теоретической точки зрения и не имеет практического применения. Однако есть несколько причин, по которым такой ответ может быть преждевременным.

Таблица 4.1. Размерности марковской транзитной матрицы для разных мощностей поискового пространства n и размеров популяции N. Взято из публикации [Reeves и Rowe, 2003, стр. 131]

число бит q	$n = 2^q$	N	T
10	2^{10}	10	10^{23}
10	2^{10}	20	10^{42}
20	2^{20}	20	10^{102}
50	2^{50}	50	10^{688}

Во-первых, несмотря на то что мы не можем применять марковские модели к задачам реалистичного размера, марковские модели все же дают точные вероятности для небольших задач. Это позволяет нам взглянуть на преимущества и недостатки различных эволюционных алгоритмов для небольших задач, исходя из допущения, что мы имеем марковские модели для эволюционных алгоритмов, помимо генетических алгоритмов. Это именно то, что мы делаем в публикации [Simon и соавт., 2011b], когда сравниваем генетические алгоритмы с биогеографической оптимизацией (biogeography-based optimization, BBO). Сегодня многие эволюционно-алгоритмические исследования сосредоточены на симуляциях. Проблема с симуляциями заключается в том, что их исходы сильно зависят от деталей реализации и от конкретного используемого генератора случайных чисел. Кроме того, если какое-то событие имеет очень малую вероятность возникновения, то потребуется много симуляций, чтобы обнаружить эту вероятность. Результаты симуляций полезны и необходимы, но их всегда следует воспринимать с долей скептицизма и не в буквальном смысле.

Во-вторых, размерность транзитных матриц Маркова может быть сокращена. Наши марковские модели включают T состояний, но многие из этих состояний очень похожи друг на друга. Например, рассмотрим генетический алгоритм с мощностью поискового пространства 10 и размером популяции 10. Таблица 4.1 показывает, что марковская модель имеет 10^{23} состояний, но к ним относятся состояния

$$v(1) = \{5, 5, 0, 0, 0, 0, 0, 0, 0, 0\}$$
$$v(2) = \{4, 6, 0, 0, 0, 0, 0, 0, 0, 0\}$$
$$v(3) = \{6, 4, 0, 0, 0, 0, 0, 0, 0, 0\} \quad (4.48)$$

Эти три состояния настолько похожи, что есть смысл сгруппировать их и рассматривать как единое состояние. Мы можем сделать это со многими другими состояниями и получить новую марковскую модель с сокращенным пространством состояний. Каждое состояние в модели с сокращенным порядком состоит из группы исходных состояний. Транзитная матрица тогда будет определять вероятность перехода из одной группы исходных состояний в другую группу исходных состояний. Эта идея была предложена в публикации [Spears и De Jong, 1997] и далее обсуждается в публикации [Reeves и Rowe, 2003]. Трудно представить, как группировать состояния, чтобы сократить $10^{23} \times 10^{23}$-матрицу до управляемого размера, но, по крайней мере, эта идея позволяет нам обрабатывать более крупные задачи, чем мы могли бы в противном случае.

4.5. Системно-динамические модели генетических алгоритмов

В этом разделе мы применим марковскую модель предыдущего раздела для получения системно-динамической модели генетического алгоритма. Марковская модель дает нам вероятность появления каждого популяционного распределения по мере приближения числа поколений к бесконечности. Системно-динамическая модель, которую мы здесь выведем, совершенно другая; она даст нам процент каждой особи в популяции в зависимости от времени, когда размер популяции приближается к бесконечности. Представление генетического алгоритма как динамической системы было первоначально дано в публикациях [Nix и Vose, 1992], [Vose, 1990], [Vose и Liepins, 1991], и этот взгляд далее разъясняется в публикациях [Reeves и Rowe, 2003], [Vose, 1999].

Напомним, что согласно уравнению (4.22) $v = [v_1 \ldots v_n]^T$ – это популяционный вектор, v_1 – число особей x_1 в популяции, а элементы v в сумме составляют число N, то есть размер популяции. Определим вектор пропорциональности как

$$p = v/N. \quad (4.49)$$

Из этого следует, что элементы p в сумме составляют 1.

4.5.1. *Отбор*

Для того чтобы найти системно-динамическую модель генетического алгоритма только с отбором (то есть без мутации или скрещивания), мы можем разделить числитель и знаменатель уравнения (4.36) на N и записать вероятность отбора особи x_i из популяции, описываемой популяционным вектором v следующим образом:

$$P_s(x_i|v) = \frac{p_i f_i}{\sum_{j=1}^{n} p_j f_j} = \frac{p_i f_i}{f^T p}, \tag{4.50}$$

где f – это вектор-столбец значений приспособленности. Запись уравнения (4.50) для $i \in [1, n]$ и объединение всех N уравнений дает

$$P_s(x|v) = \begin{bmatrix} P_s(x_1|v) \\ \cdots \\ P_s(x_n|v) \end{bmatrix} = \frac{\mathrm{diag}(f)p}{f^T p}, \tag{4.51}$$

где diag(f) – это диагональная $n \times n$-матрица, диагональные записи которой состоят из элементов f.

Закон больших чисел говорит нам, что среднее значение результатов, полученных из большого числа испытаний, должно быть близко к математическому ожиданию одного испытания [Grinstead и Snell, 1997]. Из этого следует, что по мере увеличения размера популяции доля отборов каждой особи x_i будет близка к $P_s(x_j|v)$. Но число отборов x_i просто равно v_i в следующем поколении. Следовательно, для больших размеров популяции уравнение (4.50) можно записать в следующем виде:

$$p_i(t) = \frac{p_i(t-1) f_i}{\sum_{j=1}^{n} p_j(t-1) f_j}, \tag{4.52}$$

где t – это номер поколения.

Теперь предположим, что

$$p_i(t) = \frac{p_i(0) f_i^t}{\sum_{j=1}^{n} p_j(0) f_j^t}. \tag{4.53}$$

Это совершенно верно для $t = 1$, как видно из уравнения (4.52). Если допустить, что уравнение (4.53) справедливо для $t - 1$, то числитель уравнения (4.52) можно записать в виде

$$f_i p_i(t-1) = f_i \frac{p_i(0) f_i^{t-1}}{\sum_{j=1}^{n} p_j(0) f_j^{t-1}} = \frac{p_i(0) f_i^{t}}{\sum_{j=1}^{n} p_j(0) f_j^{t-1}} \tag{4.54}$$

и знаменатель уравнения (4.52) можно записать в виде

$$\begin{aligned} \sum_{j=1}^{n} p_j(t-1) f_j &= \sum_{j=1}^{n} f_j \frac{p_j(0) f_j^{t-1}}{\sum_{k=1}^{n} p_j(0) f_k^{t-1}} \\ &= \sum_{j=1}^{n} \frac{p_i(0) f_i^{t}}{\sum_{k=1}^{n} f_k^{t-1} p_k(0)}. \end{aligned} \tag{4.55}$$

Подстановка уравнений (4.54) и (4.55) в уравнение (4.52) дает

$$p_i(t) = \frac{p_i(0) f_i^{t}}{\sum_{k=1}^{n} f_k^{t-1} p_k(0)}. \tag{4.56}$$

Это уравнение дает вектор пропорциональности как функцию времени, как функцию значений приспособленности и как функцию начального вектора пропорциональности, когда в генетическом алгоритме реализуется только отбор (без мутации или скрещивания).

■ Пример 4.11

Как и в примере 4.9, мы рассматриваем трехбитную задачу нахождения подстроки с максимально возможным числом единиц one-max со значениями приспособленности

$$\begin{aligned} &f(000) = 1, \quad f(100) = 2, \quad f(001) = 2, \quad f(101) = 3, \\ &f(010) = 2, \quad f(110) = 3, \quad f(011) = 3, \quad f(111) = 4. \end{aligned} \tag{4.57}$$

Предположим, что начальный вектор пропорциональности равен

$$p(0) = [\, 0.93 \quad 0.01 \quad 0.01 \quad 0.01 \quad 0.01 \quad 0.01 \quad 0.01 \quad 0.01 \,]^T. \tag{4.58}$$

93 % исходной популяции состоят из наименее приспособленных особей, и только 1 % популяции состоит из наиболее приспособленных особей. На рис. 4.4 показан график уравнения (4.56). Мы видим, что по мере эволюции генетико-алгоритмической популяции x_4, x_6 и x_7, которые являются вторыми лучшими особями, сначала получают бóльшую часть популяции, которая первоначально принадлежала p_1. Наименее приспособленная особь, x_1, быстро удаляется из популяции

в процессе отбора. p_2, p_3 и p_5 на рисунке не показаны. Прежде чем вся популяция сойдется к оптимальной особи x_8, пройдет не очень много поколений.

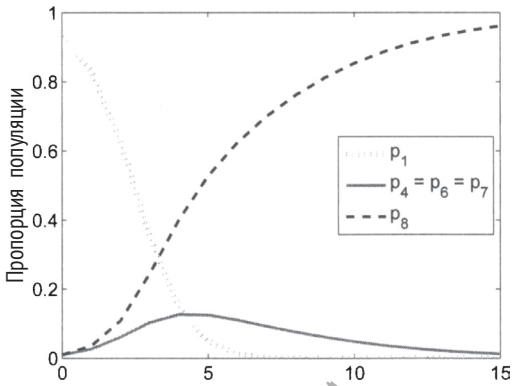


Рис. 4.4. Эволюция популяционного вектора пропорциональности для примера 4.11. Несмотря на то что лучшая особь, x_g, начинается только с 1 % популяции, он быстро сходится к 100 %. Наименее приспособленная особь, x_i, начинается с 93 % популяции, но быстро уменьшается до 0 %

■

Мы рассмотрели системно-динамическую модель для пропорционального приспособленности отбора, но и другие типы отбора, такие как турнирный отбор и ранговый отбор, тоже могут быть смоделированы как динамическая система [Reeves и Rowe, 2003], [Vose, 1999].

4.5.2. *Мутации*

Уравнение (4.51) наряду с законом больших чисел говорит нам, что

$$p(t) = \frac{\mathrm{diag}(f)\,p(t-1)}{f^T p(t-1)} \text{ (только отбор).} \tag{4.59}$$

Если после отбора следует мутация и M_{ji} – это вероятность того, что x_j мутирует в x_i, то мы можем использовать деривацию, подобную уравнению (4.38), и получить

$$p(t) = \frac{M^T \mathrm{diag}(f)\,p(t-1)}{f^T p(t-1)} \text{ (отбор и мутация).} \tag{4.60}$$

Если $p(t)$ достигает стационарного значения, то мы можем записать $p_{ss} = p(t-1) = p(t)$, для того чтобы записать уравнение (4.60) как

$$\begin{aligned} p_{ss} &= \frac{M^T \mathrm{diag}(f) p_{ss}}{f^T p_{ss}}, \\ M^T \mathrm{diag}(f) p_{ss} &= \left(f^T p_{ss}\right) p_{ss}. \end{aligned} \tag{4.61}$$

Это уравнение имеет вид $Ap = \lambda p$, где λ – это собственное значение A и p – собственный вектор A. Мы видим, что стационарный вектор пропорциональности генетического алгоритма с отбором и мутацией (то есть без скрещивания) является собственным вектором $M^T\mathrm{diag}(f)$.

■ Пример 4.12

Как и в примере 4.10, мы рассматриваем трехбитную обманчивую задачу со значениями приспособленности

$$\begin{aligned} &f(000) = 5, \quad f(100) = 2, \quad f(001) = 2, \quad f(101) = 3, \\ &f(010) = 2, \quad f(110) = 3, \quad f(011) = 3, \quad f(111) = 4. \end{aligned} \tag{4.62}$$

В данном примере мы используем скорость мутации 2 % на бит. Для этой задачи мы получаем

$M^T\mathrm{diag}(f) =$ (4.63)

$$\begin{bmatrix} 4.706 & 0.038 & 0.038 & 0.001 & 0.038 & 0.001 & 0.001 & 0.000 \\ 0.096 & 1.882 & 0.001 & 0.058 & 0.001 & 0.058 & 0.000 & 0.002 \\ 0.096 & 0.001 & 1.882 & 0.058 & 0.001 & 0.000 & 0.058 & 0.002 \\ 0.002 & 0.038 & 0.038 & 2.824 & 0.000 & 0.001 & 0.001 & 0.077 \\ 0.096 & 0.001 & 0.001 & 0.000 & 1.882 & 0.058 & 0.058 & 0.002 \\ 0.002 & 0.038 & 0.000 & 0.001 & 0.038 & 2.824 & 0.001 & 0.077 \\ 0.002 & 0.000 & 0.038 & 0.001 & 0.038 & 0.001 & 2.824 & 0.077 \\ 0.000 & 0.001 & 0.001 & 0.058 & 0.001 & 0.058 & 0.058 & 3.765 \end{bmatrix}.$$

Мы вычисляем собственные векторы $M^T\mathrm{diag}(f)$, как указано в уравнении (4.61), и шкалируем каждый собственный вектор так, чтобы его элементы в сумме составляли 1. Напомним, что собственные векторы матрицы инвариантны вплоть до значения шкалирования, то есть если p является собственным вектором, то cp также является собственным вектором для любой ненулевой константы c. Поскольку каждый собственный вектор представляет собой вектор пропорциональности, его элементы должны в сумме составлять 1, как указано в уравнении (4.49). Мы получаем восемь векторов, но только один из них полностью состоит из положительных элементов, и поэтому существует только один стационарный вектор пропорциональности:

$$p_{ss}(1) = [\,0.90074 \quad 0.03070 \quad 0.03070 \quad 0.00221 \quad 0.03070 \quad 0.00221 \quad 0.00221 \quad 0.0005\,]^T. \tag{4.64}$$

Из этого следует, что генетический алгоритм сходится к популяции, состоящей из 90.074 % особей x_i, из 3.07 % особей x_2, x_3, x_5 каждая и так далее. Более 90 % генетико-алгоритмической популяции будет состоять из оптимальных особей. Однако существует также собственный вектор $M^T\text{diag}(f)$, который содержит только один отрицательный элемент:

$$p_{ss}(2) = [\,-0.0008 \quad 0.0045 \quad 0.0045 \quad 0.0644 \quad 0.0045 \quad 0.0644 \quad 0.0644 \quad 0.7941\,]^T. \tag{4.65}$$

Он называется метастабильной точкой [Reeves и Rowe, 2003] и включает в себя высокий процент (79,41 %) особей x_8, которые являются вторыми наиболее приспособленными особями в популяции. Любой вектор пропорциональности, близкий к точке $p_{ss}(2)$, как правило, остается там, так как вектор $p_{ss}(2)$ является фиксированной точкой уравнения (4.61). Однако $p_{ss}(2)$ не является допустимым вектором пропорциональности, поскольку имеет отрицательный элемент, и поэтому даже если генетико-алгоритмическая популяция привлекается к точке $p_{ss}(2)$, в конечном итоге популяция будет от нее дрейфовать и сходиться к вектору $p_{ss}(1)$. На рис. 4.5 представлены результаты симуляции генетического алгоритма с отбором и мутацией. Мы использовали размер популяции $N = 500$ и начальный вектор пропорциональности

$$p(0) = [\,0.0 \quad 0.0 \quad 0.0 \quad 0.1 \quad 0.0 \quad 0.1 \quad 0.1 \quad 0.7\,]^T, \tag{4.66}$$

который близок к метастабильной точке $p_{ss}(2)$. По рис. 4.5 видно, что в течение примерно 30 поколений популяция остается близкой к своему исходному распределению, которое состоит из 70 % особей x_8 и которое близко к метастабильной точке $p_{ss}(2)$. Примерно через 30 поколений популяция быстро сходится к стабильной точке $p_{ss}(1)$, которая состоит примерно из 90 % особей x_1. Обратите внимание, что при повторном прогоне симуляции результаты будут отличаться из-за используемого для отбора и мутации генератора случайных чисел.

На рис. 4.6 показаны p_1 и p_8 из уравнения (4.60) для 100 поколений. В результате получается точная пропорция особей x_1 и x_8, по мере того как размер популяции приближается к бесконечности. Мы видим, что рис. 4.5 и 4.6 подобны, но рис. 4.5 является результатом симуляции конечного размера популяции и будет меняться при каждом выполнении симуляции из-за используемого генератора случайных чисел. С другой стороны, рис. 4.6 точен.

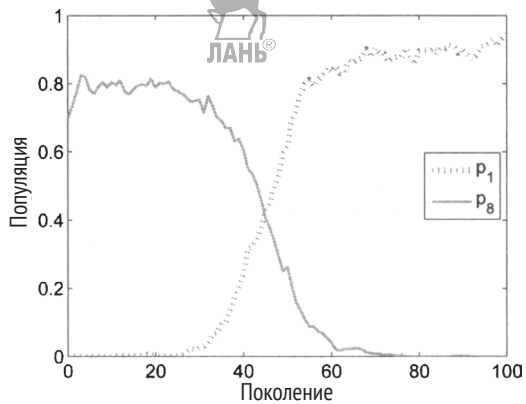


Рис. 4.5. Результаты симуляции для примера 4.12. Популяция парит вокруг метастабильной точки, которая состоит из 70 % особей x_i, прежде чем в конечном итоге она сойдется к стабильной точке из 90 % особей x_g. Результаты будут меняться из одной симуляции в другую из-за стохастического характера симуляции

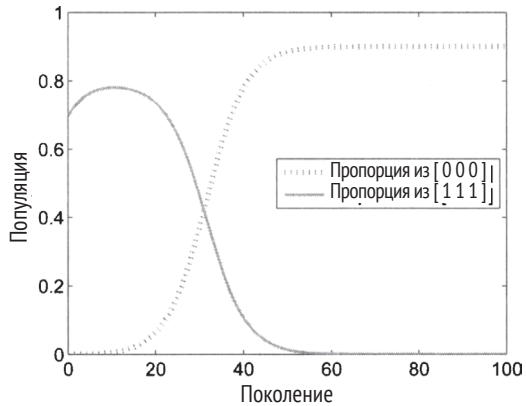


Рис. 4.6. Аналитические результаты для примера 4.12. Сравните с рис. 4.5. Аналитические результаты не зависят от генератора случайных чисел

■

4.5.3. *Скрещивание*

Как и в разделе 4.4.3, мы используем r_{jki} для обозначения вероятности того, что x_j и x_k скрещиваются, формируя x_i. Если популяция задана вектором пропорциональности p в бесконечной популяции, то вероятность того, что x_i, получается из случайного скрещивания, выводится следующим образом:

$$
\begin{aligned}
P_c(x_i \mid p) &= \sum_{j=1}^{n}\sum_{k=1}^{n} p_j p_k r_{jki} = \sum_{k=1}^{n} p_k \sum_{j=1}^{n} p_j r_{jki} \\
&= \sum_{k=1}^{n} p_k \begin{bmatrix} p_1 & \cdots & p_n \end{bmatrix} \begin{bmatrix} r_{1ki} \\ \vdots \\ r_{nki} \end{bmatrix} \\
&= \begin{bmatrix} p_1 & \cdots & p_n \end{bmatrix} \sum_{k=1}^{n} p_k \begin{bmatrix} r_{1ki} \\ \vdots \\ r_{nki} \end{bmatrix} \\
&= p^T \begin{bmatrix} \sum_{k=1}^{n} r_{1ki} p_k \\ \vdots \\ \sum_{k=1}^{n} r_{nki} p_k \end{bmatrix} \\
&= p^T \begin{bmatrix} \begin{bmatrix} r_{11i} & \cdots & r_{1ni} \end{bmatrix} p \\ \vdots \\ \begin{bmatrix} r_{n1i} & \cdots & r_{nni} \end{bmatrix} p \end{bmatrix} \\
&= p^T \begin{bmatrix} r_{11i} & \cdots & r_{1ni} \\ \vdots & \ddots & \vdots \\ r_{nni} & \cdots & r_{nni} \end{bmatrix} p \\
&= p^T R_i p, \qquad (4.67)
\end{aligned}
$$

где элемент в j-й строке и k-м столбце матрицы R_i обозначается как r_{jki} и является вероятностью того, что x_j и x_k скрещиваются, формируя x_i. Мы снова применим закон больших чисел [Grinstead и Snell, 1997], для того чтобы найти, что в пределе, по мере того как размер популяции N приближается к бесконечности, скрещивание изменяет долю особей x_i следующим образом:

$$\hat{p}_i = P_c(x_i \mid p) = p^T R_i p. \qquad (4.68)$$

Хотя матрица R_i часто несимметрична, квадратичную $P_c(x_i \mid p)$ всегда можно записать с использованием симметричной матрицы следующим образом:

$$P_c(x_i \mid p) = p^T R_i p = \tfrac{1}{2} p^T R_i p + \tfrac{1}{2}\left(p^T R_i p\right)^T, \qquad (4.69)$$

где вторая строка следует, поскольку $p^T R_i p$ является скаляром, а транспонирование скаляра равно скаляру. Поэтому, вспомнив, что $(ABC)^T = C^T B^T A^T$,

$$P_c(x_i \mid p) = \tfrac{1}{2} p^T R_i p + \tfrac{1}{2} p^T R_i^T p = \tfrac{1}{2} p^T (R_i + R_i^T) p = p^T \hat{R}_i p, \qquad (4.70)$$

где симметричная матрица $\hat{R}_i$ задается формулой

$$\hat{R}_i = \tfrac{1}{2}(R_i + R_i^T). \qquad (4.71)$$

■ Пример 4.13

Как и в примере 4.8, предположим, что у нас есть четырехэлементное поисковое пространство с особями $x = \{x_1, x_2, x_3, x_4\} = \{00, 01, 10, 11\}$. Мы реализуем скрещивание, случайно назначая $b = 1$ или $b = 2$ с равной вероятностью, а затем сцепляя биты $1 \rightarrow b$ от первого родителя с битами $(b + 1) \rightarrow 2$ от второго родителя. Возможности скрещивания могут быть записаны как

$$\begin{aligned}
&00 \times 00 \rightarrow 00\\
&00 \times 01 \rightarrow 01 \text{ или } 00\\
&00 \times 10 \rightarrow 00\\
&00 \times 11 \rightarrow 01 \text{ или } 00\\
&01 \times 00 \rightarrow 00 \text{ или } 01\\
&01 \times 01 \rightarrow 01\\
&01 \times 10 \rightarrow 00 \text{ или } 01\\
&01 \times 11 \rightarrow 01\\
&10 \times 00 \rightarrow 10\\
&10 \times 01 \rightarrow 11 \text{ или } 10\\
&10 \times 10 \rightarrow 10\\
&10 \times 11 \rightarrow 11 \text{ или } 10\\
&11 \times 00 \rightarrow 10 \text{ или } 11\\
&11 \times 01 \rightarrow 11\\
&11 \times 10 \rightarrow 10 \text{ или } 11\\
&11 \times 11 \rightarrow 11
\end{aligned} \qquad (4.72)$$

Это дает вероятности скрещивания r_{jki}, то есть вероятности того, что x_j и x_k скрестятся, чтобы дать $x_1 = 00$, следующим образом:

$$\begin{aligned}
&r_{111} = 1.0, \quad r_{121} = 0.5, \quad r_{131} = 1.0, \quad r_{141} = 0.5\\
&r_{211} = 0.5, \quad r_{221} = 0.0, \quad r_{231} = 0.5, \quad r_{241} = 0.0\\
&r_{311} = 0.0, \quad r_{321} = 0.0, \quad r_{331} = 0.0, \quad r_{341} = 0.0\\
&r_{411} = 0.0, \quad r_{421} = 0.0, \quad r_{431} = 0.0, \quad r_{441} = 0.0
\end{aligned} \qquad (4.73)$$

что приводит к матрице скрещиваний

$$R_1 = \begin{bmatrix} 1.0 & 0.5 & 1.0 & 0.5 \\ 0.5 & 0.0 & 0.5 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}. \tag{4.74}$$

R_1 явно несимметрична, но $P_c(x_i \mid p)$ по-прежнему может быть записана с использованием симметричной матрицы

$$P_c(x_1 \mid p) = p^T \hat{R}_1 p,$$

где $$\hat{R}_1 = \tfrac{1}{2}(R_1 + R_1^T) = \begin{bmatrix} 1.0 & 0.5 & 1.0 & 0.5 \\ 0.5 & 0.0 & 0.5 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}. \tag{4.75}$$

Другие матрицы $\hat{R}_i$ можно найти аналогичным образом.

■

Теперь предположим, что у нас есть генетический алгоритм с отбором, мутацией и скрещиванием в указанном порядке. У нас есть вектор пропорциональности p в поколении $t - 1$. Отбор и мутация модифицируют p, как показано в уравнении (4.60):

$$p(t) = \frac{M^T \operatorname{diag}(f)\, p(t-1)}{f^T p(t-1)}. \tag{4.76}$$

Скрещивание модифицирует p_i, как показано в уравнении (4.68). Однако p в правой части уравнения (4.68) уже был модифицирован операциями отбора и мутации, в результате приведя к вектору p в уравнении (4.76). Поэтому последовательность операций отбора, мутации и скрещивания приводит к $\hat{p}_i$, как показано в уравнении (4.68), но при этом p в правой стороне уравнения (4.68) заменяется на p, получающийся из операций отбора и мутации уравнения (4.76):

$$\begin{aligned} p_i(t) &= \left[\frac{M^T \operatorname{diag}(f)\, p(t-1)}{f^T p(t-1)}\right]^T R_i \left[\frac{M^T \operatorname{diag}(f)\, p(t-1)}{f^T p(t-1)}\right] \\ &= \frac{p^T(t-1) \operatorname{diag}(f) M R_i M^T \operatorname{diag}(f)\, p(t-1)}{\left(f^T p(t-1)\right)^2}. \end{aligned} \tag{4.77}$$

В уравнении (4.77) вместо R_i можно подставить $\hat{R}_i$ и получить эквивалентное выражение. Уравнение (4.77) дает точное аналитическое выражение для динамики пропорции особей x_i в бесконечной популяции.

Мы видим, что нам нужно вычислить системно-динамическую модель уравнения (4.77) для $i \in [1, n]$ в каждом поколении, где n – это размер поискового пространства. Матрицы в уравнении (4.77) имеют размер $n \times n$, а вычислительные усилия на матричное умножение пропорциональны n^3 в случае его реализации стандартными алгоритмами. Следовательно, системно-динамическая модель требует вычисления на порядке n^4. Это намного меньше вычислительных усилий, чем те, которые требует марковская модель, но они по-прежнему растут очень быстро по мере увеличения размера поискового пространства n, и они по-прежнему требуют недостижимых вычислительных ресурсов даже для задач среднего размера.

■ Пример 4.14

Еще раз рассмотрим трехбитную задачу нахождения подстроки с максимально возможным числом единиц one-max (см. пример 4.9), в которой приспособленность каждой особи пропорциональна числу единиц. Мы используем вероятность скрещивания 90 %, вероятность мутации 1 % на бит, размер популяции 1000 и исходный популяционный вектор пропорциональности

$$p(0) = [\, 0.8 \quad 0.1 \quad 0.1 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0 \,]^T. \tag{4.78}$$

На рис. 4.7 показан процент оптимальных особей в популяции в результате одного симулирования, а также точные теоретические результаты уравнения (4.77). Результаты симулирования хорошо соответствуют теории, но они являются приближенными и будут варьироваться от одного прогона к другому, в то время как теория точна.

Теперь предположим, что мы изменили исходный популяционный вектор пропорциональности на

$$p(0) = [\, 0.0 \quad 0.1 \quad 0.1 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.8 \,]^T. \tag{4.79}$$

На рис. 4.8 показан процент *наименее подогнанных* особей в результате одного симулирования, а также точные теоретические результаты. Поскольку вероятность получения наименее подогнанной особи настолько низка, результаты симулирования показывают пару всплесков на графике из-за случайных мутаций. Пики выглядят большими, учитывая масштаб графика, но на самом деле они довольно малы, достигая

всего 0.2 %. Однако теоретические результаты точны. Они показывают, что доля наименее подогнанных особей сначала увеличивается в течение нескольких поколений из-за мутации, а затем быстро уменьшается до стационарного значения ровно 0.00502 %. Для того чтобы прийти к такому же выводу, потребуется очень и очень много симуляций. И даже после тысяч симуляций можно прийти к неправильному выводу в зависимости от целостности используемого генератора случайных чисел.

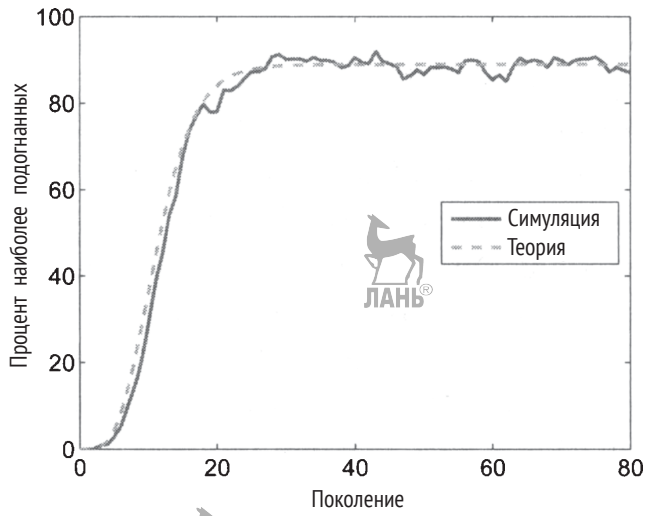


Рис. 4.7. Доля наиболее подогнанных особей для примера 4.14

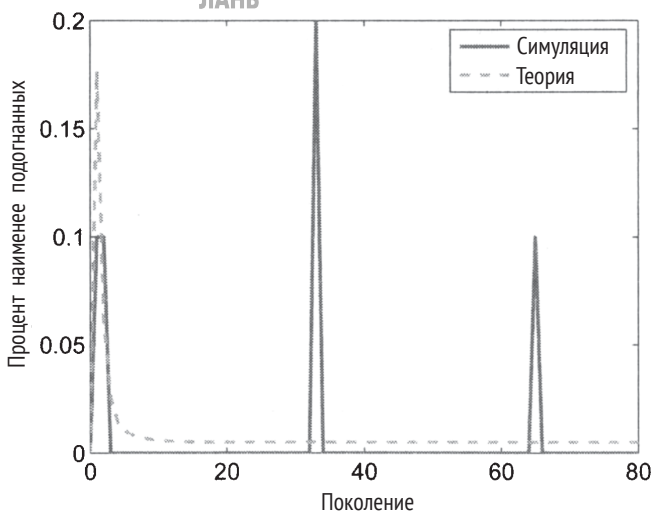


Рис. 4.8. Доля наименее подогнанных особей для примера 4.14

4.6. Заключение

В этой главе мы дали краткое изложение марковских моделей и системно-динамических моделей для генетических алгоритмов. Эти модели, которые были впервые разработаны в 1990-х годах, дают теоретически точные результаты, в то время как симуляция меняется из одного прогона в другой из-за генератора случайных чисел, который используется для операций отбора, скрещивания и мутации. Размер марковской модели возрастает факториально вместе с увеличением размера популяции и мощности поискового пространства. Системно-динамическая модель увеличивается с порядком n^4, где n – это мощность поискового пространства. Эти вычислительные потребности ограничивают применение марковских моделей и системно-динамических моделей очень небольшими задачами. Тем не менее эти модели по-прежнему полезны для сравнения разных реализаций генетических алгоритмов и для сравнения разных эволюционных алгоритмов, как мы видим в публикации [Simon и соавт., 2011b]. Некоторые дополнительные идеи и разработки по этим направлениям можно найти в публикациях [Reeves и Rowe, 2003], [Boce, 1999].

Марковское и системно-динамическое моделирования являются очень зрелыми областями, где был получен ряд общих результатов. Существует большой потенциал для дополнительного применения этих моделей к генетическим и другим эволюционным алгоритмам.

Для моделирования и анализа поведения генетических алгоритмов могут также использоваться и другие методы. Например, область статистической механики предусматривает усреднение многих молекулярных частиц для моделирования поведения группы молекул, и мы можем использовать эту идею для моделирования поведения генетического алгоритма с крупными популяциями [Reeves и Rowe, 2003, глава 7]; для анализа поведения генетического алгоритма мы также можем применять преобразования Фурье и Уолша [Vose и Wright, 1998a], [Vose и Wright, 1998b], наконец, для математического моделирования генетических алгоритмов мы можем использовать теорему отбора и ковариации Прайса [Poli и соавт., 2008, глава 3].

Идеи, представленные в этой главе, могут быть применены ко многим другим эволюционным алгоритмам, помимо генетических алгоритмов. Мы делаем это в публикациях [Simon и соавт., 2011a] и [Simon, 2011a] для биогеографической оптимизации, тогда как другие исследователи применяют эти идеи к другим эволюционным алгоритмам, но по-прежнему остается еще много возможностей для применения

марковских моделей и системно-динамических моделей к эволюционным алгоритмам. Это позволит проводить сопоставления и сравнения между различными эволюционными алгоритмами на аналитическом уровне, по крайней мере для небольших задач, а не полагаться на симуляцию. В нашем эволюционно-алгоритмическом исследовании симуляция необходима, но ее следует использовать для поддержки теории.

Задачи

Письменные упражнения

4.1. Сколько существует схем длины 2? Сколько из них порядка 0, сколько порядка 1 и сколько порядка 2?

4.2. Сколько существует схем длины 3? Сколько из них порядка 0, сколько порядка 1, сколько порядка 2 и сколько порядка 3?

4.3. Сколько существует схем длины l?

a) сколько из них порядка 0?
b) сколько из них порядка 1?
c) сколько из них порядка 2?
d) сколько из них порядка 3?
e) сколько из них порядка p?

4.4. Предположим, что экземпляры схемы h имеют значения приспособленности, на 25 % превышающие среднюю приспособленность генетико-алгоритмической популяции, и что вероятность разрушения h при мутации и скрещивании отрицательна. Предположим, что генетический алгоритм инициализируется одним экземпляром h. Определите номер поколения, когда h обгонит популяцию для размеров популяций 20, 50, 100 и 200 [Goldberg, 1989a].

4.5. Предположим, что у нас есть генетический алгоритм с 2-битными особями, такими, что вероятность мутирования из любой битовой строки x_i в любую другую битовую строку x_j равна p_m для всех $j \neq i$. Какова будет мутационная матрица? Убедитесь, что каждая строка в сумме составляет 1.

4.6. Предположим, что у нас есть генетический алгоритм с 2-битными особями, такими, что вероятность мутирования 0-го бита равна p_0, а вероятность мутирования 1-го бита равна p_1. Какова будет мутационная матрица? Убедитесь, что каждая строка в сумме составляет 1.

4.7. Вычислите t_{2ij} в примере 4.8 для $i \in [1, 4]$ и $j \in [1, 4]$.

4.8. Найдите R_2 и $\hat{R}_2$ для примера 4.13.

4.9. Предположим, что популяция в f-м поколении полностью состоит из оптимальных особей. Предположим, что мутация реализована так, что вероятность мутирования оптимальной особи в другую особь равна 0. Используйте уравнение (4.77), для того чтобы показать, что популяция в поколении $(t + 1)$ состоит полностью из оптимальных особей.

Компьютерные упражнения

4.10. Рассмотрим генетический алгоритм, в котором каждая особь состоит из одного бита. Пусть m_1 обозначает число экземпляров схемы $h_1 = 1$, и пусть f_1 обозначает ее приспособленность. Пусть m_0 обозначает число экземпляров схемы $h_0 = 0$, и пусть f_0 обозначает ее приспособленность.

Предположим, что генетический алгоритм имеет бесконечно большую популяцию и использует размножение и мутацию (без скрещивания). Выведите рекурсивное уравнение для $p(t)$, которое является отношением m_1/m_0 в t-м поколении [Goldberg, 1989a].

a) Предположим, что генетический алгоритм инициализирован $p(0) = 1$. Постройте график $p(t)$ для первых 100 поколений, когда скорость мутации составляет 10 % для коэффициентов приспособленности $f_1/f_0 = 10$, 2 и 1.1.

b) Повторите для скорости мутации 1 %.

c) Повторите для скорости мутации 0.1 %.

d) Дайте интуитивное объяснение ваших результатов.

4.11. Проверьте свойство 6 теоремы 4.2 для транзитной матрицы примера 4.1.

4.12. Некий профессор регулярно проводит сложные экзамены. 70 % студентов, которые в настоящее время имеют балл A на курсе, будут снижать его до B или хуже после каждого экзамена. 20 % студентов, которые в настоящее время имеют балл B или хуже, будут повышать свою оценку до A после каждого экзамена. С учетом бесконечного числа экзаменов сколько студентов заработают балл A по данному курсу?

4.13. Примените уравнения (4.26) и (4.28) для вычисления числа возможных популяций в генетическом алгоритме с размером популяции 10, в которых каждая особь состоит из 6 бит.

4.14. Повторите пример 4.10 со следующими ниже значениями приспособленности:

$$f(000) = 7,\ \ f(100) = 2,\ \ f(001) = 2,\ \ f(101) = 4,$$
$$f(010) = 2,\ \ f(110) = 4,\ \ f(011) = 4,\ \ f(111) = 6.$$

Что вы получаете в качестве вероятности отсутствия оптимальных решений? Как эта вероятность соотносится с полученной в примере 4.10? Как вы объясните разницу?

4.15. Повторите пример 4.10 со скоростью мутации 1%. Что вы получаете в качестве вероятности отсутствия оптимальных решений? Как эта вероятность соотносится с полученной в примере 4.10? Как вы объясните разницу?

4.16. Повторите пример 4.9 с тем изменением, что если оптимальное решение получено, то оно никогда не мутирует. Как это меняет мутационную матрицу? Что вы получаете в качестве вероятности всех оптимальных решений? Как вы объясните свои результаты?

Глава 5

Эволюционное программирование

Успех в предсказании окружающей среды является необходимым условием интеллектуального поведения.

– Лоуренс Фогель [Fogel, 1999, стр. 3]

Эволюционное программирование (evolutionary programming) было изобретено Лоуренсом Фогелем (Lawrence Fogel) вместе со своими коллегами Элом Оуэнсом и Джеком Уолшем в 1960-х годах [Fogel и соавт., 1966], [Fogel, 1999]. Эволюционное программирование эволюционно формирует популяцию особей, но не предусматривает рекомбинацию. Новые особи создаются исключительно путем мутации.

Эволюционное программирование было первоначально изобретено для эволюционирования конечных автоматов. Конечный автомат (finite state machine, FSM, машина с конечным числом состояний) – это виртуальная машина, которая генерирует последовательность выходных данных из последовательности входных данных. Генерация выходной последовательности определяется не только входами, но и множеством состояний и правил перехода из состояния в состояние. Лоуренс Фогель рассматривал предсказание ключевым компонентом интеллекта. Поэтому он считал, что разработка конечных автоматов, которые могут предсказывать следующий вывод некого процесса, является ключевым шагом в направлении развития вычислительного разума.

Краткий обзор главы

Раздел 5.1 дает обзор эволюционного программирования для задач с непрерывными областями. Несмотря на то что эволюционное програм-

мирование изначально описывалось как работающее на дискретной области, сегодня оно часто (можно сказать, обычно) реализуется на непрерывных областях, и именно так мы описываем ее в общих чертах. В разделе 5.2 дается описание конечных автоматов и показывается, как эволюционные программы могут их оптимизировать. Конечные автоматы интересны потому, что их можно использовать для моделирования самых разнообразных видов систем, включая компьютерные программы, цифровую электронику, системы управления и классификаторные системы.

В разделе 5.3 обсуждается оригинальный метод эволюционного программирования Фогеля для задач с дискретными областями. В разделе 5.4 разбирается дилемма заключенного, представляющая собой классическую задачу из теории игр. Решения дилеммы заключенного могут быть представлены как конечные автоматы, и поэтому эволюционные программы могут найти оптимальные решения дилеммы заключенного. В разделе 5.5 обсуждается 2-exchange, в которой используется эволюционное программирование для эволюционирования конечного автомата, с помощью которого муравей может перемещаться по решетке для эффективного поиска пищи.

5.1. Непрерывное эволюционное программирование

Предположим, мы хотим минимизировать $f(x)$, где x – это n-мерный вектор. Предположим, что $f(x) \geq 0$ для всех x. Эволюционная программа начинается со случайно сгенерированной популяции особей $\{x_i\}$, $i \in [0, N]$. Мы создаем детей $\{x'_i\}$ следующим образом:

$$x'_i = x_i + r_i\sqrt{\beta f(x_i) + \gamma}, \quad i \in [1, N], \tag{5.1}$$

где r_i – это случайный n-элементный вектор, каждый из элементов которого взят из гауссова распределения с нулевым средним значением и единичной дисперсией, а β и γ – это регулировочные параметры эволюционной программы. Дисперсия мутации x_i равна $(\beta f(x_i) + \gamma)$. Если $\beta = 0$, то все особи имеют одинаковую среднюю величину мутации. Если $\beta > 0$, то особь с низкой стоимостью не мутирует столько же, сколько мутирует особь с высокой стоимостью. Часто $\beta = 1$ и $\gamma = 0$ используются по умолчанию, как стандартные значения параметров эволюционной программы. В главе 6 мы увидим, что эволюционная программа с раз-

мером популяции, равным единице, эквивалентна двучленной эволюционной стратегии (ЭС).

Рассмотрение уравнения (5.1) выявляет некоторые проблемы реализации, связанные с эволюционным программированием [Bäck, 1996, раздел 2.2].

- Во-первых, стоимостные значения $f(x)$ должны быть сдвинуты так, чтобы они всегда были неотрицательными. Это не сложно на практике, но тем не менее это то, что приходится делать.
- Во-вторых, необходимо настроить β и γ. Значения по умолчанию $\beta = 1$ и $\gamma = 0$, но нет оснований полагать, что эти значения будут эффективными. Например, предположим, что значения x_i имеют большую область, и мы используем значения по умолчанию $\beta = 1$ и $\gamma = 0$. Тогда мутация в уравнении (5.1) будет очень мала по отношению к значению x_i, что приведет к очень медленному схождению или вообще к отсутствию схождения. И наоборот, если значения x_i имеют очень малую область, то принятые по умолчанию значения β и γ приведут к неоправданно большим мутациям; то есть мутации приведут к значениям x_i, которые находятся вне области.
- В-третьих, если $\beta > 0$ (типичный случай) и все стоимостные значения высокие, то $\beta f(x_i) + \gamma$) будет приблизительно постоянным для всех x_i, что приведет к приблизительно постоянной ожидаемой мутации для всех особей, независимо от их стоимости. Даже если особь повышает свою стоимость за счет полезной мутации, это улучшение, вероятно, будет обращено вспять пагубной мутацией.

Например, предположим, что значения стоимости находятся в диапазоне от минимума $f(x_1) = 1000$ до максимума $f(x_N) = 1100$. Особь x_1 относительно намного лучше, чем x_N, но значения стоимости шкалируются таким образом, что и x_1, и x_N мутируют примерно на ту же величину. Однако эта задача не является характерной исключительно для эволюционного программирования. Этот вопрос шкалирования значения стоимостной функции относится и к другим эволюционным алгоритмам, и мы обсудим его далее в разделе 8.7.

После того как уравнение (5.1) порождает N детей, мы имеем $2N$ особей: $\{x_i\}$ и $\{x_i'\}$. Из этих $2N$ особей мы отбираем лучших N и формируем популяцию в следующем поколении. Базовый алгоритм эволюционной программы кратко сформулирован на рис. 5.1.

Отобрать неотрицательные параметры эволюционной программы β и γ.
Номинально $\beta = 1$ и $\gamma = 0$.
$\{x_i\} \leftarrow$ {случайно сгенерированная популяция}, $i \in [1, N]$
While not (критерий останова)
Рассчитать стоимость $f(x_i)$ для каждой особи в популяции
For each особь x_i, $i \in [1, N]$
Сгенерировать случайный вектор r_i, в котором каждый элемент ~ $N(0,1)$
$x'_i = x_i + r_i\sqrt{\beta f(x_i) + \gamma}$
Next особь
$\{x_i\} \leftarrow$ лучшие N особей из $\{x_i, x'_i\}$
Next поколение

Рис. 5.1. Приведенный выше псевдокод дает схематичное описание стандартной эволюционной программы для минимизации $f(x)$

В эволюционной программе могут использоваться разные варианты отбора особей из $\{x_i, x'_i\}$ для следующего поколения. Рисунок 5.1 показывает, что это делается детерминированно; лучшие N особей отбираются из $\{x_i, x'_i\}$. Вместе с тем отбор также может быть сделан вероятностно. Например, для отбора N особей из $\{x_i, x'_i\}$ можно N раз применить рулеточный либо турнирный отбор, или различные другие методы отбора (см. раздел 8.7).

Эволюционная программа часто пишется так, что эволюционируют не только кандидатные решения; эволюционируют и их мутационные дисперсии. Такая программа нередко называется эволюционной метапрограммой; она кратко сформулирована на рис. 5.2. В эволюционной метапрограмме каждая особь x_i связана с мутационной дисперсией v_i. Мутационные дисперсии сами мутируют в поисках оптимальной мутационной дисперсии. Мы ограничиваем мутационные дисперсии на рис. 5.2 минимумом ϵ, который представляет собой определяемый пользователем регулировочный параметр. Эволюционная метапрограмма может ускорить схождение, автоматически адаптируя мутационные дисперсии, но она также может замедлить схождение в зависимости от конкретной задачи.

Отобрать неотрицательные параметры эволюционной программы ε и c.
Номинально $\varepsilon \ll 1$ и $c = 1$.
$\{x_i\} \leftarrow$ {случайно сгенерированная популяция}, $i \in [1, N]$

$\{v_i\} \leftarrow$ {случайно сгенерированные дисперсии}, $i \in [1, N]$
While not (критерий останова)
  Рассчитать стоимость $f(x_i)$ для каждой особи в популяции
  For each особь x_i, $i \in [1, N]$
    Сгенерировать случайные векторы r_{xi} и r_{vi}, где каждый элемент ~ $N(0,1)$
    $x'_i \leftarrow x_i + r_{xi}\sqrt{v_i}$
    $v'_i \leftarrow x_i + r_{vi}\sqrt{x_i}$
    $v'_i \leftarrow \max(v'_i, \varepsilon)$
  Next особь
  $\{x_i\} \leftarrow$ лучшие N особей из $\{x_i, x'_i\}$
  $\{v_i\} \leftarrow$ дисперсии, которые соответствуют $\{x_i\}$
Next поколение

Рис. 5.2. Приведенный выше псевдокод дает схематичное описание эволюционной метапрограммы для минимизации $f(x)$. Обратите внимание, что v_i связана с особью x_i для всех $i \in [1, N]$

■ Пример 5.1

В данном примере мы используем эволюционную программу для оптимизации эталонных функций Гривенка и Экли (см. определения этих эталонов в приложении С). В каждом эталоне мы используем 20 размерностей. Мы выполняем стандартную эволюционную программу на рис. 5.1 $\beta = (x_{\max} - x_{\min}) / 10$ и $\gamma = 0$. Используем популяцию размером 50 и нормализуем стоимость каждой особи так, что $f(x_i) \in [1, 2]$ в каждом поколении.

Мы также используем эволюционную метапрограмму на рис. 5.2 с $c = 1$ и $\epsilon = \beta / 10$. На рис. 5.3 и 5.4 показана минимальная стоимость популяции в зависимости от номера поколения, усредненно по 20 симуляциям Монте-Карло. Мы видим, что эволюционная метапрограмма сходится лучше, чем стандартная эволюционная программа на функции Гривенка, но гораздо хуже, чем стандартная эволюционная программа на функции Экли. Это, несомненно, обусловлено разными областями этих двух функций. В функции Гривенка область каждой независимой переменной равняется ±600, но только ±30 в функции Экли. Сравнение чисел на вертикальных осях двух рисунков показывает, что диапазоны двух функций также очень различны.

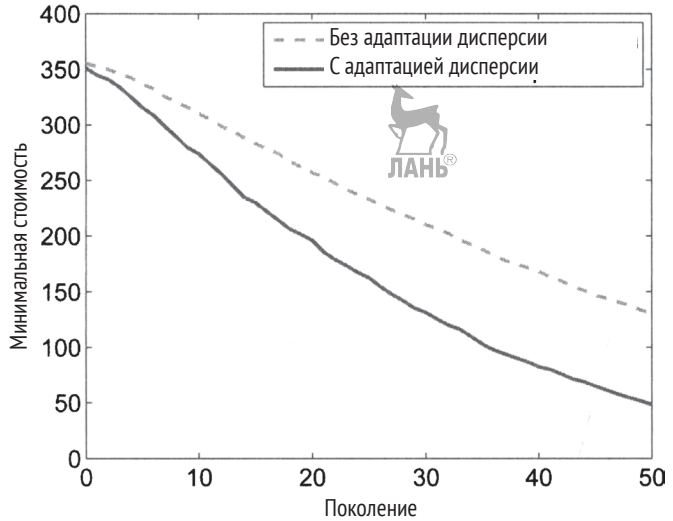


Рис. 5.3. Пример 5.1: схождение эволюционной программы на 20-мерной функции Гривенка, усредненно по 20 симуляциям Монте-Карло. Эволюционная метапрограмма сходится гораздо быстрее, чем стандартная эволюционная программа

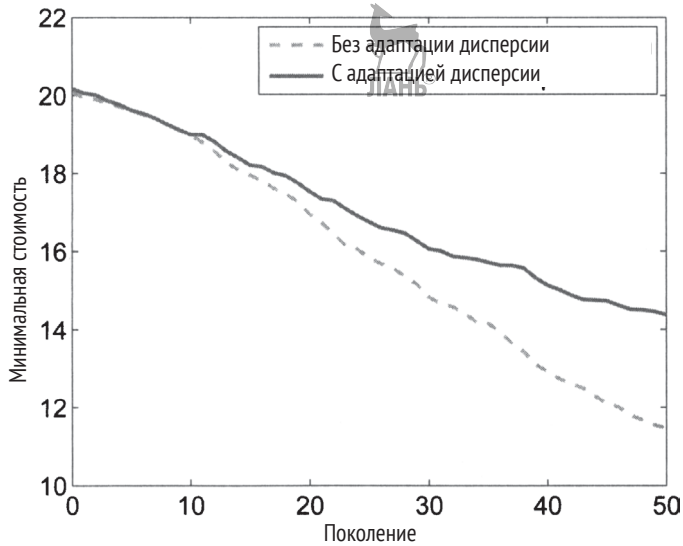


Рис. 5.4. Пример 5.1: схождение эволюционной программы на 20-мерной функции Экли, усредненно по 20 симуляциям Монте-Карло. Стандартная эволюционная программа сходится гораздо быстрее, чем эволюционная метапрограмма

■

5.2. Конечно-автоматная оптимизация

Эволюционное программирование было первоначально изобретено для эволюционирования (то есть эволюционного формирования) конечных автоматов. Конечные автоматы генерирует последовательность выходов в зависимости от внутреннего состояния и последовательности входов. На рис. 5.5 показан пример конечного автомата. Он имеет четыре состояния: *A*, *B*, *C* и *D*; два возможных ввода, 0 и 1, которые показаны слева от каждой правосторонней косой черты на рисунке; и три возможных вывода, *a*, *b* и *c*, которые показаны справа от каждой правосторонней косой черты. Стрелка в правом верхнем углу рисунка показывает, что конечный автомат начинается в состоянии *C*. Стрелки показывают, как состояние переходит после определенного ввода. Метки на строках показывают комбинации ввода/вывода. Рисунок 5.5 также может быть представлен в табличной форме, как показано в табл. 5.1.

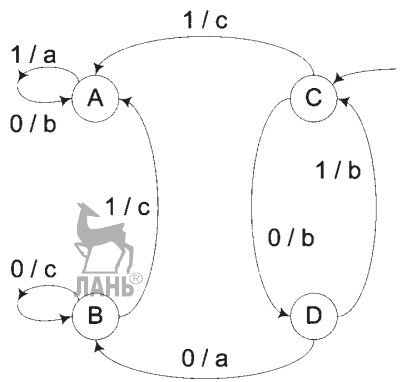


Рис. 5.5. Конечный автомат табл. 5.1 представлен в виде диаграммы. Этот конечный автомат имеет четыре состояния. Пара рядом с каждой стрелкой показывает ввод и соответствующий вывод, если конечный автомат находится в состоянии в конце стрелки. Стрелка в правом верхнем углу показывает, что конечный автомат начинается в состоянии C

Таблица 5.1. Конечный автомат рис. 5.5 в табличном виде

Текущее состояние	A	A	B	B	C	C	D	D
Ввод	0	1	0	1	0	1	0	1
Следующее состояние	A	A	B	A	D	A	B	C
Вывод	b	a	c	c	b	c	a	b

Предположим, что мы хотим создать конечный автомат, который реплицирует определенную выходную последовательность из некой входной последовательности. Например, мы, возможно, знаем, что входная последовательность

$$\text{Ввод} = \{ 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0 \} \tag{5.2}$$

должна привести к выходной последовательности

$$\text{Вывод} = \{0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1 \}. \tag{5.3}$$

Можно ли создать машину с переходами из состояния в состояние, которая будет производить желаемое поведение? Такая задача является оптимизационной: мы хотим эволюционно сформировать конечный автомат, который минимизирует различие между автоматным поведением и целевым поведением. Мы можем представить машину переходов в формате

$$S = \left[\left(\text{Вывод}_0 \quad \begin{matrix}\text{Следующее} \\ \text{состояние}_0\end{matrix}\right) \left(\text{Вывод}_1 \quad \begin{matrix}\text{Следующее} \\ \text{состояние}_1\end{matrix}\right) \ldots \right]^T. \tag{5.4}$$

Исходим из допущения, без потери общности, что конечный автомат начинается в состоянии 1. Элементы S расположены в виде:

$S(1)$ = вывод, если конечный автомат в состоянии 1 и ввод 0
$S(2)$ = следующее состояние, если конечный атомат в состоянии 1 и ввод 0
$S(3)$ = вывод, если конечный автомат в состоянии 1 и ввод 1
$S(4)$ = следующее состояние, если конечный атомат в состоянии 1 и ввод 1
$S(5)$ = вывод, если конечный автомат в состоянии 2 и ввод 0
⋮
$S(4n)$ = следующее состояние, если конечный атомат в состоянии n и ввод 1

(5.5)

где мы допустили, что ввод является двоичным. Мы можем легко расширить эту структуру на случай недвоичных входных данных. Мы видим, что S – это $4n$-элементный вектор-столбец, который описывает конечный автомат, где n – это число состояний. Мы можем применить вход уравнения (5.2) к конечному автомату и определить стоимость ошибки конечного автомата в виде

$$\text{Стоимость} = \sum_{i=1}^{12} \left| (\text{Желаемый вывод})_i - (\text{Вывод конечного автомата})_i \right|, \qquad (5.6)$$

где (Желаемый вывод)$_i$ – это i-й вывод уравнения (5.3), при этом имеется последовательность из 12 выводов. Затем мы можем применить алгоритм эволюционной программы на рис. 5.1 или 5.2 и эволюционно сформировать конечный автомат для минимизации уравнения (5.6).

Одна деталь реализации, которую мы должны рассмотреть, состоит в том, что на рис. 5.1 x_i является непрерывной величиной; но в эволюции конечного автомата элементы каждой особи являются целыми числами, которые ограничены конкретными областями. Уравнение (5.5) показывает, что для конечного автомата с двоичными выводами $S(i) \in [0, 1]$ для нечетных значений i и $S(i) \in [1, n]$ для четных значений i, где n – это число состояний. С этим можно справиться, просто выполнив мутации, показанные на рис. 5.1, затем ограничив элементы x'_i соответствующей областью и, наконец, округлив элементы x'_i до ближайших целых чисел. Опять же, возможны и иные подходы, оставленные для исследований и на изобретательность читателя.

Другие детали реализации включают регулировку β и γ и соответствующее шкалирование стоимостных значений $f(x_i)$ перед их использованием для получения мутационных дисперсий.

■ Пример 5.2

В данном примере мы применяем эволюционную программу для эволюционирования конечного автомата с целью минимизации функции стоимости уравнения (5.6), где входные и выходные данные приведены в уравнениях (5.2) и (5.3). В каждой особи эволюционной программы мы используем четыре состояния, $\beta = 1$, $\gamma = 0$, и размер популяции 5. Мы применяем алгоритм эволюционной программы на рис. 5.1 для эволюционирования популяции конечных автоматов. Шкалируем стоимость каждой особи так, что в каждом поколении стоимости $\in [1, 2]$. На рис. 5.6 показано схождение функции стоимости для одной симуляции эволюционной программы. Вектор S, который дает нулевую стоимость, был найден следующим образом:

$$S = [\,1\,3\,1\,2,\;\; 1\,1\,0\,4,\;\; 0\,1\,0\,2,\;\; 1\,4\,0\,2\,], \qquad (5.7)$$

что соответствует конечному автомату, показанному на рис. 5.7.

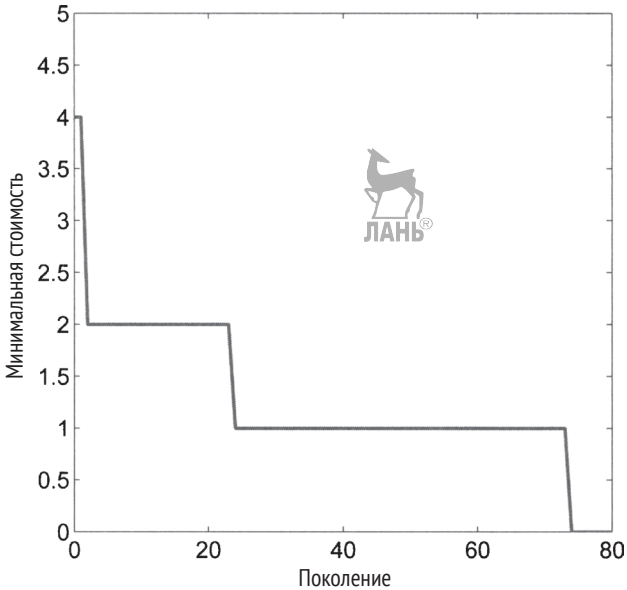


Рис. 5.6. Пример 5.2: схождение конечного автомата

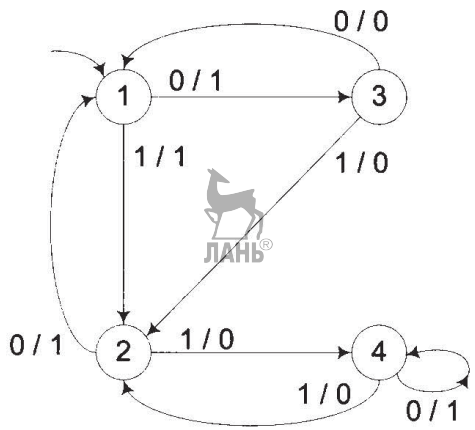


Рис. 5.7. Конечный автомат, эволюционно сформированный эволюционной программой примера 5.2. Если ввод в этот конечный автомат дается уравнением (5.2), то он создает вывод уравнения (5.3)

■

5.3. Дискретное эволюционное программирование

Оригинальная эволюционная программа Фогеля для генерации конечного автомата была реализована иначе, чем та, которую мы описали в предыдущем разделе [Fogel и соавт., 1966], [Fogel, 1999]. Его реализация была непосредственно применима к целочисленным областям. Его подход может быть использован не только для оптимизации конечных автоматов, но и для любой задачи, которая определена на дискретной области. На рис. 5.8 приведем краткое изложение его подхода.

$\{x_i\} \leftarrow$ {случайно сгенерированная популяция}, $i \in [1, N]$
While not (критерий останова)
    Рассчитать стоимость $f(x_i)$ каждой особи в популяции
    For each особь x_i, $i \in [1, N]$
        $x'_i \leftarrow$ случайная мутация x_i
    Next особь
    $\{x_i\} \leftarrow$ лучшие N особей из $\{x_i, x'_i\}$
Next поколение

Рис. 5.8. Приведенный выше псевдокод дает схематичное описание эволюционной программы Фогеля для задач дискретной оптимизации. Обратите внимание, что этот алгоритм является обобщением рис. 5.1

«Случайная мутация» на рис. 5.8 полностью зависит от конкретной задачи, которую мы пытаемся решить. Например, случайная мутация, которую Фогель использовал для оптимизации конечного автомата, была выбрана случайно как одна из следующих:

- добавить состояние со случайными парами ввода/вывода и ввода/перехода;
- удалить состояние. Любые переходы состояния в удаленное состояние перенаправляются в другое случайно отобранное состояние;
- случайно изменить пару ввод/вывод для случайно отобранного состояния;
- случайно изменить пару ввод/переход для случайно отобранного состояния;
- случайно изменить начальное состояние.

Фогель также предложил добавлять в стоимостную функцию штраф, пропорциональный сложности машины переходов. Это смещает выбор

лучших N особей в конце каждого поколения в сторону более простых машин переходов. Эта идея позволяет найти простые конечные автоматы для создания желаемого шаблона.

■ Пример 5.3

В данном примере мы пытаемся найти машину переходов из состояния в состояние, которая может генерировать простые числа. Мы используем 0 для обозначения непростых чисел и 1 для обозначения простых чисел. Ввод в конечный автомат на каждом временном шаге является индикатором простого числа (0 для false, 1 для true) предыдущего временного шага. В результате получаем входные и выходные последовательности

$$\begin{aligned} \text{Ввод} &= \{\, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, \ldots \} \\ \text{Желаемый вывод} &= \{\quad 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, \ldots \}. \end{aligned} \tag{5.8}$$

Входная последовательность соответствует тому, что 1 не является простым числом; 2 и 3 – простые; 4 – непростое; 5 – простое; 6 – непростое; 7 – простое; 8, 9, и 10 – непростые; 11 – простое; 12 – непростое и т. д. Выходная последовательность равна входной последовательности с задержкой на один временной шаг. Мы используем первые 100 положительных целых чисел для оценивания результативности конечного автомата, поэтому входные и выходные последовательности имеют длину 99 бит; входная последовательность соответствует целым числам 1–99, а желаемая выходная последовательность соответствует целым числам 2–100. Мы используем популяцию $N = 20$. Для каждой особи в каждом поколении эволюционной программы мы случайно отбираем одну из пяти мутаций, описанных ранее в этом разделе. Мы выполняем симуляцию эволюционной программы со стоимостным штрафом на число состояний конечного автомата, а также без штрафа. Рисунок 5.9 показывает лучший конечный автомат для каждого поколения, усредненно по 100 симуляциям Монте-Карло. Среднее число состояний составляет 4.7, если нет штрафа на число состояний, и 2.8, если есть стоимостной штраф $n/2$ на число состояний, где n – число состояний. Рисунок 5.9 показывает, что лучшая стоимость, которую мы можем достичь, составляет от 18 до 19, что не так впечатляет, учитывая, что в первых 100 положительных целых числах есть только 25 простых чисел. Машина переходов, которая всегда генерирует 0, будет стоить 25.

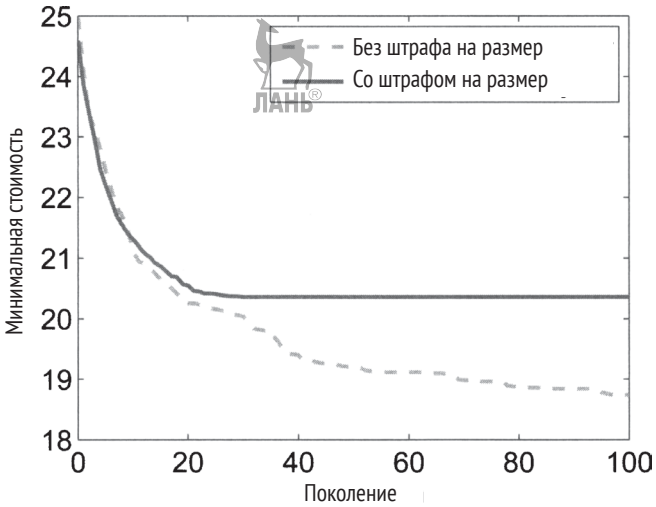


Рис. 5.9. Пример 5.3: схождение конечного автомата для предсказания простых чисел, усредненно по 100 симуляциям Монте-Карло

■

5.4. Дилемма заключенного

Дилемма заключенного – это классическая задача из теории игр. Предположим, что двое подозреваемых арестованы полицией. Полиция раздельно допрашивает подозреваемых. Полиция предоставляет каждому подозреваемому иммунитет от судебного преследования, если он предаст своего сообщника. Признание любого из подозреваемых даст полиции достаточно доказательств для заключения его сообщника в тюрьму на длительный срок. Однако если оба подозреваемых будут молчать, то у полиции не будет достаточно доказательств, для того чтобы заключить подозреваемых в тюрьму на очень длительный срок. В этом и состоит сложность для заключенных, которым запрещено общаться друг с другом. Если каждый подозреваемый хранит молчание (сотрудничают друг с другом), то оба подозреваемых получают условное наказание. Если каждый подозреваемый говорит (влияют друг на друга), то оба подозреваемых получают среднее наказание. Однако если один подозреваемый сотрудничает и один дезертирует, то подозреваемый, который дезертирует, выходит на свободу, в то время как подозреваемый, который сотрудничает, получает длительный приговор. Дилемма заключенного кратко сформулирована в табл. 5.2.

Таблица 5.2. Стоимостная матрица дилеммы заключенного

	Заключенный B Сотрудничает	**Заключенный B Дезертирует**
Заключенный A Сотрудничает	Заключенный A: 1 год Заключенный B: 1 год	Заключенный A: 10 лет Заключенный B: На свободе
Заключенный A Дезертирует	Заключенный A: На свободе Заключенный B: 10 лет	Заключенный A: 5 лет Заключенный B: 5 лет

Предположим, что вы – заключенный *A*. Если ваш сообщник сотрудничает, то вы можете выйти на свободу, если вы дезертируете. Если ваш сообщник дезертирует, то вы можете получить 5 лет вместо 10 лет, дезертировав. Следовательно, дело выглядит так, что независимо от того, что делает ваш сообщник, вы должны дезертировать. Тем не менее если оба заключенных применят эту стратегию, то оба заключенных дезертируют и получат 5 лет лишения свободы. Если оба заключенных будут сотрудничать друг с другом, то они оба получат только 1 год лишения свободы. Эгоистичные решения приводят к тому, что оба заключенных оказываются в худшем положении, чем если бы они действовали в интересах своего сообщника. Вот почему проблема называется дилеммой.

В повторяющейся дилемме заключенного игра в дилемму заключенного играется несколько раз, и цель каждого игрока состоит в том, чтобы максимизировать свою общую выгоду (то есть свести к минимуму свое общее время заключения в тюрьме) во всех играх [Axelrod, 2006]. Как вы играете в игру, делая выбор, сотрудничать или дезертировать, вы помните предыдущие решения вашего сообщника. Поэтому если ваш сообщник неоднократно дезертирует, то вы можете решить дезертировать, чтобы максимизировать свою выгоду. Если ваш сообщник сотрудничает, то вы можете решить сотрудничать для поддержания взаимного сотрудничества и взаимной выгоды. Слово «повторяющийся» из термина «повторяющаяся дилемма заключенного» часто опускается. Поэтому термин «дилемма заключенного» может относиться к одному раунду табл. 5.2 или нескольким раундам.

Для дилеммы заключенного было предложено несколько стратегий, каждая из которых может быть удобно представлена как конечный автомат [Ashlock, 2009], [Rubin-stein, 1986]. Одна стратегия заключается в постоянном сотрудничестве. Эта оптимистичная стратегия изображена с помощью конечного автомата с одним состоянием, показанного на рис. 5.10. Мы начинаем с сотрудничества на первом раунде и переходим в состояние 1. Если предыдущее решение соперника было C, то вы-

водим C и остаемся в состоянии 1. Если предыдущее решение соперника было D, то делаем то же самое – выводим C и остаемся в состоянии 1.

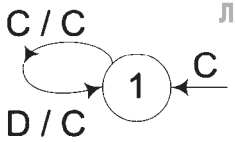


Рис. 5.10. Конечный автомат для стратегии «всегда сотрудничать» в дилемме заключенного

Другая стратегия, стратегия «око за око», следует философии «поступайте с другими так же, как они поступали с вами». Мы начинаем с сотрудничества на первом раунде и переходим в состояние 1. Мы сотрудничаем, если наш соперник сотрудничал на его предыдущем ходу, и дезертируем, если наш соперник дезертировал на его предыдущем ходу. Эта стратегия показана на рис. 5.11.

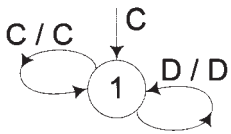


Рис. 5.11. Конечный автомат для стратегии «око за око» в дилемме заключенного

Еще одна стратегия «око за два ока» чуть более обнадеживающа и всепрощающа, чем око за око. Мы сотрудничаем, если соперник не дезертирует в течение двух ходов подряд. Эта стратегия показана на рис. 5.12.

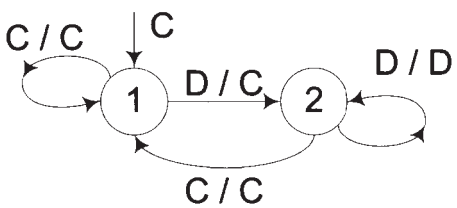


Рис. 5.12. Конечный автомата для стратегии «око за два ока» в дилемме заключенного

Мрачная стратегия весьма неумолима. Мы начинаем оптимистично, сотрудничая на первом ходу и переходя в состояние 1, и продолжаем

сотрудничать до тех пор, пока наш соперник сотрудничает. Однако если наш оппонент дезертирует, то мы больше никогда не будем сотрудничать. Эта стратегия показана на рис. 5.13.

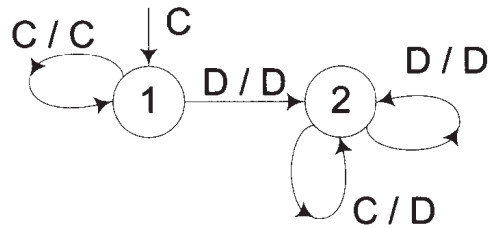


Рис. 5.13. Конечный автомат для мрачной стратегии в дилемме заключенного

В стратегии наказания мы отомстим нашему сопернику за дезертирство, но в конце концов простим. Если наш соперник дезертирует, то и мы дезертируем, и мы продолжаем дезертировать до тех пор, пока наш соперник не будет сотрудничать в течение трех ходов подряд. Мы снова сотрудничаем только после того, как соперник трижды подряд сотрудничает. Эта стратегия показана на рис. 5.14.

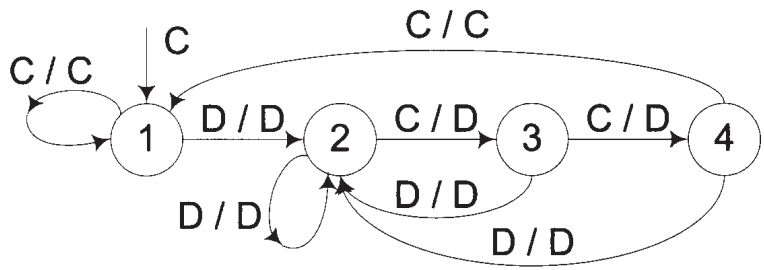


Рис. 5.14. Конечный автомат для стратегии наказания в дилемме заключенного

«Дилемма заключенного» достаточно много изучалась, потому что она имеет много приложений, включая обмен файлами между одноранговыми устройствами [Ellis и Yao, 2007], рекламные стратегии среди конкурирующих компаний [Corfman и Lehmann, 1994], политику [Grieco, 1988], жульничество в спорте и других сферах [Ehrnborg и Rosen, 2009] и многие другие [Poundstone, 1993].

■ Пример 5.4

В данном примере мы эволюционно формируем конечный автомат, для того чтобы минимизировать стоимость в игре дилеммы заключенного. Конечный автомат для дилеммы заключенного может быть представлен, как показано в уравнении (5.5), за исключением того, что мы добавим еще одно целое число в начале вектора, с тем чтобы указать первый ход. Мы используем 0 для обозначения сотрудничества и 1 для обозначения дезертирства. Мы создаем четырех случайных, но постоянных соперников, каждый со стратегией конечного автомата с четырьмя состояниями. Мы выполняем эволюционную программу с $\beta = 1$ и $\gamma = 0$. При этом мы случайно инициализируем популяцию эволюционной программы размером 5 особей, каждая из которых содержит четыре состояния. Каждая особь эволюционной программы играет 10 игр против каждого из четырех случайных, но постоянных соперников с целью оценивания ее результативности. В данном примере машинами переходов для четырех случайных, но постоянных соперников являются:

$$\begin{aligned} S_1 &= [\,0,\ 0\ 3\ 0\ 3,\ 0\ 3\ 1\ 2,\ 1\ 4\ 0\ 2,\ 1\ 3\ 1\ 3\,] \\ S_2 &= [\,0,\ 0\ 1\ 0\ 4,\ 1\ 2\ 0\ 2,\ 0\ 4\ 0\ 3,\ 0\ 2\ 0\ 4\,] \\ S_3 &= [\,0,\ 1\ 4\ 0\ 4,\ 0\ 1\ 0\ 1,\ 1\ 4\ 0\ 2,\ 0\ 1\ 1\ 4\,] \\ S_4 &= [\,1,\ 0\ 4\ 1\ 4,\ 0\ 4\ 0\ 1,\ 0\ 3\ 0\ 3,\ 0\ 3\ 0\ 2\,] \end{aligned} \tag{5.9}$$

Мы используем алгоритм эволюционной программы на рис. 5.1 для эволюционирования популяции конечных автоматов. На рис. 5.15 показано схождение стоимостной функции эволюционной программы для одной симуляции.

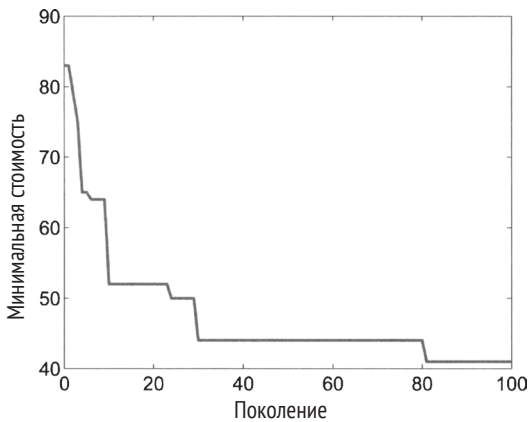


Рис. 5.15. Пример 5.4: схождение стоимости конечного автомата дилеммы заключенного

Вектором S, дающим минимальную стоимость, оказался вектор

$$S = [\ 1,\ 1\ 2\ 1\ 2,\ 0\ 1\ 1\ 3,\ 1\ 4\ 0\ 1,\ 1\ 1\ 0\ 1\ ], \tag{5.10}$$

который соответствует конечному автомату, показанному на рис. 5.16.

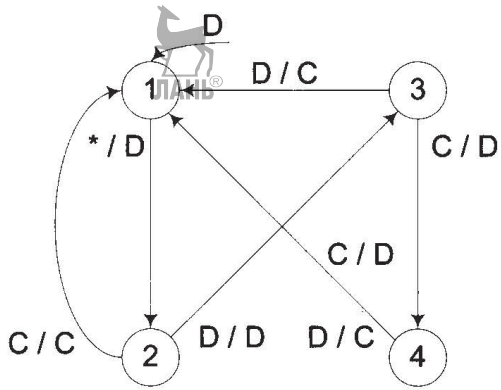


Рис. 5.16. Пример 5.4: лучший конечный автомат, эволюционно сформированный эволюционной программой. Звездочка, выходящая из состояния 1, означает либо C, либо D; то есть если конечный автомат находится в состоянии 1, то независимо от предыдущего хода соперника (C или D) выводом конечного автомата будет D, а следующим состоянием будет состояние 2

■

Мы также можем провести интересные эксперименты, в которых популяция эволюционной программы эволюционирует, играя против самой себя. То есть каждая особь в эволюционной программе играет против каждой другой особи в эволюционной программе с целью оценивания индивидуальных стоимостей.

Существует ряд вариантов дилеммы заключенного. Например, мы исходили из того, что каждая особь на каждом шагу может выбрать один из двух возможных ходов. Вместе с тем мы можем также допустить уровни сотрудничества, такие что каждый раунд может включать в себя континуум шагов наряду с континуумом связанных с ними стоимостей. 0 может представлять суммарное сотрудничество, 1 – суммарное дезертирство, а что-нибудь между 0 и 1 может представлять разную степень сотрудничества и дезертирства [Harrald и Fogel, 1996]. Еще один вариант – дать возможность каждому добровольно прекратить игру в любое желаемое время [Delahaye и Mathieu, 1995]. Другой вариант – позволить более чем двум игрокам играть одновременно. В этом случае

стоимость конкретного игрока часто является функцией от хода игрока и числа сотрудничающих соперников [Bonacich и соавт., 1976]. Можно ввести еще одно осложнение, если стоимостная матрица табл. 5.2 изменяется со временем [Worden и Levin, 2007].

5.5. Задача искусственного муравья

В этом разделе мы обсудим задачу искусственного муравья (не путать с оптимизацией на основе муравьиной кучи), еще одну известную задачу, которая может быть решена с помощью конечного автомата. Задача искусственного муравья была введена в 1990 году [Jefferson и соавт., 2003] и хорошо описана в публикации [Koza, 1992, раздел 3.3.2]. Искусственный муравей помещается в тороидальную решетку 32×32, в которой в 90 из 1024 квадратов имеется пища. Сенсорные возможности муравья чрезвычайно ограничены; он может только ощущать, есть ли пища в квадрате прямо перед ним. В каждом квадрате он может сделать один из трех ходов: он может двигаться на один квадрат вперед в направлении перед ним, и в этом случае он съест пищу в квадрате, если она там присутствует; либо он может повернуть направо или налево, оставаясь в своем текущем квадрате. Путь муравья называется тропой Санта-Фе, и она показана на рис. 5.17.

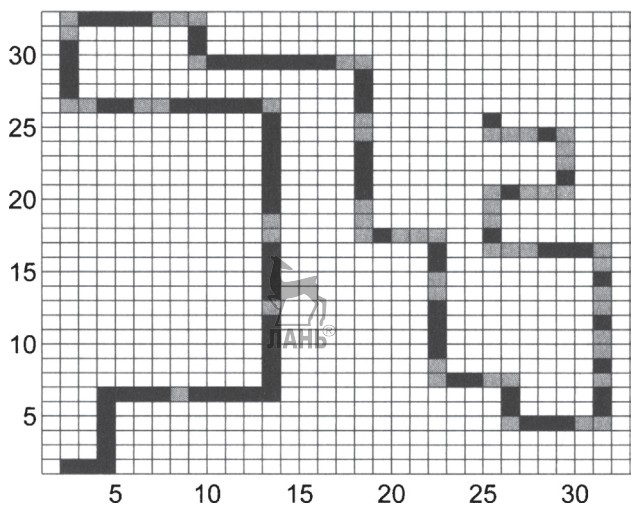


Рис. 5.17. Тропа Санта-Фе 32×32. Муравей находится в нижнем левом углу, глядя вправо. Белые квадраты пусты, а черные квадраты имеют пищу. Серые квадраты также пусты, но показаны серым цветом, для того чтобы лучше проиллюстрировать оптимальный путь муравья по решетке

Муравей начинает в квадрате (1,1), который является нижним левым углом решетки (хотя технически нет никаких «углов», так как решетка тороидальная), а начальная ориентация муравья имеет направление направо. Он ощущает пищу впереди себя на рис. 5.17, и поэтому он должен двигаться вперед к следующему квадрату в координате (2,1) и съесть эту пищу. Находясь в квадрате (2,1), он снова почувствует перед собой пищу, и поэтому должен двигаться вперед к квадрату (3,1) и съесть эту пищу. Находясь в квадрате (3,1), он почувствует пищу перед собой, и поэтому должен двигаться вперед к квадрату (4,1) и снова съесть эту пищу. Но теперь он сталкивается с загвоздкой в своем до этого момента предсказуемом и удовлетворительном путешествии. Находясь в квадрате (4,1), он почувствует, что перед ним нет еды. Должен ли он двигаться вперед, надеясь найти пищу в следующем квадрате? Или же он должен повернуть налево или направо, надеясь найти еду рядом с его текущим положением? Если он повернет налево, то почувствует еду в квадрате (4,2) и останется на оптимальной тропе. Но если он повернет направо, то почувствует еду в квадрате (4,32) (напомним, что решетка тороидальная), и это обеспечит ему некоторое краткосрочное удовлетворение, но в конечном итоге собьет его с тропы.

Задача искусственного муравья заключается в том, чтобы найти конечный автомат, который будет направлять муравья по тропе Санте-Фе, давая муравью съесть всю пищу за наименьшее число ходов. Шаг вперед, поворот направо и поворот налево считаются одним ходом. Оптимальный путь через решетку, который достигается перемещением по черным и серым квадратам на рис. 5.17, состоит из 167 ходов.

Мы можем применить эволюционную программу для эволюционного формирования решения задачи искусственного муравья. Сначала мы решаем, сколько состояний хотим использовать. Предположим, что мы решили использовать пять состояний. Затем мы кодируем конечный автомат со следующей последовательностью целых чисел:

$$\begin{aligned} &1_{0,m}, 1_{0,s}, 1_{1,m}, 1_{1,s}, \\ &2_{0,m}, 2_{0,s}, 2_{1,m}, 2_{1,s}, \\ &\quad\vdots \\ &5_{0,m}, 5_{0,s}, 5_{1,m}, 5_{1,s}. \end{aligned} \tag{5.11}$$

В приведенном выше представлении конечного атомата используются следующие обозначения:

- $n_{0,m}$ – это движение муравья, если он находится в состоянии n и не чувствует пищу непосредственно перед собой. Мы принимаем $n_{0,m}$ = 0, 1 или 2, соответственно обозначая ход вперед, поворот вправо или поворот влево;
- $n_{0,s}$ – это состояние, в которое муравей переходит, если он в данный момент находится в состоянии n и не чувствует пищу непосредственно перед собой;
- $n_{1,m}$ – это движение муравья, если он находится в состоянии n и чувствует пищу прямо перед собой;
- $n_{1,s}$ – это состояние, в которое муравей переходит, если он в данный момент находится в состоянии n и чувствует пищу непосредственно перед собой.

Таким образом, мы кодируем конечный автомат целыми числами $4N$, где N – число состояний. Мы исходим из того, что муравей всегда начинает в состоянии 1.

Мы можем вычислить каждый конечный автомат в популяции эволюционной программы и посмотреть, насколько хорошо он перемещается по решетке. Мы инициализируем муравья, помещая его в левый нижний угол по направлению вправо. Мы устанавливаем верхний предел на число ходов, равное 500; это число шагов, которые муравей может сделать с каждым конечным автоматом. Стоимость конечного автомата измеряется числом ходов, которое муравью потребуется, для того чтобы съесть всю пищу в решетке. Если после 500 ходов муравей не съел всю пищу, то стоимость равна 500 плюс число квадратов с пищей, которых муравей не достиг. На рис. 5.18 показан ход выполнения одной симуляции эволюционной программы для конечных автоматов с пятью состояниями и размером популяции 100.

Среднее число пищевых гранул, которые муравей съест, зависит от того, сколько состояний мы используем в конечных автоматах. Если мы используем слишком мало состояний, то у нас нет достаточной гибкости, для того чтобы найти хорошее решение. Если мы используем слишком много состояний, то результативность эволюционной программы улучшается, но это улучшение не окупит увеличение времени работы компьютера. Среднее количество пищи, съеденной каждым муравьем в популяции после 100 поколений в эволюционной программе с размером популяции 100, представлено следующим образом:

размерность конечного автомата = 4 : 50.1 пищевых гранул
размерность конечного автомата = 6 : 60.5 пищевых гранул
размерность конечного автомата = 8 : 62.5 пищевых гранул
размерность конечного автомата = 10 : 63.1 пищевых гранул
размерность конечного автомата = 12 : 63.8 пищевых гранул (5.12)

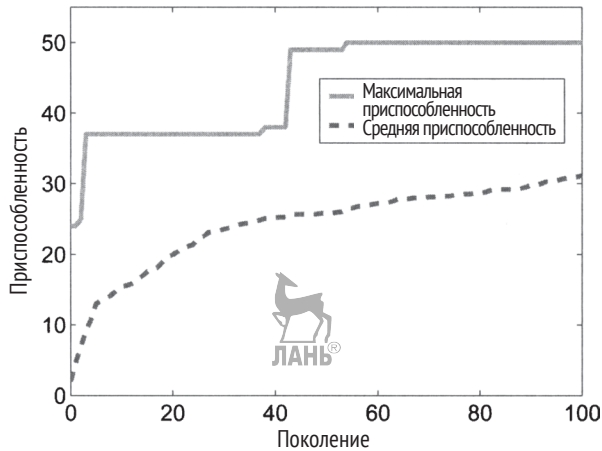


Рис. 5.18. Ход выполнения одной симуляции эволюционной программы эволюционирования конечного автомата для решения задачи искусственного муравья. Каждый конечный автомат имеет пять состояний, и число ходов ограничено 500. Лучший конечный автомат при инициализации позволяет муравью съесть 24 из 90 пищевых гранул. После 100 поколений лучший конечный автомат позволяет муравью съесть 50 пищевых гранул

Мы видим, что имеется большой скачок в результативности, если мы увеличим число состояний с четырех до шести, но после этого увеличение числа состояний приведет к меньшим улучшениям.

После нескольких прогонов Монте-Карло мы обнаружили, что лучший конечный автомат, который эволюционная программа сформировала, имеет 12 состояний. Однако пять из этих состояний так и не были достигнуты, так что фактически конечный автомат содержит только семь оперативных состояний. Этот конечный автомат изображен в формате уравнения (5.11) в виде 28-элементного массива, показанного в табл. 5.3.

Таблица 5.3. Лучший конечный автомат, эволюционно сформированный эволюционной программой для тропы Санта-Фе 32×32. Ходы обозначены следующим образом: 0 = движение вперед, 1 = поворот направо и 2 = поворот налево. Этот конечный автомат показан в графической форме на рис. 5.19

	Пища не ощущается		Пища ощущается	
	Ход	**Следующее состояние**	**Ход**	**Следующее состояние**
Состояние 1	2	2	0	5
Состояние 2	1	3	0	3
Состояние 3	1	4	0	1
Состояние 4	2	5	1	7
Состояние 5	0	1	0	6
Состояние 6	0	5	0	1
Состояние 7	2	6	0	7

На рис. 5.19 изображен конечный автомат в графическом формате. Муравей, перемещавшийся с помощью этого конечного автомата, смог съесть все 90 пищевых гранул за 349 ходов, что немногим более чем в два раза превышает минимальное число ходов. Внимательный анализ рис. 5.19 показывает нам некоторые недостатки конечного автомата. Например, в состоянии 4, если муравей чувствует пищу, он поворачивает направо и переходит в состояние 7. Однако всякий раз, когда муравей чувствует пищу, мы интуитивно ожидаем, что он должен двигаться вперед и съесть пищу. Похоже, что метка «1,R», идущая из состояния 4, расточительна. Мы можем исправить эту проблему, заставив все конечные автоматы всякий раз, когда они чувствуют еду, двигаться вперед. Такой подход явился бы способом встраивания специфичной для конкретной задачи информации в нашу эволюционную программу, который может улучшить результативность эволюционной программы. В общем случае с целью повышения результативности наших эволюционных алгоритмов мы всегда должны стараться встраивать в них специфичную для конкретной задачи информацию.

Другие версии задачи искусственного муравья включают тропу по холмам Лос-Альтос [Koza, 1992, раздел 7.2], тропу Сан-Матео [Koza, 1994, глава 12] и тропу Джона Мьюира [Jefferson и соавт., 2003].

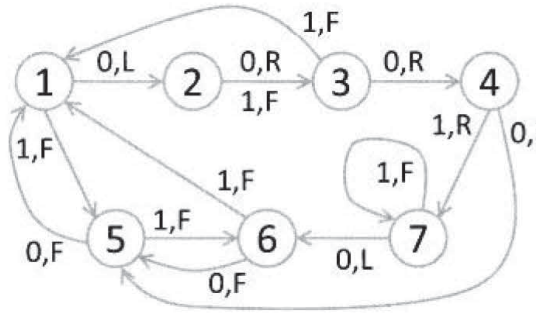


Рис. 5.19. Лучший конечный автомат, эволюционно сформированный эволюционной программой для тропы Санта-Фе 32×32. Выводы каждого состояния имеют маркировку (*f*, *s*), где *f* = 0 обозначает, что еда не ощущается, *f* = 1 говорит, что еда ощущается, и *s* = *F* обозначает движение вперед, *s* = *L* – поворот налево, и *s* = *R* обозначает поворот направо. Этот конечный автомат показан в табличной форме в табл. 5.3

5.6. Заключение

Исторически сложилось так, что эволюционное программирование часто используется для поиска оптимальных конечных автоматов. Вместе с тем в завершение данной главы мы подчеркиваем два важных момента. Во-первых, мы можем применять эволюционную программу в качестве универсального алгоритма для решения любой оптимизационной задачи, и эволюционная программа на самом деле является популярным алгоритмом универсальной оптимизации. Во-вторых, мы можем решать задачи с участием конечных автоматов не только с помощью эволюционного программирования, но и с помощью любых других эволюционных алгоритмов, обсуждаемых в этой книге. Дилемма заключенного и ее вариации являются общими оптимизационными задачами, которые могут решаться с помощью многих типов алгоритмов оптимизации. Причина, по которой в этой главе мы уделили так много внимания дилемме заключенного, заключается в том, что эволюционное программирование изначально было разработано для решения конечных автоматов. В заключение отметим, что в нескольких книгах и исследовательских работах эволюционное программирование обсуждается с других точек зрения, а иногда и подробнее, чем в этой главе, в том числе в публикациях [Bäck и Schwefel, 1993], [Bäck, 1996] и [Yao и соавт., 1999].

Задачи

Письменные упражнения

5.1. Мутационная дисперсия эволюционной программы

$$\sigma_i^2 = \beta\, f(x_i) + \gamma$$

часто упоминается как линейная связь между $f(x_i)$ и σ_i^2. На самом деле она является не линейной, а аффинной. Как записать приведенное выше уравнение, для того чтобы получить линейную связь между $f(x_i)$ и σ_i^2?

5.2. Лифт может быть в одном из двух состояний: на первом этаже либо на втором этаже. Он может принимать один из двух входов: пользователь может нажать кнопку первого этажа или кнопку второго этажа. Напишите конечный автомат для этой системы в графической и табличной формах.

5.3. Разверните систему задачи 5.2 таким образом, чтобы лифт имел четыре состояния (на первом этаже, на втором этаже, перемещение с первого на второй этаж и перемещение со второго на первый этаж) и чтобы он имел три входа (пользователь нажал кнопку первого этажа, кнопку второго этажа или ничего). Напишите конечный автомат для этой системы в графической и табличной формах.

5.4. Напишите стратегию дилеммы заключенного «всегда сотрудничать» в векторной форме уравнения (5.9). Сделайте то же самое для стратегии «око за око», стратегии «око за два ока» и мрачной стратегии. Если исходить из ваших векторных форм, то какие стратегии более похожи: стратегии «всегда сотрудничать» и «око за око» либо стратегии «око за два ока» и «мрачная»?

5.5. Предположим, что стратегии «всегда сотрудничать», «око за око», «око за два ока» и «мрачная» конкурируют между собой в состязании с дилеммой заключенного. Какая из них победит?

5.6. Конечный автомат рис. 5.20 был предложен для искусственного муравья на тропе Санта-Фе, где ввод 0 означает, что пища не ощущается, ввод 1 говорит, что пища ощущается, и выводы L, R и F означают движение соответственно влево, вправо и вперед [Meuleau и соавт., 1999], [Kim, 2006]. Напишите этот конечный автомат в формате уравнения (5.11).

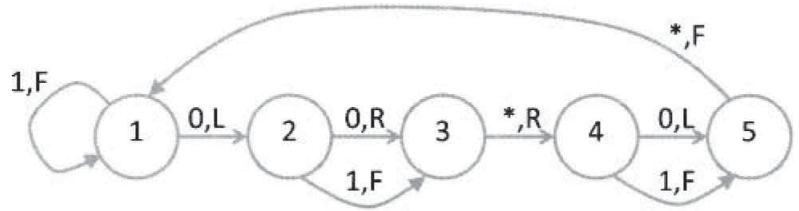


Рис. 5.20. Задача 5.6: конечный автомат для задачи искусственного муравья

5.7. Какое минимальное число ходов требуется искусственному муравью для посещения каждого квадрата тропы Санте-Фе?

5.8. Предположим, что искусственный муравей посещает β уникальных случайных квадратов тропы Санте-Фе 32×32. Какова вероятность того, что муравей найдет всю пищу?

Компьютерные упражнения

5.9. Просимулируйте эволюционную программу на рис. 5.1 для минимизации 10-мерной сферической функции.
Используйте поисковую область [−5.12, +5.12], $\beta = (x_{max} - x_{min})/10 = 1.024$, $\gamma = 0$, размер популяции = 50 и лимит поколений = 50. При расчете мутационной дисперсии нормализуйте значения функции стоимости так, чтобы $f(x_i) \in [1, 2]$ для всех i.

   a) Каким является лучшее решение, полученное эволюционной программой, усредненно по 20 симуляциям Монте-Карло?

   b) Замените дисперсию функцией $\beta f^2(x_i)$ и повторите.

   c) Замените дисперсию функцией $\beta\sqrt{f(x_i)}$ и повторите.

   d) Примените β как дисперсию для всех i и повторите.

5.10. Повторите пример 5.3 со штрафом на размер конечного автомата n, где n – это число состояний. Каково число состояний и стоимость лучшего конечного автомата усредненно по 100 симуляциям Монте-Карло? Как ваши результаты соотносятся с примером 5.3, в котором используется штраф на размер конечного автомата $n/2$?

5.11. Дан соперник дилеммы заключенного, который дезертирует каждый раунд. Мы знаем, что наша лучшая стратегия состоит в том, чтобы тоже дезертировать каждый раунд. Примените эволю-

ционную программу, для того чтобы эволюционно сформировать конечный автомат, который работает как можно лучше против всегда дезертирующего соперника.

5.12. Сколько ходов потребуется муравью, использующему конечный автомат задачи 5.6, для того чтобы съесть всю пищу на тропе Санте-Фе?

5.13. Используйте свой ответ на вопрос задачи 5.8, для того чтобы построить график вероятности для $\beta \in [1, 1024]$ того, что муравей найдет всю пищу на тропе Санте-Фе после посещения β уникальных случайных квадратов. Применяйте логарифмическую шкалу для оси вероятностей. Сколько квадратов муравей должен посетить, для того чтобы иметь, по крайней мере, 50%-ный шанс найти пищу?

Глава 6

Эволюционные стратегии

Первая версия эволюционной стратегии работала только с одним ребенком на «поколение», потому что у нас не было популяции объектов для работы.

– Ханс-Пол Швефель [Schwefel и Mendes, 2010]

Ранний европейский набег на эволюционные алгоритмы был осуществлен в Берлинском техническом университете в 1960-х годах тремя студентами, которые пытались найти оптимальные формы тела в аэродинамической трубе, для того чтобы минимизировать сопротивление воздуха. Студенты, Инго Рехенберг (Ingo Rechenberg), Ханс-Пол Швефель (Hans-Paul Schwefel) и Питер Бинерт (Peter Bienert), столкнулись с трудностями в решении своей задачи аналитическим путем. Поэтому они пришли к идее случайного изменения форм тела, отбирая те, которые работали лучше всего, и повторяя процесс до тех пор, пока они не нашли хорошего решения своей задачи.

Первая публикация Рехенберга по эволюционным стратегиям (evolution strategy, ES), которая так и называется – «Эволюционная стратегия», была опубликована в 1964 году [Rechenberg, 1998]. Интересно, что первые реализации были экспериментальными. Вычислительных ресурсов для высокоточных симуляций не хватало, поэтому функции приспособленности были получены экспериментально, а мутации реализованы на физическом оборудовании. В 1970 году за свои усилия Рехенберг получил докторскую степень и позже опубликовал свою работу в виде книги [Rechenberg, 1973]. Несмотря на то что данная книга написана на немецком языке, она по-прежнему интересна для ненемецких читателей из-за ее графических изображений процессов оптимизации. В книге показана эволюция форм крыла для минимизации сопротивления в поле воздушного потока, формы сопла ракеты для минимизации

сопротивления топлива при прохождении через сопло и формы трубы для минимизации сопротивления жидкости при прохождении через трубу. Эти ранние алгоритмы были названы *кибернетическими траекториями решения*.

Швефель получил докторскую степень в 1975 году, а позже написал несколько книг об эволюционных стратегиях [Schwefel, 1977], [Schwefel, 1981], [Schwefel, 1995]. С 1985 года он работает в дортмундском университете. Бинерт получил свою докторскую степень в 1967 г. Интересный рассказ о ранних годах эволюционных стратегий приводится в интервью Швефеля в публикации [Schwefel и Mendes, 2010].

Краткий обзор главы

В разделе 6.1 рассматривается эволюционная стратегия (1 + 1). Эта эволюционная стратегия стала первой и самой простой стратегией, которая была применена ее разрабочиками. Она состоит исключительно из мутации и не предусматривает рекомбинации. В разделе 6.2 выводится правило 1/5 эволюционных стратегий, которое говорит нам, как корректировать скорость мутации для получения лучшей результативности, и этот раздел не заинтересованные в математических доказательствах читатели могут пропустить. В разделе 6.3 обобщается эволюционная стратегия (1 + 1) для получения алгоритма с μ родителями в каждом поколении, где μ – это определяемая пользователем константа. Родители комбинируются, формируя одного ребенка, который, возможно, станет частью следующего поколения, если он будет достаточно приспособлен. Для рекомбинации есть несколько вариантов. Раздел 6.4 является еще одним обобщением, которое в результате приводит к появлению λ детей в каждом поколении. В разделе 6.5 описано, как можно адаптировать скорость мутации, для того чтобы резко повысить результативность эволюционной стратегии. Эти варианты адаптации включают в себя передовые алгоритмы ковариационно-матричной адаптивной эволюционной стратегии (CMA-ES) и ковариационно-матричной самоадаптивной эволюционной стратегии (CMSA-ES).

6.1. Эволюционная стратегия (1 + 1)

Предположим, что $f(x)$ является функцией вещественного случайного вектора x и что мы хотим максимизировать приспособленность $f(x)$. Исходный алгоритм эволюционной стратегии работал путем инициализации одного кандидатного решения и оценки его приспособленности.

Затем кандидатное решение мутировалось, и оценивалась приспособленность мутировавшей особи. Лучшее из двух кандидатных решений (родительское и дочернее) формировало отправную точку для следующего поколения. Оригинальная эволюционная стратегия была разработана для дискретных задач, использовала небольшие мутации в дискретном поисковом пространстве и, следовательно, имела тенденцию попадать в локальный оптимум. По этой причине исходная эволюционная стратегия была модифицирована для использования непрерывных мутаций в непрерывных поисковых пространствах [Beyer и Schwefel, 2002]. Этот алгоритм кратко сформулирован на рис. 6.1.

Инициализировать неотрицательную мутационную дисперсию σ^2
$x_0 \leftarrow$ случайно сгенерированная особь
While not (критерий останова)
    Сгенерировать случайный вектор r с $r_i \sim N(0, \sigma^2)$ для $i \in [1, n]$
    $x_1 \leftarrow x_0 + r$
    If x_1 лучше x_0 **then**
        $x_0 \leftarrow x_1$
    End if
Next поколение

Рис. 6.1. Приведенный выше псевдокод дает схематичное описание эволюционной стратегии (1 + 1), где n – размерность задачи, а x_1 равняется x_0, но где каждый элемент мутирован

Рисунок 6.1 называется эволюционной стратегией (1 + 1), потому что каждое поколение состоит из 1 родителя и 1 ребенка, и лучшая особь отбирается из родителя и ребенка как особь в следующем поколении. Эволюционная стратегия (1 + 1), так называемая двухчленная эволюционная стратегия, очень похожа на стратегии восхождения к вершине холма в разделе 2.6. Она также идентична эволюционной программе с размером популяции 1 (см. раздел 5.1). Следующая ниже теорема гарантирует, что эволюционная стратегия (1 + 1) в конечном итоге находит глобальный максимум $f(x)$.

Теорема 6.1. Если $f(x)$ есть непрерывная функция, определенная на закрытой области с глобальным оптимумом $f^*(x)$, то

$$\lim_{t \to \infty} f(x) = f^*(x), \tag{6.1}$$

где t – номер поколения.

Теорема 6.1 доказывается в публикациях [Devroye, 1978], [Rudolph, 1992], [Bäck, 1996, теорема 7] и [Michalewicz, 1996]. Она также согласуется с нашей интуицией. Поскольку мы используем случайные мутации для разведывания поискового пространства, при наличии достаточного времени мы в конечном итоге разведаем все поисковое пространство (в пределах прецизионности компьютера) и найдем глобальный оптимум.

Дисперсия σ^2 в эволюционной стратеги (1 + 1) рис. 6.1 является регулировочным параметром. Значение σ является компромиссом.

- σ должна быть достаточно большой, для того чтобы мутации могли достичь всех участков поискового пространства за разумный период времени.
- σ должна быть достаточно малой, для того чтобы поиск мог найти оптимальное решение в желаемой для пользователя разрешающей способности.

По мере поступательного развития эволюционной стратегии может оказаться целесообразным уменьшать σ. В начале эволюционной стратегии большие значения σ позволят эволюционной стратегии проводить крупнозернистый поиск и приблизиться к оптимальному решению. Ближе к концу эволюционной стратегии меньшие значения σ позволят ей точно настроить свое потенциальное решение и сойтись к оптимальному решению с более высокой разрешающей способностью.

Мутация на рис. 6.1 называется *изотропной*, поскольку мутация каждого элемента x_0 имеет одинаковую дисперсию. На практике мы, возможно, захотим реализовать неизотропные мутации следующим образом:

$$x_1 \leftarrow x_0 + N(0, \Sigma), \tag{6.2}$$

где Σ – диагональная $n \times n$-матрица с диагональными элементами σ_i, для $i \in [1, n]$. Из этого следует, что каждый элемент x_0 мутирует с разной дисперсией. Мы назначаем каждую σ_i независимо в зависимости от области i-го элемента x и формы целевой функции в этой размерности.

Рехенберг проанализировал эволюционную стратегию (1 + 1) для некоторых простых оптимизационных задач и пришел к выводу, что 20 % мутаций должны привести к улучшению функции приспособленности $f(x)$ [Rechenberg, 1973], [Bäck, 1996, раздел 2.1.7]. Мы воспроизведем

некоторые части его анализа в разделе 6.2. Если уровень успеха мутации выше 20 %, то мутации слишком малы, что приводит к небольшим улучшениям, в результате давая длительное время схождения. Если уровень успеха мутации ниже 20 %, то мутации слишком велики, что приводит к большим, но нечастым улучшениям, и это также дает длительное время схождения. Следствием работы Рехенберга стало правило 1/5:

> *Если в эволюционной стратегии (1 + 1) соотношение успешных мутаций к общим мутациям меньше 1/5, то стандартное отклонение σ должно быть уменьшено. Если соотношение больше 1/5, то стандартное отклонение должно быть увеличено.*

Это правило действительно относится только к нескольким конкретным целевым функциям, как мы увидим в разделе 6.2, но оно оказалось полезным руководством для общих реализаций эволюционных стратегий. Однако правило 1/5 ставит вопрос: на сколько должно быть уменьшено или увеличено стандартное отклонение? Швефель теоретически обосновал коэффициент, на который следует уменьшать или увеличивать σ:

$$\begin{aligned} &\text{уменьшение стандартного отклонения:} && \sigma \leftarrow c\sigma \\ &\text{увеличение стандартного отклонения:} && \sigma \leftarrow \sigma/c \\ &\text{где} && c = 0.817. \end{aligned} \tag{6.3}$$

Эти результаты приводят к адаптивной эволюционной стратегии (1 + 1), показанной на рис. 6.2. Адаптивная эволюционная стратегия (1 + 1) требует определить длину движущегося окна G. Мы хотим, чтобы G было достаточно большим, для того чтобы получить хорошее представление об успешности мутаций эволюционной стратегии, но не настолько большим, чтобы адаптация σ была вялой. В публикации [Beyer и Schwefel, 2002] рекомендуется, чтобы

$$G = \min(n, 30), \tag{6.4}$$

где n – это размерность задачи.

Инициализировать неотрицательную мутационную дисперсию σ^2
$x_0 \leftarrow$ случайно сгенерированная особь
While not (критерий останова)

Сгенерировать случайный вектор r с $r_i \sim N(0, \sigma^2)$ для $i \in [1, n]$
$x_1 \leftarrow x_0 + r$
If x_1 лучше x_0 **then**
    $x_0 \leftarrow x_1$
End if
$\phi \leftarrow$ доля успешных мутаций во время прошлых G поколений
If $\phi < 1/5$
    $\sigma \leftarrow c^2\sigma$
else if $\phi > 1/5$
    $\sigma \leftarrow \sigma/c^2$
End if
Next поколение

Рис. 6.2. Приведенный выше псевдокод дает схематичное описание адаптивной эволюционной стратегии (1 + 1), где n – размерность задачи. x_1 равно x_0, но где каждый элемент мутирован. ϕ – это доля мутаций за последние G поколений, которые приводят к тому, что x_1 лучше, чем x_0. Мутационная дисперсия автоматически корректируется для увеличения скорости схождения. Номинальное значение c равняется 0.817

■ Пример 6.1

В данном примере мы используем эволюционную стратегию (1 + 1) для оптимизации 20-мерной эталонной функции Экли (см. приложение С.1.2). Мы сравниваем стандартную эволюционную стратегию (1 + 1), показанную на рис. 6.1, с адаптивной эволюционной стратегией (1 + 1), показанной на рис. 6.2. В случае адаптивной эволюционной стратегии мы отслеживаем число успешных мутаций и общее число мутаций. Общее число мутаций равно числу поколений. Каждые 20 поколений мы исследуем уровень успешности мутации и корректируем стандартные отклонения, как показано в уравнении (6.3). На рис. 6.3 сравнивается средняя скорость схождения стандартной эволюционной стратегии (1 + 1) и адаптивной эволюционной стратегии (1 + 1). Мы видим, что адаптивная эволюционная стратегия сходится гораздо быстрее, чем стандартная. На рис. 6.4 показаны типичный профиль успешности мутации и стандартное отклонение мутации. Мы видим, что когда уровень успешности превышает 20 %, по сравнению с предыдущим промежутком из 20 поколений, стандартное отклонение мутации автоматически увеличивается; когда уровень успешности меньше 20 %, стандартное отклонение мутации автоматически уменьшается.

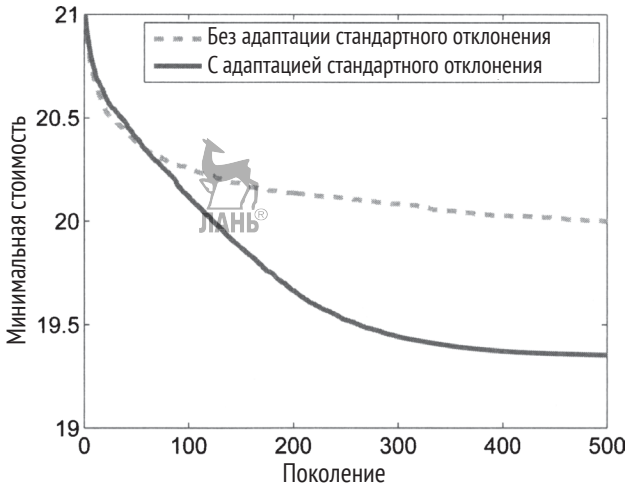


Рис. 6.3. Пример 6.1: схождение алгоритмов эволюционной стратегии (1 + 1), усредненно по 100 симуляциям. Адаптивная эволюционная стратегия, которая автоматически регулирует стандартные отклонения мутации, сходится гораздо быстрее, чем стандартная эволюционная стратегия

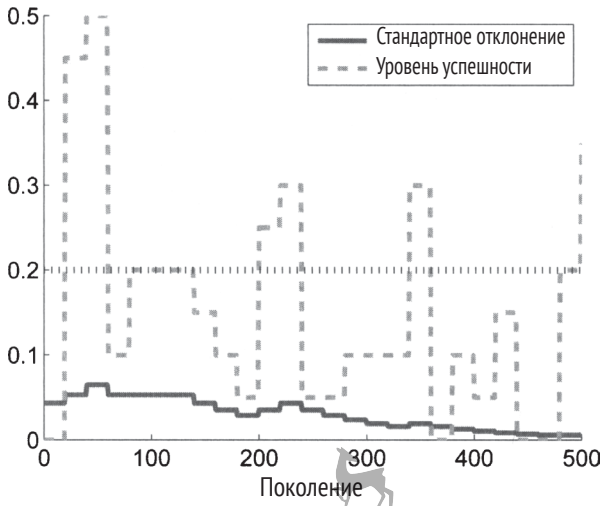


Рис. 6.4. Пример 6.1: уровень успешности мутации и стандартное отклонение мутации адаптивной эволюционной стратегии (1 + 1). Адаптивная эволюционная стратегия автоматически увеличивает стандартное отклонение мутации, когда уровень успешности составляет более 20 %, и снижает стандартное отклонение мутации, когда уровень успешности составляет менее 20 %. Этим иллюстрируется правило 1/5

■

6.2. Правило 1/5: деривация

В этом разделе выводится правило 1/5, которое констатирует, что приблизительно 20 % всех мутаций должны приводить к улучшению мутировавшей особи эволюционной стратегии. Этот раздел мотивирован книгой [Rechenberg, 1973, главы 14–15] и может быть пропущен читателями, которые не заинтересованы в деталях математических доказательств.

Предположим, что у нас есть n-мерная задача минимизации с функцией стоимости $f(x)$, где $x = [x_1 \ldots x_n]$. В данном разделе мы сосредоточимся на коридорной задаче, которая имеет область

$$\begin{aligned} x_1 &\in [0, \infty) \\ x_j &\in (\infty, \infty), \quad j \in [2, n]. \end{aligned} \tag{6.5}$$

Коридорная задача имеет функцию стоимости

$$f(x) = \begin{cases} c_0 + c_1 x_1, & \text{если } x_j \in [-b, b] \text{ для всех } j \in [2, n] \\ \infty, & \text{в противном случае,} \end{cases} \tag{6.6}$$

где c_0, c_1 и b – это положительные константы. Данная задача называется коридорной задачей, потому что стоимость улучшается по мере уменьшения x_1, но только если x находится в коридоре $x_j \in [-b, b]$ для всех $j \in [2, n]$.

Напомним по рис. 6.1 и 6.2, что заданная особь эволюционной стратегии x_0 мутируется в соответствии с уравнением $x_1 \leftarrow x_0 + r$, где r – это случайный n-элементный вектор[1]. Мы используем x_{0j} для обозначения j-го элемента вектора x_0 и x_{1j} для обозначения j-го элемента вектора x_1, для $j \in [1, n]$.

Мутация каждого элемента x_0 отбирается из гауссова распределения с нулевым средним и дисперсией σ^2. Следовательно, функция плотности вероятности (PDF) x_{1j} может быть записана как

$$\text{PDF}(x_{1j}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-(x_{1j} - x_{0j})^2 / (\sigma^2)\right], \quad j \in [1, n]. \tag{6.7}$$

Улучшающая x мутация требует появления n независимых событий:

$$\begin{aligned} r_1 &< 0 \\ x_{1j} &\in [-b, b], \quad j \in [2, n] \end{aligned} \tag{6.8}$$

[1] Здесь присутствует вре́менная несогласованность в обозначениях. x_0 и x_1 на рис. 6.1 и 6.2 относятся к кандидатному решению эволюционной стратегии до и после мутации, в то время как x_1 в уравнениях (6.5) и (6.6) относятся к первому элементу вектора x.

где r_1 – это мутация первого элемента x_0 (то есть $x_{11} \leftarrow x_{01} + r_1$). Следовательно, ϕ', ожидаемая величина полезной мутации, равна

$$\begin{aligned}\phi' &= \left|E(r_1 \mid r_1 < 0)\right| \prod_{j=2}^{n} \text{Prob}(x_{1j} \in [-b, b]) \\ &= \left| \int_{-\infty}^{0} \frac{r_1}{\sigma\sqrt{2\pi}} \exp(-r_1^2 / (2\sigma^2)) dr_1 \right| \prod_{j=2}^{n} \int_{-b}^{b} \frac{1}{\sigma\sqrt{2\pi}} \exp(-(x_{1j} - x_{0j})^2 / (2\sigma^2)) dx_{1j} \\ &= \frac{\sigma}{\sqrt{2\pi}} \prod_{j=2}^{n} \frac{1}{2} \left[\text{erf}\left(\frac{b - x_{0j}}{\sigma\sqrt{2}} \right) + \text{erf}\left(\frac{b + x_{0j}}{\sigma\sqrt{2}} \right) \right] \end{aligned} \quad (6.9)$$

(см. задачу 6.2), где erf(·) – это функция ошибок:

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp -(t^2) dt, \quad x \geq 0. \quad (6.10)$$

Ожидаемая величина полезной мутации, при условии что XQ находился в коридоре [–6, 6] до мутации, может быть записана как

$$\begin{aligned}\phi &= E(\phi' \mid x_{0j} \in [-b, b] \text{ для всех } j \in [2, n]) \\ &= \int_{-b}^{b} \cdots \int_{-b}^{b} \phi' \, \text{PDF}(x_{02}) \ldots \text{PDF}(x_{0n}) dx_{02} \ldots dx_{0n}. \end{aligned} \quad (6.11)$$

При условии что $x_{0j} \in [-b, b]$, мы принимаем допущение, что x_{0j} равномерно распределен в $[-b, b]$ (по общему признанию недоказанное предположение). В результате получаем:

$$\phi = \frac{\sigma}{\sqrt{2\pi}} \prod_{j=2}^{n} \int_{-b}^{b} \frac{1}{2} \left[\text{erf}\left(\frac{b - x_{0j}}{\sigma\sqrt{2}} \right) + \text{erf}\left(\frac{b + x_{0j}}{\sigma\sqrt{2}} \right) \right] \left(\frac{1}{2b} \right) dx_{0j}. \quad (6.12)$$

Теперь напомним, что

$$\int \text{erf}(z) dz = z \, \text{erf}(z) + \frac{\exp(-z^2)}{\sqrt{\pi}}. \quad (6.13)$$

Применение этого выражения в уравнении (6.12) дает

$$\begin{aligned}\phi &= \frac{\sigma}{\sqrt{2\pi}(4b)^{n-1}} \prod_{j=2}^{n} \left[4b \, \text{erf}\left(\frac{2b}{\sigma\sqrt{2}} \right) + \frac{2\sigma\sqrt{2}(\exp(-2b^2 / \sigma^2) - 1)}{\sqrt{\pi}} \right] \\ &= \frac{\sigma}{\sqrt{2\pi}} \left[\text{erf}\left(\frac{2b}{\sigma\sqrt{2}} \right) - \frac{\sigma}{b\sqrt{2\pi}} (1 - \exp(-2b^2 / \sigma^2)) \right]^{n-1}. \end{aligned} \quad (6.14)$$

Напомним, что $\lim_{x\to\infty}\text{erf}(x) = 1$ и $\lim_{x\to\infty}\exp(-x) = 0$. Следовательно,

$$\phi \approx \frac{\sigma}{\sqrt{2\pi}}\left(1-\frac{\sigma}{b\sqrt{2\pi}}\right)^{n-1} \quad \text{для больших } b/\sigma. \tag{6.15}$$

ϕ – это ожидаемая величина полезной мутации, которую мы хотели бы видеть большой. Мы можем найти значения σ, приводящие к экстремальным значениям ϕ, взяв производную ϕ по σ и приравняв результат к 0. Мы находим, что

$$\frac{d\phi}{d\sigma} = \frac{1}{\sqrt{2\pi}}\left(1-\frac{\sigma}{b\sqrt{2\pi}}\right)^{n-1} - \frac{\sigma(n-1)}{b2\pi}\left(1-\frac{\sigma}{b\sqrt{2\pi}}\right)^{n-2}. \tag{6.16}$$

Приравняв эту производную к 0 и решив для σ, получим

$$\sigma^* = b\sqrt{2\pi}/n, \tag{6.17}$$

что в результате дает наибольшее возможное значение ϕ и, следовательно, наибольшую ожидаемую величину полезной мутации.

Теперь рассмотрим вероятность w', что мутация полезна. Мутация полезна, если происходят n независимых событий, показанных в уравнении (6.8). Вероятность того, что это произойдет, может быть записана как

$$w' = \text{Prob}(r_1 < 0)\prod_{j=2}^{n}\text{Prob}(x_{1j} \in [-b, b]). \tag{6.18}$$

Поскольку r_1 является нулевым средним значением, вероятность того, что $r_1 < 0$, равна половине, поэтому приведенное выше уравнение можно записать в виде

$$w' = \tfrac{1}{2}\prod_{j=2}^{n}\text{Prob}(x_{1j} \in [-b, b]). \tag{6.19}$$

Сравнивая данное уравнение с уравнением (6.9), мы видим, что

$$w' = \frac{\sqrt{2\pi}}{2\sigma}\phi'. \tag{6.20}$$

Теперь рассмотрим ожидаемое значение w вероятности, что мутация полезна, при условии что x_0 было в коридоре $[-b, b]$ до мутации. Это можно записать следующим образом:

$$w = E(w' \mid x_{0j} \in [-b, b] \text{ для всех } j \in [2, n])$$
$$= \int_{-b}^{b} \cdots \int_{-b}^{b} w' \,\mathrm{PDF}(x_{02}) \ldots \mathrm{PDF}(x_{0n}) dx_{02} \ldots dx_{0n}. \quad (6.21)$$

Сравнивая это уравнение с уравнениями (6.11) и (6.20), мы видим, что

$$w = \frac{\sqrt{2\pi}}{2\sigma}\phi. \quad (6.22)$$

Теперь подставим ϕ из уравнения (6.15) и получим

$$w = \left(\frac{\sqrt{2\pi}}{2\sigma}\right)\left(\frac{\sigma}{\sqrt{2\pi}}\right)\left(1 - \frac{\sigma}{b\sqrt{2\pi}}\right)^{n-1}. \quad (6.23)$$

Далее мы подставляем оптимальные значения σ из уравнения (6.17) для получения оптимального значения w:

$$w^* = \tfrac{1}{2}(1 - 1/n)^{n-1}. \quad (6.24)$$

Напомним, что $\exp(-x) \approx 1 - x$ для малых x. Следовательно, $\exp(-1/n) \approx 1 - 1/n$ для больших n. Это дает

$$w^* = \tfrac{1}{2}(\exp(-1/n))^{n-1} = \tfrac{1}{2e} \approx 0.18. \quad (6.25)$$

Оптимальное стандартное отклонение σ^* приводит к величине мутации, которая дает улучшения в 18 % случаев.

Рехенберг также проанализировал сферическую функцию, где цель состояла в том, чтобы минимизировать

$$f(x) = \sum_{j=1}^{n} x_j^2. \quad (6.26)$$

Он обнаружил, что оптимальный уровень успешности мутации для сферической функции составляет 27 %.

Приведенные выше результаты применимы только к конкретной функции, и они были выведены в условиях упрощающих аппроксимаций, но в итоге привели к эмпирическому правилу 1/5, которое оказалось полезным для многих задач. Для того чтобы максимизировать скорость схождения, стандартные отклонения в эволюционной стратегии должны быть скорректированы и давать соотношение 1/5 успешных мутаций к общим мутациям.

6.3. Эволюционная стратегия (μ + 1)

Первым обобщением эволюционной стратегии (1 + 1) является эволюционная стратегия (μ + 1). В эволюционной стратегии (μ + 1) родители μ используются в каждом поколении, где μ – это определяемый пользователем параметр. Каждый родитель также имеет связанный с ним вектор σ, который управляет величиной мутаций. Родители комбинируются друг с другом и формируют одного ребенка, а затем ребенок мутирует. Лучшие μ особей отбираются из числа μ родителей и ребенка, и они становятся μ родителями следующего поколения. Данный алгоритм кратко сформулирован на рис. 6.5. Поскольку эволюционная стратегия (μ + 1) сохраняет лучших особей в каждом поколении, он элитарен; то есть его лучшая особь никогда не становится хуже из поколения в поколение (см. раздел 8.4). Эволюционная стратегия (μ + 1) также называется стационарной эволюционной стратегией. Поскольку в конце каждого поколения из общей популяции удаляется только одна особь, эту стратегию можно назвать *вымиранием худшего*, которая является обратной стороной выживания наиболее приспособленных.

Родители, показанные на рис. 6.5, сочетают в себе как признаки решения, так и их мутационные дисперсии. Вместе с тем рис. 6.5 не включает тип адаптации мутационной дисперсии, которую мы видели на рис. 6.2. Фактически после некоторого числа поколений, пропорциональных μ, значения σ на рис. 6.5 свернутся до одного значения, которое может отражать или не отражать соответствующую силу мутации [Beyer, 1998]. Рисунок 6.5, вероятно, не должен быть реализован, как показано на изображении, однако он является ступенькой к более эффективным самоадаптивным эволюционным стратегиям раздела 6.5.

$\{(x_k, \sigma_k)\} \leftarrow$ случайно сгенерированные особи, $k \in [1, \mu]$.

Каждое x_k является кандидатным решением,

и каждый σ_k – вектором стандартного отклонения.

Отметить, что $x_k \in R^n$ и $\sigma_k \in R^n$, где каждый элемент положителен.

While not (критерий останова)

Случайно отобрать родителей из популяции $\{(x_k, \sigma_k)\}$

Применить метод рекомбинации для объединения двух родителей

и получения ребенка, который обозначается как $(x_{\mu+1}, \sigma_{\mu+1})$

$\Sigma_{\mu+1} \leftarrow \text{diag}(\sigma^2_{\mu+1,1}, ..., \sigma^2_{\mu+1,n}) \in R^{n \times n}$

Сгенерировать случайный вектор r из $N(0, \Sigma_{\mu+1})$

$x_{\mu+1} \leftarrow x_{\mu+1} + r$

Удалить худшую особь из популяции: то есть

$\{(x_k, \sigma_k)\} \leftarrow$ лучшие μ особей из $\{(x_1, \sigma_1), ..., (x_{\mu+1}, \sigma_{\mu+1})\}$

Next поколение

Рис. 6.5. Приведенный выше псевдокод дает схематичное описание эволюционной стратегии (μ + 1), где n – это размерность задачи

Шаг рекомбинации на рис. 6.5 гласит: «Применить метод рекомбинации для объединения двух родителей...». Существуют разные способы выполнения рекомбинации[2]. Дискретное половое скрещивание работает путем случайного отбора каждого ребенка из x_p или x_q и случайного отбора каждого стандартного отклонения ребенка из σ_p или σ_q, где каждый отбор независим от других. Этот тип скрещивания описывается словом «*дискретный*», потому что каждый детский признак происходит от одного родителя, и он описывается словом «*половой*», поскольку каждый детский признак происходит от одного из двух родителей. Дискретное половое скрещивание проиллюстрировано на рис. 6.6.

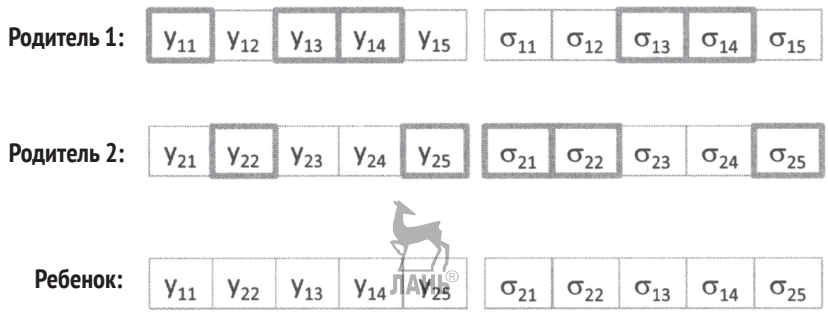


Рис. 6.6. Дискретное половое скрещивание в эволюционной стратегии, где размерность задачи n = 5. Каждый признак решения и стандартное отклонение в ребенке отбирается случайно у одного из двух родителей

Еще один вариант рекомбинации – промежуточное половое скрещивание. В этом варианте детские признаки принимаются равными срединной точке их родительских признаков; отсюда и обозначение «*промежуточное*». Промежуточное половое скрещивание показано на рис. 6.7.

Еще одним вариантом рекомбинации является глобальное скрещивание, или панмиктическое скрещивание. Панмиктическая популяция – это такая популяция, в которой каждая особь является потенциальной

[2] Сообщество исследователей эволюционных стратегий обычно предпочитает термин *рекомбинация* или *смешивание*, а не скрещивание. На протяжении всей этой книги мы используем эти термины как синонимы.

парой для каждой особи. В дискретном глобальном скрещивании каждый детский признак происходит от родителя, отобранного случайно из всей популяции. Это показано на рис. 6.8. Варианты дискретного скрещивания на рис. 6.6 и 6.8 также называются доминантным скрещиванием.

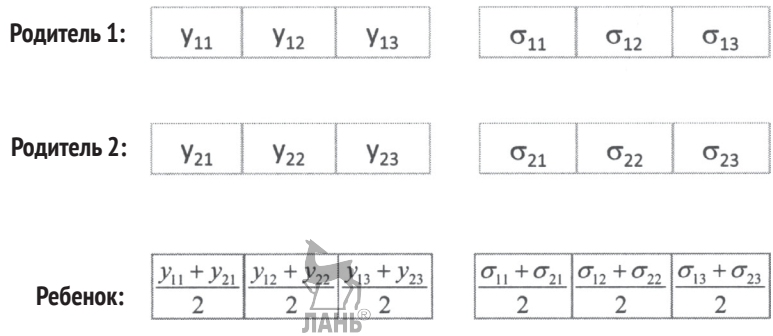


Рис. 6.7. Промежуточное половое скрещивание в эволюционной стратегии, где размерность задачи $n = 3$. Каждый признак решения и стандартное отклонение в ребенке находятся между двумя родителями

Рис. 6.8. Дискретное глобальное скрещивание в пятичленной эволюционной стратегии ($\mu = 5$), где размерность задачи $n = 5$. Каждый признак решения и стандартное отклонение в ребенке отбираются случайно из всей популяции в целом

Глобальное скрещивание можно совместить с промежуточным скрещиванием, для того чтобы получить промежуточное глобальное скрещивание. В этом варианте каждый детский признак представляет собой

линейную комбинацию случайно отобранной пары родителей. Это показано на рис. 6.9. Промежуточное глобальное скрещивание – это тип скрещивания, который обычно используется на практике, и он также рекомендуется по хорошо понятным теоретическим причинам [Beyer и Schwefel, 2002].

Другие типы скрещивания можно тоже использовать. Например, один родитель $x_{p(0)}$ может быть отобран для ребенка, а n других родителей могут быть отобраны, $\{x_{p(k)}\}$ для $k \in [1, n]$, по одному для каждого признака решения. Тогда признак решения k-го ребенка $x_{\mu+1,k}$ может быть сгенерирован путем скрещивания $x_{p(0)}$ с $x_{p(k)}$ для $k \in [1, n]$. Аналогичным образом стандартное отклонение $\sigma_{\mu+1,k}$ k-го ребенка может быть сгенерировано скрещиванием $\sigma_{p(0)}$ с $\sigma_{p(k)}$. Обратитесь к разделу 8.8 относительно дополнительных типов операторов скрещивания.

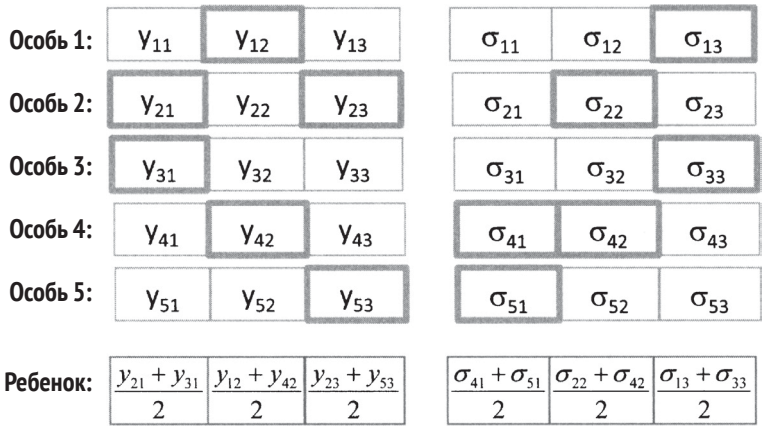


Рис. 6.9. Иллюстрация промежуточного глобального скрещивания в пятичленной эволюционной стратегии ($\mu = 5$), где размерность задачи $n = 3$. Каждый признак решения и стандартное отклонение в ребенке находятся между двумя случайно отобранными родителями

6.4. Эволюционные стратегии ($\mu + \lambda$) и (μ, λ)

Следующим обобщением эволюционной стратегии является эволюционная стратегия ($\mu + \lambda$). В эволюционной стратегии ($\mu + \lambda$) мы имеем размер популяции μ, и мы генерируем λ детей в каждом поколении. После генерации детей у нас в общей сложности ($\mu + \lambda$) особей, которые включают в себя как родителей, так и детей. Из этих особей мы отбираем μ лучших в качестве родителей следующего поколения.

Еще одной широко используемой эволюционной стратегией является эволюционная стратегия (μ, λ). В эволюционной стратегии (μ, λ) родители следующего поколения отбираются как лучшие μ особей из λ числа детей. Другими словами, ни один из родителей μ не доживает до следующего поколения; вместо этого отбирается подмножество μ из λ детей, становящихся родителями следующего поколения. В эволюционной стратегии (μ, λ) нам нужно сделать так, чтобы мы отбирали $\lambda \geq \mu$. Родители предыдущего поколения никогда не доживают до следующего поколения. Жизнь каждой особи ограничена одним поколением.

Если в эволюционной стратегии ($\mu + \lambda$) или эволюционной стратегии (μ, λ) $\mu > 1$, то эволюционная стратегия называется многочленной. Несмотря на успех этих обобщений, изначально были серьезные возражения против принятия μ и λ равными больше 1. Аргумент против $\lambda > 1$ был в том, что эксплуатация информации будет отложена. Аргумент против $\mu > 1$ заключался в том, что выживание ущербных особей задержит поступательное развитие эволюционной стратегии [De Jong и соавт., 1997].

Эволюционная стратегия (μ, λ) часто работает лучше, чем эволюционная стратегия ($\mu + \lambda$), когда функция приспособленности шумная или варьируется во времени (глава 21). В эволюционной стратегии ($\mu + \lambda$) конкретная особь (x, σ) может иметь хорошую приспособленность, но вряд ли улучшится из-за не соответствующей σ. Поэтому особь (x, σ) может оставаться в популяции в течение многих поколений без улучшения, что приводит к пустой трате места в популяции. Эволюционная стратегия ($\mu + \lambda$) эту проблему решает, вытесняя из популяции всех особей через одно поколение и позволяя выживать только лучшим детям. Это помогает ограничить выживания в следующем поколении только детьми с хорошей σ, то есть σ, приводящей к мутационному вектору, который дает улучшение в x. В публикации [Beyer и Schwefel, 2002] эволюционная стратегия (μ, λ) рекомендуется для задач с неограниченным пространством, эволюционная стратегия ($\mu + \lambda$) – для задач с дискретным пространством.

На рис. 6.10 кратко сформулированы эволюционные стратегии ($\mu + \lambda$) и (μ, λ). Обратите внимание, что если σ_k = константа для всех k, то значения σ_k останутся неизменными из поколения в поколение. Рисунок 6.10, как и рис. 6.5, не включает адаптацию мутационной дисперсии. Следовательно, рис. 6.10 не должен быть реализован, как показано, и является ступенькой к самоадаптивной эволюционной стратегии. Мы обобщим рис. 6.10, получив самоадаптивные эволюционные стратегии ($\mu + \lambda$) и (μ, λ) в следующем далее разделе.

$\{(x_k, \sigma_k)\} \leftarrow$ случайно сгенерированные особи, $k \in [1, \mu]$.
Каждое x_k является кандидатным решением, и каждый σ_k – вектором стандартного отклонения.
Отметить, что $x_k \in R^n$ и $\sigma_k \in R^n$, где каждый элемент положителен.
While not (критерий останова)
  For $k = 1, ..., \lambda$
    Случайно отобрать родителей из популяции $\{(x_k, \sigma_k)\}$
    Применить метод рекомбинации для объединения двух родителей и получения ребенка, который обозначается как (x'_k, σ'_k)
    $\Sigma'_{k1} \leftarrow \text{diag}((\sigma'_{k1})^2, ..., (\sigma'_{kn})^2) \in R^{n \times n}$
    Сгенерировать случайный вектор r из $N(0, \Sigma'_k)$
    $x'_k \leftarrow x'_k + r$
  Next k
  If – это эволюционная стратегия $(\mu + \lambda)$ **then**
    $\{(x_k, \sigma_k)\} \leftarrow$ лучшие μ особей из $\{(x_k, \sigma_k)\} \cup \{(x'_k, \sigma'_k)\}$
  else if – это эволюционная стратегия $(\mu + \lambda)$ **then**
    $\{(x_k, \sigma_k)\} \leftarrow$ лучшие μ особей из $\{(x'_k, \sigma'_k)\}$
  End if
Next поколение

Рис. 6.10. Приведенный выше псевдокод дает схематичное описание эволюционных стратегий $(\mu + \lambda)$ и (μ, λ), где n – это размерность задачи

■ Пример 6.2

В данном примере мы сравниваем эволюционные стратегии $(\mu + \lambda)$ и (μ, λ). Мы выполняем обе стратегии с $\mu = 10$, $\lambda = 20$, с дискретным половым скрещиванием и размерностью задачи 20. На рис. 6.11 показана средняя результативность обоих алгоритмов на эталоне Швефеля 2.26. Мы видим, что эволюционная стратегия $(\mu + \lambda)$ превосходит эволюционную стратегию (μ, λ). В общем случае это ожидаемо, потому что эволюционная стратегия (μ, λ), возможно, отбросила хорошее решение из-за ограничения жизни каждой особи одним поколением. Однако сравнение результативности между двумя вариантами эволюционной стратегии зависит от задачи. На рис. 6.12 показана средняя результативность двух алгоритмов на эталоне Экли. Эволюционная стратегия $(\mu + \lambda)$ изначально превосходит эволюционную стратегию (μ, λ), но в конечном итоге эволюционная стратегия (μ, λ) догоняет и работает лучше, чем эволюционная стратегия $(\mu + \lambda)$. Иногда эволюционная стратегия (μ, λ) целесообразна. И не только из-за своей большой адаптируемости к

шумным и варьирующимся со временем приспособленностям; в некоторых функциях ее больший упор на разведывании в результате приводит к более высокой результативности, чем у эволюционной стратегии $(\mu + \lambda)$.

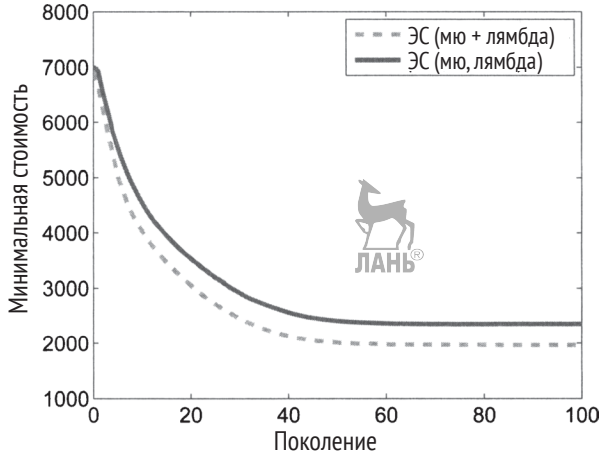


Рис. 6.11. Пример 6.2: скорость схождения эволюционной стратегии $(\mu + \lambda)$ и эволюционной стратегии (μ, λ) на эталоне Швефеля 2.26, усредненно по 100 симуляциям. Эволюционная стратегия $(\mu + \lambda)$ очевидным образом превосходит эволюционную стратегию (μ, λ)

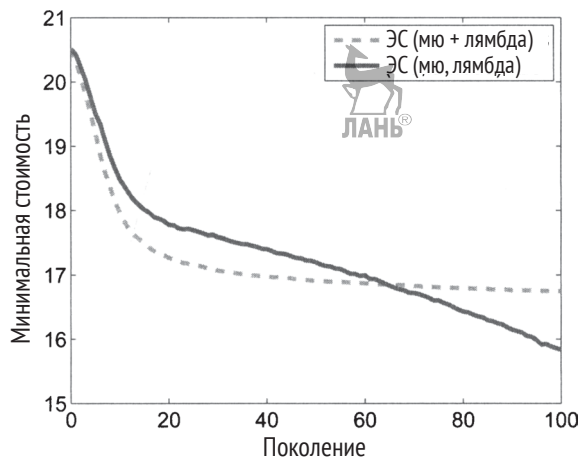


Рис. 6.12. Пример 6.2: скорость схождения эволюционной стратегии $(\mu + \lambda)$ и эволюционной стратегии (μ, λ) на эталоне Экли, усредненно по 100 симуляциям. Эволюционная стратегия $(\mu + \lambda)$ изначально превосходит эволюционную стратегию (μ, λ), но в конечном итоге эволюционная стратегия (μ, λ) не намного лучше эволюционной стратегии $(\mu + \lambda)$

Эволюционная стратегия (μ, κ, λ, ρ)

Напомним, что на рис. 6.10 каждый ребенок имеет двух родителей. Однако нет никаких причин ограничивать число родителей двумя. Вместо этого мы можем объединить более двух родителей, и мы используем ρ для обозначения числа родителей, которые вносят вклад в каждого ребенка. В конце раздела 6.3 мы обсудили некоторые многородительские операторы скрещивания, и мы обсудим дополнительные возможности в разделе 8.8.

Мы можем также установить максимальное время жизни для каждой особи в популяции, которое мы обозначаем как κ. Если максимальное время жизни – одно поколение, то $\kappa = 1$, и у нас будет эволюционная стратегия (μ, λ), потому что родителям не разрешается оставаться до следующего поколения. Если максимальное время жизни не ограничено, то $\kappa = \infty$, и у нас будет эволюционная стратегия $(\mu + \lambda)$, потому что нет ограничений на то, чтобы родители могли оставаться до следующего поколения; до тех пор, пока родитель является одним из μ наиболее приспособленных особей в объединенной популяции детей/родителей, он оставляется до следующего поколения, независимо от того, как долго он был в популяции. В общем случае мы, возможно, захотим ограничить продолжительность жизни особей эволюционной стратегии, для того чтобы предотвратить застой, в особенности для варьирующихся со временем задач (см. раздел 21.2).

Объединение этих двух обобщений приводит к эволюционной стратегии $(\mu, \kappa, \lambda, \rho)$ [Schwefel, 1995]. Популяция эволюционной стратегии $(\mu, \kappa, \lambda, \rho)$ имеет μ родителей, каждая особь имеет максимальное время жизни из κ поколений, и каждое поколение производит λ детей, каждый из которых имеет ρ родителей.

6.5. Самоадаптивные эволюционные стратегии

Эволюционно-стратегические алгоритмы, которые мы изучили, не дают нам много вариантов для корректировки стандартных отклонений σ_{kj} мутаций. Наш единственный вариант до сих пор был адаптивная эволюционная стратегия (1 + 1) на рис. 6.2, которая корректирует стандартные отклонения на основе уровня успешности мутации. Это может быть обобщено на эволюционно-стратегические алгоритмы $(1 + \lambda)$, исследуя все мутации λ в каждом поколении и отслеживая, сколько из них приводят к улучшениям. Однако нет четкого способа обобщить эту

идею на эволюционно-стратегические алгоритмы $(\mu + \lambda)$ или (μ, λ) при $\mu > 1$. Дети в этом случае состоят не только из мутаций, но и из комбинаций их родителей. Следовательно, возможно, не имеет смысла определять соответствующую скорость мутаций, сравнивая приспособленность ребенка с его родителями.

Вместе с тем так же, как мы мутируем признаки решения $\{x_i\}$ для $i \in [1, n]$ с целью поиска оптимального x, мы можем мутировать и элементы $\{\sigma_i\}$ вектора стандартного отклонения с целью поиска оптимального σ. После создания ребенка (x', σ') мы мутируем ребенка следующим образом [Schwefel, 1977], [Bäck, 1996, раздел 2.1.2]:

$$\sigma_i' \leftarrow \sigma_i' \exp(\tau' \rho_0 + \tau \rho_i)$$
$$x_i' \leftarrow x_i' + \sigma_i' r_i \tag{6.27}$$

для $i \in [1, n]$, где ρ_0, ρ_i и r_i – это скалярные случайные величины, взятые из $N(0, 1)$, а τ и τ' – регулировочные параметры. Составляющая $\tau' \rho_0$ допускает общее изменение скорости мутации x', а составляющие $\tau \rho_i$ допускают изменение скорости мутации конкретных элементов x'. Форма мутации σ' гарантирует, что σ' остается положительной.

Обратите внимание, что ρ_0 и ρ_i одинаково вероятно являются положительными, как и отрицательными. Это означает, что экспонента в уравнении (6.27) с равной вероятностью будет больше единицы, так же как и меньше единицы. Это, в свою очередь, означает, что σ_i' с такой же вероятностью увеличится, как и уменьшится. Швефель приходит к выводу, что этот мутационный подход устойчив к изменениям в τ и τ', но он предлагает устанавливать их следующим образом [Schwefel, 1977], [Bäck, 1996, раздел 2.1.2]:

$$\tau = P_1 \left(\sqrt{2\sqrt{n}} \right)^{-1}$$
$$\tau' = P_2 \left(\sqrt{2n} \right)^{-1}, \tag{6.28}$$

где n – это размерность задачи, а P_1 и P_2 – константы пропорциональности, которые, как правило, равны 1.

Важно реализовать мутацию в порядке, указанном уравнением (6.27), то есть σ' должна быть мутирована до того, как будет мутирована x'. Это связано с тем, что σ' должна использоваться для мутации x'. Благодаря этому приспособленность x' будет максимально точно указывать на уместность σ'. Эти идеи приводят к самоадаптивным эволюционным стратегиям $(\mu + \lambda)$ и (μ, λ), показанным на рис. 6.13. Обратите внимание,

что рис. 6.13 называется *самоадаптивной* эволюционной стратегией, в отличие от более простой идеи адаптивной эволюционной стратегии на рис. 6.2. Самоадаптивная эволюционная стратегия, представленная в публикации [Rechenberg, 1973], является, пожалуй, наиважнейшим вкладом в научные исследования и практику применения эволюционных алгоритмов. Сегодня практически во всех эволюционных алгоритмах используются некоторые виды самоадаптации для корректировки алгоритмических регулировочных параметров. Кроме того, в публикации [Beyer и Deb, 2001] показано, что даже эволюционные алгоритмы без явной самоадаптации могут проявлять самоадаптивное поведение. Интерпретация эволюционных алгоритмов как самоадаптивных и эффект самоадаптивного поведения на результативность эволюционного алгоритма остаются важнейшей задачей будущих исследований.

Инициализировать константы τ и τ', как показано в уравнении (6.28).
$\{(x_k, \sigma_k)\} \leftarrow$ случайно сгенерированные особи, $k \in [1, \mu]$.
Каждое x_k – это кандидатное решение, и каждый σ_k – вектор стандартного отклонения.
Отметить, что $x_k \in R^n$ и $\sigma_k \in R^n$.
While not (критерий останова)
  For $k = 1, \cdots, \lambda$
    Случайно отобрать двух родителей из $\{(x_k, \sigma_k)\}$.
    Применить метод рекомбинации для объединения двух родителей и получения ребенка, который обозначается как (x'_k, σ'_k).
    Сгенерировать случайный скаляр ρ_0 из $N(0, 1)$.
    Сгенерировать случайный вектор $[\rho_0 \ldots \rho_n]$ из $N(0, I)$.
    $\sigma'_{ki} \leftarrow \sigma'_{ki} \exp(\tau'\rho_0 + \tau\rho_i)$ для $i \in [1, n]$
    $\Sigma'_k \leftarrow \text{diag}((\sigma'_{k1})^2, \ldots, (\sigma'_{kn})^2) \in R^{n \times n}$
    Сгенерировать случайный вектор r из $N(0, \Sigma'_k)$.
    $x'_k \leftarrow x'_k + r$
  Next k
  If – это эволюционная стратегия $(\mu + \lambda)$ **then**
    $\{(x_k, \sigma_k)\} \leftarrow$ лучшие μ особей из $\{(x_k, \sigma_k)\} \cup \{(x'_k, \sigma'_k)\}$
  else – это эволюционная стратегия (μ, λ) **then**
    $\{(x_k, \sigma_k)\} \leftarrow$ лучшие μ особей из $\{(x'_k, \sigma'_k)\}$
  End if
Next поколение

Рис. 6.13. Приведенный выше псевдокод дает схематичное описание самоадаптивных эволюционных стратегий $(\mu + \lambda)$ и (μ, λ), где n – это размерность задачи

Алгоритм на рис. 6.13 состоит в том, что мутационная ковариационная матрица $\sum_k'$ – диагональная. В общем случае для генерации мутационного вектора r мы можем использовать недиагональную ковариацию. Затем мы можем попытаться оптимизировать всю ковариационную матрицу в целом, а не только диагональные элементы [Bäck, 1996, раздел 2.1].

■ Пример 6.3

В данном примере мы используем эволюционную стратегию $(\mu + \lambda)$ для оптимизации эталонной функции Экли. Мы сравниваем стандартную эволюционную стратегию $(\mu + \lambda)$, показанную на рис. 6.10, и самоадаптивную эволюционную стратегию $(\mu + \lambda)$, показанную на рис. 6.13. Мы применяем оба алгоритма с $\mu = 10$, $\lambda = 20$, дискретным половым скрещиванием и размерностью задачи $n = 20$. Мы используем стандартные значения для τ и τ', показанные в уравнении (6.28), и стандартные значения $P_1 = P_2 = 1$. На рис. 6.14 сравниваются средние скорости схождения стандартной эволюционной стратегии $(\mu + \lambda)$ и самоадаптивной эволюционной стратегии $(\mu + \lambda)$. Мы видим, что самоадаптивная эволюционная стратегия сходится гораздо быстрее, чем стандартная. На рис. 6.15 показаны значения стандартного отклонения σ_{ki}, $i \in [1, 20]$ для лучшей особи (x_k, σ_k) в популяции за последнее поколение. На рис. 6.15 показано, что стандартные отклонения эволюционировали по-разному для разных чисел признаков. Они эволюционировали так, чтобы попытаться оптимизировать эффективность мутаций.

■

■ Пример 6.4

Данный пример аналогичен примеру 6.3, за исключением использования эволюционной стратегии (μ, λ) и эталонной функции Гриванка. На рис. 6.16 сравниваются средние скорости схождения стандартной эволюционной стратегии (μ, λ) и самоадаптивной эволюционной стратегии (μ, λ). Мы видим, что самоадаптивная эволюционная стратегия показывает очень слабую результативность. Причина этого в том, что, хотя мутация σ' в уравнении (6.27) имеет медиану 1, то есть что σ' равно вероятно увеличится, как и уменьшается, мутация σ' имеет среднее значение, которое больше 1. Аргументом экспоненциальной функции в уравнении (6.27) является сумма двух нулевых средних гауссовых случайных величин. Для простоты предположим, что аргумент x экспо-

ненциальной функции $\exp(x)$ является гауссовой случайной величиной с нулевым средним значением и дисперсией 1. Тогда экспонента имеет медиану 1, но при этом она имеет среднее значение

$$\begin{aligned} E[\exp(x)] &= \int_{-\infty}^{\infty} \text{PDF(x)} \exp(x)\, dx \\ &= \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}} \exp(-x^2/2) \exp(x)\, dx \\ &= \exp(1/2) \approx 1.65. \end{aligned} \tag{6.29}$$

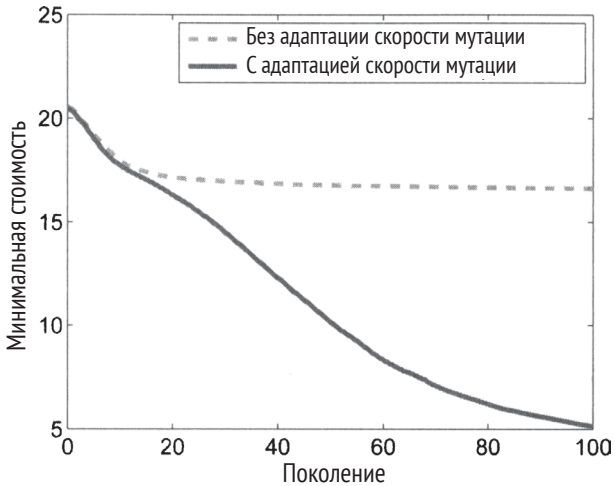


Рис. 6.14. Пример 6.3: схождение алгоритмов стандартной и самоадаптивной эволюционных стратегий (μ, λ) на 20-мерной функции Экли, усредненно по 100 симуляциям. Самоадаптивная эволюционная стратегия, которая автоматически корректирует мутационные стандартные отклонения, сходится гораздо быстрее, чем стандартная эволюционная стратегия

Мы видим, что мутация σ' имеет тенденцию к увеличению σ' больше, чем к уменьшению σ'. Это может привести к преобладанию больших значений σ' в ребенке. Если все родители отбрасываются в конце каждого поколения, как и в эволюционной стратегии (μ, λ), неприемлемо большие значения σ' могут увековечиться в популяции и привести к слабой результативности.

■

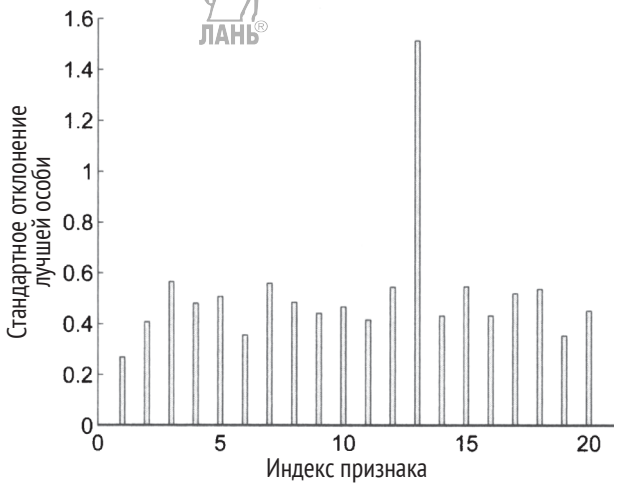


Рис. 6.15. Пример 6.3: значения стандартного отклонения лучшей особи в последнем поколении для самоадаптивной эволюционной стратегии применительно к 20-мерной функции Экли. 20-мерной оптимизационной задаче соответствует 20 стандартных отклонений. Самоадаптивная эволюционная стратегия стремится корректировать мутационные стандартные отклонения таким образом, чтобы максимизировать эффективность мутаций

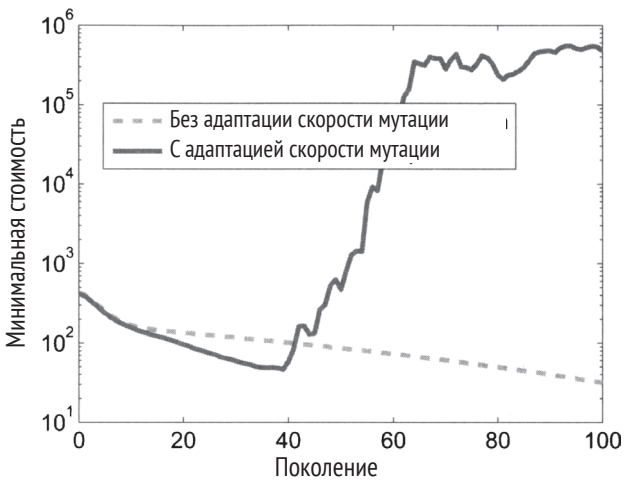


Рис. 6.16. Пример 6.4: скорость схождения подходов стандартной и самоадаптивной эволюционных стратегий (μ, λ) на 20-мерной функции Гриванка, усредненно по 100 симуляциям. Самоадаптивная эволюционная стратегия, которая автоматически корректирует мутационные стандартные отклонения, показывает очень слабые результаты

Ковариационно-матричная адаптация

Одним из успешных вариаций эволюционной стратегии является эволюционная стратегия CMA-ES, где CMA является аббревиатурой ковариационно-матричной адаптации (covariance matrix adaptation) [Hansen, 2010]. Цель ковариационно-матричной адаптивной эволюционной стратегии, которая показала большой успех на эталонных функциях, заключается в том, чтобы (как можно лучше) выполнять подгонку распределения мутаций эволюционной стратегии к контуру целевой функции. Эта предпринятая подгонка вполне может быть успешной только для квадратичных целевых функций, но многие целевые функции могут быть аппроксимированы квадратичной функцией вблизи их оптимума. Недостатками ковариационно-матричной адаптивной эволюционной стратегии являются сложная стратегия адаптации и сложные настройки регулировочных параметров. Здесь мы приводим упрощенную версию ковариационно-матричной адаптивной эволюционной стратегии, которая называется ковариационно-матричной *само*адаптивной эволюционной стратегией (covariance matrix self-adaptive ES, CMSA-ES) [Beyer и Sendhoff, 2008]; идея эволюционной стратегии CMSA – заучить форму поискового пространства в процессе эволюции и адаптировать мутационную дисперсию. Рисунок 6.17 дает схематичное описание ковариационно-матричной самоадаптивной эволюционной стратегии (CMSA).

На рис. 6.17 τ – это параметр самообучения, определяющий адаптационную скорость значений σ_k. Значения σ_k определяют силу мутации. Обратите внимание, что каждый σ_k – это скаляр, который контрастирует с самоадаптивной эволюционной стратегией рис. 6.13. Временна́я константа τ_C определяет адаптационную скорость ковариационной матрицы C, которая управляет относительными величинами и корреляциями мутаций вдоль каждой размерности. Рекомендуемые значения для τ и τ_C следующие [Beyer и Sendhoff, 2008]:

$$\tau \leftarrow 1 / \sqrt{2n}$$
$$\tau_C \leftarrow 1 + n\,(n+1) / (2\mu) \qquad (6.30)$$

хотя более эффективная результативность могла бы быть получена с варьирующимися со временем значениями τ и τ_C.

Квадратный корень $\sqrt{C}$ на рис. 6.17 можно рассчитать несколькими способами. В исходной эволюционной стратегии CMA использовалось спектральное разложение, или разложение на собственные значения [Hansen и Ostermeier, 2001], а в эволюционной стратегии CMSA используется разложение Холецкого [Beyer и Sendhoff, 2008].

Инициализировать константы τ и τ_C.
$C \leftarrow I$ = единичная $n \times n$-матрица.
$\{(x_k, \sigma_k)\} \leftarrow$ случайно сгенерированные особи, $k \in [1, \mu]$.
Каждое x_k – это кандидатное решение, и каждый σ_k – вектор стандартного отклонения.
Отметить, что $x_k \in R^n$ и $\sigma_k \in R^n$.
While not (критерий останова)
  $\overline{\sigma} \leftarrow \sum_{k=1}^{\mu} \sigma_k / \mu$
  $\overline{x} \leftarrow \sum_{k=1}^{\mu} x_k / \mu$
  For $k = 1, \cdots, \lambda$
    $r \leftarrow N(0, 1)$ = гауссов случайный скаляр
    $\sigma_k \leftarrow \overline{\sigma} \exp(r\, r)$
    $R \leftarrow N(0, I)$ = n-мерный гауссов случайный вектор
    $s_k \leftarrow \sqrt{C} R$
    $z_k \leftarrow \sigma_k s_k$
    $x_k \leftarrow \overline{x} + z_k$
  Next k
  $\hat{S} \leftarrow \sum_{k=1}^{\lambda} s_k s_k^{\mathrm{T}} / \lambda$
  $C \leftarrow (1 - 1/\tau_C) C + \hat{S}/\tau_C$
Next поколение

Рис. 6.17. Приведенный выше псевдокод дает схематичное описание ковариационно-матричной самоадаптивной эволюционной стратегии (CMSA-ES), где n – это размерность задачи. Подробности см. в тексте

На рис. 6.17 средние значения $\overline{\sigma}$, $\overline{x}$ и $\hat{S}$ показаны как простые средние. Вместе с тем мы можем также вычислить их как средневзвешенные значения, имея в виду, что можем придать больше веса более приспособленным особям. В связи с хорошим компромиссом между простотой и эффективностью на эталонных задачах эволюционная стратегия CMSA, похоже, имеет ряд перспектив для будущих исследований, включая гибридизацию с другими эволюционными алгоритмами.

6.6. Заключение

Мы обсудили оригинальную эволюционную стратегию (1 + 1), более общую эволюционную стратегию (μ + 1) и еще более общую эволюционную стратегию (μ + λ). Мы также обсудили эволюционную стратегию (μ, λ). Мы видим, что эволюционная стратегия похожа на генетический алгоритм, но генетические алгоритмы изначально разрабатывались

путем кодирования кандидатных решений в виде битовых строк, в то время как эволюционная стратегия всегда работала с непрерывными параметрами. Несмотря на то что генетические алгоритмы часто проектируются для работы на непрерывных параметрах, это приводит к философскому различию между двумя алгоритмами: эволюционная стратегия имеет тенденцию работать на представлениях, которые ближе к постановке задачи, в то время как генетические алгоритмы, как правило, работают на представлениях, которые удалены от исходной постановки задачи дальше. Еще одно различие между двумя алгоритмами заключается в том, что генетический алгоритм подчеркивает рекомбинацию, а эволюционная стратегия – мутацию. Это может направлять наш выбор алгоритмов, когда мы сталкиваемся с конкретной задачей оптимизации. Если в конкретной задаче более важно разведывание, чем эксплуатация, то мы, возможно, захотим применить эволюционную стратегию. Однако если более важна эксплуатация, то мы, возможно, захотим применить генетический алгоритм[3].

Во всех вариациях эволюционных стратегий, которые мы обсуждали до сих пор, механизм отбора является детерминированным. То есть для следующего поколения отбираются лучшие μ особей. В качестве вариации этого подхода вместо этого можно выполнить вероятностный отбор особей для следующего поколения. Например, в эволюционной стратегии ($\mu + \lambda$) мы можем применить рулеточный отбор, для того чтобы вероятностно отобрать родителей следующего поколения, где каждый сектор колеса рулетки пропорционален приспособленности соответствующей особи. Работа в этом направлении остается перспективной.

Дополнительные материалы по эволюционным стратегиям представлены в публикациях [Bäck и Schwefel, 1993] и [Beyer, 2010]. Марковская модель эволюционной стратегии представлена в публикации [Francpis, 1998]. Эволюционные стратегии для многокритериальных задач (см. главу 20) обсуждаются в публикации [Rudolph и Schwefel, 2008].

Задачи

Письменные упражнения

6.1. В заключении к настоящей главе говорится, что эволюционная стратегия, возможно, будет более подходить для решения задачи, требующей разведывания, а генетический алгоритм, вероятно,

[3] Обсуждение вопросов разведывания и эксплуатации см. в разделе 2.7.5.

будет более подходить для решения задачи, требующей эксплуатации. В каком типе задачи более желательным будет разведывание, и в каком типе задачи более желательной будет эксплуатация?

6.2. Покажите, что вторая и третья строки уравнения (6.9) эквивалентны.

6.3. Примените уравнение (6.16) для получения уравнения (6.17).

6.4. Предположим, что мы применяем эволюционную стратегию (1 + 1) для минимизации одномерной сферической функции. Какова вероятность того, что мутация со стандартным отклонением σ улучшит кандидатное решение x_0?

6.5. Уравнение (6.27) показывает, что стандартное отклонение мутации самоадаптивной эволюционной стратегии меняется на экспоненциальный множитель. Уравнение (6.29) показывает, что среднее значение этого множителя больше 1, что может привести к слабой результативности эволюционной стратегии. Как изменить уравнение (6.27) так, чтобы среднее значение множителя равнялось 1?

Компьютерные упражнения

6.6. Просимулируйте адаптивную эволюционную стратегию (1 + 1) на рис. 6.2 для минимизации 10-мерной сферической функции (см. раздел С.1.1) на области [–5.12, +5.12]. Инициализируйте стандартное отклонение мутации каждой размерности до $0.1/(2\sqrt{3})$. Выполните симуляцию для 500 поколений и запишите стоимость каждого поколения. Выполните 50 подобных симуляций и усредните 50 значений стоимости в каждом поколении. Постройте график средней стоимости как функции номера поколения. Сделайте это для c = 0.6, 0.8 и 1.0. Какое значение c дает лучшую результативность?

6.7. Повторите задачу 6.6, но вместо симулирования для трех разных значений c используйте значение c, принятое по умолчанию, и выполните симуляцию для трех разных пороговых соотношений успешности мутации ϕ_{thresh} – то есть, вместо того чтобы использовать значение по умолчанию $\phi_{\text{thresh}} = 1/5$, используйте c = 0.01, 0.2 и 0.4. Какое значение ϕ_{thresh} дает лучшую результативность?

6.8. Постройте график двумерной коридорной функции на области [−50, +50] с константами $c_0 = 0$, $c_1 = 1$ и $b = 10$.

6.9. Примените эволюционную стратегию ($\mu + \lambda$) для минимизации 10-мерной сферической функции на области [−5.12, +5.12] с $\mu = 10$ и $\lambda = 20$. Установите стандартное отклонение мутации каждой размерности равным $0.1/(2\sqrt{3})$. Просимулируйте для 100 поколений и запишите минимальную стоимость для каждого поколения. Выполните 50 подобных симуляций и усредните 50 минимальных значений стоимости в каждом поколении. Постройте график значений средней минимальной стоимости как функции номера поколения. Выполните это для дискретного полового, дискретного глобального, промежуточного полового и промежуточного глобального скрещиваний. Какой тип скрещивания дает лучшую результативность?

6.10. Напомним, что уравнение (6.29) показало, что если $x \sim N(0,1)$, то $\exp(x)$ имеет медиану, равную 1, и среднее значение – около 1,65. Уравнение (6.27) показывает, что стандартное отклонение мутации самоадаптивной эволюционной стратегии меняется на экспоненциальный множитель. Численно аппроксимируйте медиану и среднее значение экспоненциального множителя, если $n = 10$. Как увеличение n влияет на медиану и среднее значение экспоненциального множителя?

6.11. P_1 и P_2 в уравнении (6.28) по умолчанию имеют значения 1, но, возможно, другие значения дадут более высокую результативность. Примените самоадаптивную эволюционную стратегию ($\mu + \lambda$) для минимизации 10-мерной сферической функции на области [−5.12, −1−5.12] с $\mu = 10$ и $\lambda = 20$. Инициализируйте стандартное отклонение мутации каждой размерности значением $0.1/(2\sqrt{3})$. Просимулируйте для 100 поколений и запишите минимальную стоимость каждого поколения. Выполните 50 подобных симуляций и усредните 50 минимальных значений стоимости в каждом поколении. Постройте график средней минимальной стоимости как функции номера поколения. Выполните это для $P_1 = P_2 = 0{,}1$, $P_1 = P_2 = 1$ и $P_1 = P_2 = 10$. Какое значение P_1 и P_2 дает лучшую результативность?

Глава 7

Генетическое программирование

Машины были бы полезнее, если бы они могли учиться выполнять задачи, для которых им не были даны точные методы.

– Ричард Фридберг [Friedberg, 1958]

Генетические алгоритмы и аналогичные им эволюционные алгоритмы являются мощными оптимизационными методами, но у них есть неотъемлемое ограничение: они встраивают предполагаемую структуру решения в представление своих кандидатных решений. Например, если мы хотим применить генетический алгоритм для решения непрерывной оптимизационной задачи с 10 переменными, то хромосома генетического алгоритма обычно представляется как $(x_1, x_2, \ldots, x_{10})$. В этом заключается как преимущество, так и недостаток генетических алгоритмов. Преимущество состоит в том, что это позволяет инженеру закодировать специфичную для конкретной задачи информацию в представление решения. Если мы знаем, что наше целевое решение может быть хорошо представлено 10 вещественными параметрами, то определение хромосомы как $(x_1, x_2, \ldots, x_{10})$ имеет большой смысл. Однако мы можем не знать, какие параметры в данной задаче необходимо оптимизировать. Кроме того, мы можем не знать структуру параметров, которые нужно оптимизировать. Являются ли параметры действительными числами, либо конечными автоматами, либо компьютерными программами, либо временны́ми планами-графиками, либо чем-то еще?

Генетическое программирование (genetic programming, GP) – это попытка обобщить эволюционные стратегии до алгоритма, который способен заучить не только лучшее решение задачи при наличии специфической структуры, но и оптимальную структуру. Генетическое программирование эволюционно формирует компьютерные программы для решения оптимизационных задач. Это отличительная особенность генетического программирования, по сравнению с другими эволюционными алгоритмами; другие эволюционные алгоритмы эволюционно формируют решения, в то время как генетическое программирование эволюционно формирует программы, которые могут вычислять решения. Фактически такая формулировка была одной из первоначальных целей сообщества искусственного интеллекта. Артур Сэмюэль (Arthur Samuel), один из ранних американских первопроходцев в области искусственного интеллекта, в 1959 году писал: «Программирование компьютеров так, чтобы они учились на опыте, должно в конечном итоге устранить необходимость в значительной части всех этих попыток дотошного программирования».

Фундаментальные особенности генетического программрования могут быть обобщены в трех основных принципах [Koza, 1992, глава 2]. Во-первых, генетическое программирование, которое эволюционно формирует компьютерные программы, дает нам гибкость в получении методов решения широкого спектра задач. Многие инженерные задачи могут быть решены с помощью структур, которые организованы как компьютерная программа, дерево решений или сетевая архитектура. Во-вторых, генетическое программирование не ограничивает свои решения почти так же, как и другие эволюционные алгоритмы; эволюционировавшие программы могут свободно принимать размер, форму и структуру, которые лучше всего подходят для рассматриваемой задачи. В-третьих, генетическое программирование эволюционно формирует компьютерные программы с использованием индукции. Это одновременно и сила, и слабость. Генетическое программирование не формирует программы, строя их, как это делал бы человек, дедуктивно и логически. Так или иначе, некоторые задачи не поддаются дедукции. Если мы хотим написать компьютерную программу на основе множества тренировочных образцов, то это будет сложно сделать стандартными методами компьютерного программирования. Но так работает генетическое программирование, как, впрочем, и другие эволюционные алгоритмы. Генетическое программирование индуктивно конструирует оптимальные компьютерные программы.

Ранние результаты генетического программирования

Алан Тьюринг, один из родоначальников информатики и искусственного интеллекта, предвидел нечто вроде генетического программирования, когда в 1950 году написал в известной статье: «мы не можем рассчитывать, что с первой попытки найдем хорошую детскую машину. Нужно поэкспериментировать с обучением одной такой машины и посмотреть, насколько хорошо она учится. Затем можно попробовать другую и посмотреть, будет ли она лучше или хуже» [Turing, 1950, стр. 456]. Ричард Фридберг, который изучал компьютерный интеллект в 1950-х годах, а затем продолжил карьеру в медицине, был одним из первых, кто работал над задачами, которые можно было бы классифицировать как генетическое программирование. Фридберг писал компьютерные программы, которые могли эволюционно формировать другие компьютерные программы, которые могли затем сами решать задачи. Его работа была опубликована в конце 1950-х годов под названием «Обучающаяся машина» [Friedberg, 1958], [Friedberg и соавт., 1958]. В своей работе он принял ряд укороченных методов из-за ограниченных вычислительных возможностей того времени. Например, он сгруппировал подобные программы вместе и предположил, что их приспособленность коррелирует, и поэтому он может уменьшить число оцениваний приспособленности. Это было удивительно прозорливым предвестником многих методов сокращения вычислений приспособленности, которые мы видим сегодня в эволюционных алгоритмах (см. главу 21). С 1960 по 2010 год вычислительная мощность (скорость и память) увеличилась примерно в один миллион раз в течение 50 лет, но сегодня мы по-прежнему так же озабочены вычислительной мощностью, как и Фридберг в 1950-х.

Предшественником современного генетического программирования был генетический алгоритм переменной длины (variable-length GA), разработанный Стивеном Смитом в своей докторской диссертации 1980 года [Smith, 1980], в которой каждая особь в популяции генетического алгоритма представляла собой множество правил принятия решений. Еще одной ранней работой, которая стала предвестницей сегодняшнего генетического программирования, было исследование Ричарда Форсайта в 1981 году, в котором эволюционно формировались правила классификации шаблонов [Forsyth, 1981]. Никола Крамер в 1985 году написал, возможно, самую первую недвусмысленную работу по генетическому программированию [Cramer, 1985], которая частично

была основана на диссертации Смита. В 1990 году Уго де Гарис применил термин «генетическое программирование» для обозначения оптимизации нейронных сетей с использованием генетических алгоритмов [de Garis, 1990], но его применение этого термина с тех пор было заменено сегодняшней формулировкой, которая определяет генетическое программирование как эволюционное формирование компьютерных программ. Книга Джона Коза 1992 года, которая по-прежнему является отличным введением в эту тему, сыграла важную роль в популяризации исследований в области генетического программирования [Koza, 1992]. Дж. Коза с тех пор написал три книги по генетическому программированию [Koza, 1994], [Koza и соавт., 1999], [Koza и соавт., 2005], в которых обсуждаются вопросы практического применения и более продвинутые аспекты генетического программирования. Обсуждение ранней истории генетического программирования можно найти в публикации [Koza, 2010].

Краткий обзор главы

Мы начинаем эту главу с предварительного обсуждения языка программирования Lisp в разделе 7.1. Lisp часто используется для генетического программирования, потому что структура языка Lisp поддается таким операциям эволюционных алгоритмов, как скрещивание и мутация. В разделе 7.2 дается обзор основных понятий генетического программирования, включая некоторые проектные решения, которые мы должны сделать. В разделе 7.3 рассматривается пример генетической программы для управления объектом за минимальное время. В разделе 7.4 обсуждается вопрос раздувания генетической программы, то есть тенденции решений на основе генетического программирования, заключающейся в неконтролируемом увеличении размера программ. В разделе 7.5 рассматривается использование генетического программирования для эволюционного формирования других решений, помимо компьютерных программ, включая электрические схемы и другие инженерные проекты. В разделе 7.6 обсуждаются некоторые способы, которыми результативность генетического программирования может быть смоделирована математически, в особенности используя теорию схем; этот раздел может быть пропущен читателем, который хочет получить лишь оперативные знания практики применения генетического программирования. В разделе 7.7 дается обобщение данной главы и высказаны предложения для будущих исследований в области генетического программирования.

7.1. Lisp: язык генетического программирования

Генетическое программирование часто реализуется на компьютерном языке Lisp, потому что структура данного языка очень хорошо увязывается с генетическим скрещиванием и мутацией компьютерных программ. В этом разделе содержится краткий обзор языка Lisp и концептуальное описание того, как программы на языке Lisp могут быть объединены для создания новых программ.

Эволюционное формирование компьютерных программ является чрезвычайно сложной задачей, потому что программы обычно не представлены таким образом, который делает мутацию и скрещивание возможным. Это основное препятствие, которое нам нужно преодолеть на пути к эволюционному формированию компьютерных программ. Например, рассмотрим две стандартные программы на языке MATLAB:

Программа 1

```
if x < 1
      z = [1, 2, 3, 4, 5];
else
      for i = 1 : 5
            z(i) = i/x;
      end
end
```

Программа 2

```
for i = 1 : 10
      if x > 5
            z(i)=xi;
      else
            z(i) = x/i;
      end
end
```

Как выполнить скрещивание на этих программах? Операция скрещивания, которая тщательно не анализирует синтаксис, приведет к недопустимой программе (то есть программе, которая не запускается или даже не компилируется). Например, если мы заменим первые две строки в левой приведенной выше программе первыми двумя строками из правой программы, то получим следующее:

Дочерняя программа

```
for i = 1 : 10
      if x > 5
else
      for i = 1 : 5
            z(i) = i/x;
      end
end
```

Вышеприведенная дочерняя программа явно не является допустимой. Многие операции скрещивания и мутации, которые мы выполняем в программах на MATLAB, приведут к недопустимым программам. То же самое можно сказать и о программах, написанных на большинстве других популярных языков (C, Java, Fortran, Basic, Perl, Python и т. д.). Все эти языки имеют схожие структуры, поэтому ни одна из них не может быть легко подвергнута мутации или скрещиванию с другими программами.

Однако есть один язык, который подходит для скрещивания и мутации. Этот язык называется Lisp [Winston и Horn, 1989]. Язык Lisp, изобретенный в 1958 году, вероятно, является вторым старейшим языком программирования, лишь на один год новее языка Fortran. Lisp – это аббревиатура от англ. «list processing» (обработка списков). Связные списки являются одной из основных структур языка Lisp, и в ранние годы это сделало его подходящим вариантом для приложений искусственного интеллекта (то есть экспертных систем и их цепочек правил вывода). Язык Lisp перестал быть особо популярным, потому что он отличается от других языков. Тем не менее он стал популярным среди исследователей и практиков в области генетического программирования из-за его приспособленности для операций скрещивания и мутации.

Программный код на языке Lisp пишется в круглых скобках, при этом после имени функции следуют ее аргументы. Например, следующий программный код складывает x и 3:

$$(+\ x\ 3).$$

Это пример префиксной нотации, поскольку математический оператор предшествует входным данным. Скобочное выражение в Lisp также называется S-выражением и является сокращением термина «символьное выражение» (symbolic expression). Все S-выражения можно рассматривать как функции, возвращающие значение, которое они вычисляют. S-выражения, вычисляющие несколько значений, возвращают последнее вычисляемое значение. Выражение $(+\ x\ 3)$ не только складывает x и 3, но и возвращает $x + 3$ на следующий, более высокий уровень выполнения функции.

Приведем еще несколько примеров. Следующий ниже программный код вычисляет косинус для $(x + 3)$:

$$(\cos(+\ x\ 3)).$$

Следующий далее программный код вычисляет минимум из $\cos(x + 3)$ и $z/14$:

(min (cos(+ *x* 3)) (/ *z* 14).

Следующий ниже программный код копирует значение *y* в *x*, если *z* > 4:

(if (> *z* 4) (setf *x y*)).

Обратите внимание, что S-выражения похожи на множества в том, что S-выражения могут содержать другие S-выражения. В приведенном выше S-выражении (> *z* 4) и (setf *x y*) оба являются S-выражениями, которые являются частью S-выражения более высокого уровня (if (> *z* 4) (setf *x y*)).

Причина, по которой программный код на языке Lisp поддается эволюционному изменению, заключается в том, что S-выражения непосредственно соответствуют древовидным структурам, так называемым синтаксическим деревьям. Например, S-выражение языка Lisp для вычисления *xy* + |*z*| может быть записано следующим образом:

(+ (* *x y*) (abs *z*)).

Это S-выражение может быть представлено синтаксическим деревом, показанным на рис. 7.1. Мы выполняем интерпретацию синтаксического дерева, подобно тому, как показано на рис. 7.1, продвигаясь снизу вверх. На рис. 7.1 показано, что *x* и *y* находятся в нижней части дерева и соединены друг с другом оператором умножения. Это дает нам выражение *xy*, или S-выражение (* *x y*). Мы видим, что данное под-S-выражение соответствует поддереву на рис. 7.1. Символы, которые появляются в нижней части синтаксического дерева (например, *x*, *y* и *z* на рис. 7.1), называются листьями.

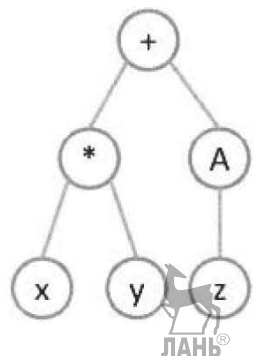


Рис. 7.1. Синтаксическое дерево для функции *xy* + |*z*|, которая представлена в виде S-выражения (+ (* *x y*) (abs *z*)). Узел «A» представляет оператор абсолютной величины

На рис. 7.1 показано, что z находится в нижней части дерева и на нем оперирует функция абсолютной величины. Это дает нам выражение $|z|$, или S-выражение (abs 2). Мы снова видим, что под-S-выражение соответствует поддереву на рис. 7.1.

Наконец, рис. 7.1 показывает, что xy и $|z|$ встречаются в узле сложения в верхней части дерева. Это дает нам выражение $xy + |z|$, или S-выражение (+(* x y) (abs z)). Мы видим, что это высокоуровневое S-выражение соответствует всей древовидной структуре на рис. 7.1.

В еще одном примере рассмотрим функцию, которая возвращает $(x + y)$, если $t > 5$, и $(x + 2 + z)$ в противном случае:

```
if t > 5
    return (x + y)
else
    return (x + 2 + z)
End
```

В нотации Lisp данная функция может быть записана как:

$$(\text{if } (> t\ 5)\ (+ x\ y)\ (+ x\ 2\ z)).$$

На рис. 7.2 показано синтаксическое дерево для этой функции. Обратите внимание, что многие функции Lisp, такие как функция сложения в приведенном выше примере, могут принимать переменное число аргументов.

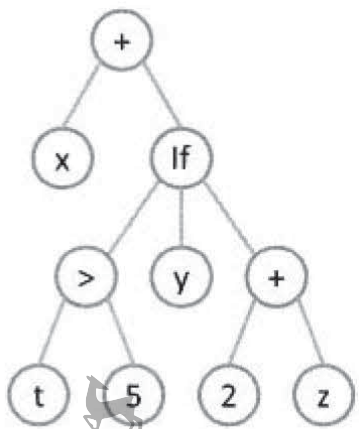


Рис. 7.2. Синтаксическое дерево для функции:
«если t > 5, то вернуть (x + y), иначе вернуть (x + 2 + z)»

Скрещивание с помощью программ на языке Lisp

Соответствие между S-выражениями и поддеревьями делает выполнение таких операций, как скрещивание и мутация, в компьютерных программах на языке Lisp концептуально прямолинейным. Например, рассмотрим следующие ниже функции:

Родитель 1: $xy + |z| \Rightarrow$ (+ (* x y)) abs z)

Родитель 2: $(x + z)x - (z + y/x) \Rightarrow$ (– (* (+ x z) x) (+ z (/ y x))) (7.1)

Эти две родительские функции показаны на рис. 7.3. Мы можем создать две дочерние функции, случайно отбирая точку скрещивания в каждом родителе и меняя местами поддеревья, которые лежат ниже этих точек. Например, предположим, что в качестве точек скрещивания мы выбираем узел умножения в родителе 1 и второй узел сложения в родителе 2. На рис. 7.3 показано, как поддеревья в родителях, которые находятся ниже этих точек, можно поменять местами для создания дочерних функций. Дочерние функции, созданные таким образом, всегда являются допустимыми синтаксическими деревьями.

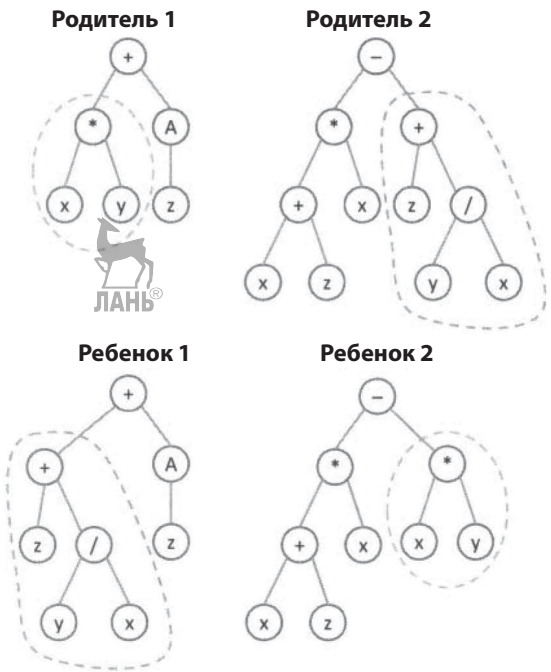


Рис. 7.3. Два синтаксических дерева (родители) скрещиваются и производят два новых синтаксических дерева (детей). Выполненное таким образом скрещивание всегда приводит к допустимым дочерним синтаксическим деревьям

Теперь рассмотрим скрещивание S-выражений уравнения (7.1). Следующее ниже уравнение подчеркивает скобочные пары, соответствующие поддеревьям рис. 7.3, и показывает, как замена скобочных пар в двух исходных S-выражениях (родительских выражениях) создает новые S-выражения (дочерние выражения):

$$\left.\begin{array}{r}(+\ [*\ \boldsymbol{x}\ \boldsymbol{y}])\ \text{abs}\ z) \\ (-\ (*\ (+\ x\ z)\ x)\ \{+\ z\ (/\ y\ x)\})\end{array}\right\} \Rightarrow \left\{\begin{array}{l}(+\ \{+\ \boldsymbol{z}\ \boldsymbol{(/\ y\ x)}\})\ \text{abs}\ z) \\ (-\ (*\ (+\ x\ z)\ x)\ [\ *\ \boldsymbol{x}\ \boldsymbol{y}])\end{array}\right., \tag{7.2}$$

где скрещивающиеся S-выражения выделены жирным шрифтом. Это называется древовидным скрещиванием. Любое S-выражение в синтаксическом дереве может быть заменено любым другим S-выражением, и синтаксическое дерево останется допустимым. Это то, что делает язык Lisp идеальным языком для генетического программирования. Если мы хотим выполнить скрещивание между двумя программами на языке Lisp, то просто находим случайную левую скобку в родителе 1, а затем находим подходящую правую скобку; текст между этими двумя скобками образует допустимое S-выражение. Точно так же мы находим случайную левую скобку в родителе 2, затем находим подходящую правую скобку и получаем еще одно S-выражение. После того как мы поменяем местами два S-выражения, у нас будет двое детей. Мы можем выполнить мутацию аналогичным образом, заменив случайно выбранное S-выражение случайным S-выражением, и такая мутация называется древовидной мутацией.

Рисунок 7.4 показывает дополнительный пример скрещивания. У нас те же два родителя, что и на рис. 7.3, но мы случайно отбираем узел 0 в качестве точки скрещивания в родителе 1 и узел деления в качестве точки скрещивания в родителе 2. Операция скрещивания рис. 7.4 представлена в записи в виде S-выражения следующим образом (где, как и прежде, скрещивающиеся S-выражения показаны полужирным шрифтом):

$$\left.\begin{array}{r}(+\ (*\ x\ y))\ \text{abs}\ \boldsymbol{z}) \\ (-\ (*\ (+\ x\ z)\ x)\ (+\ z\ [\boldsymbol{/\ y\ x}]))\end{array}\right\} \Rightarrow \left\{\begin{array}{l}(+\ (\ *\ x\ y))\ \text{abs}\ [\boldsymbol{/\ y\ x}]) \\ (-\ (*\ (+\ x\ z)\ x)\ (+\ z\ \boldsymbol{x}))\end{array}\right.. \tag{7.3}$$

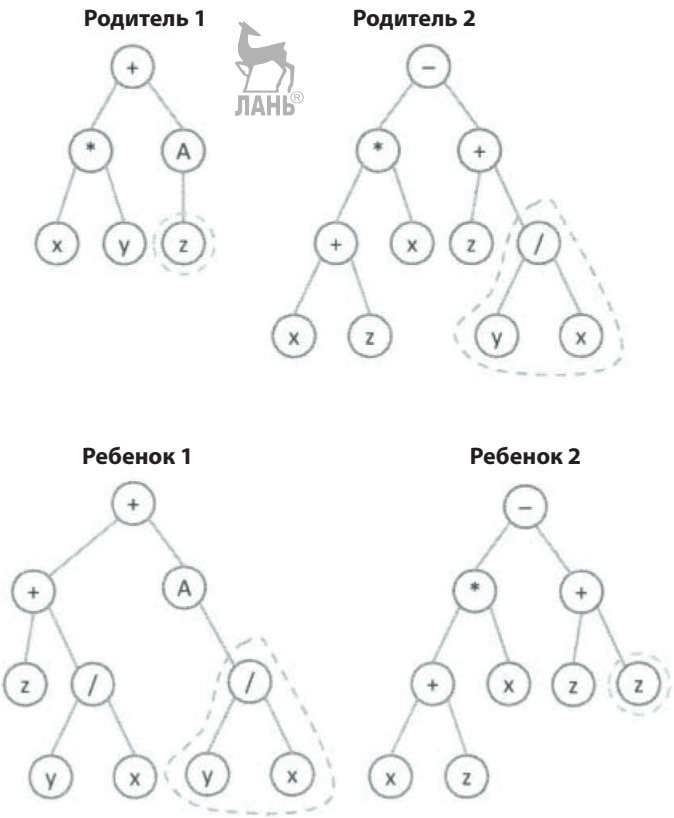


Рис. 7.4. Два синтаксических дерева (родители) скрещиваются и производят два новых синтаксических дерева (детей). Показанные здесь родители – такие же, как и на рис. 7.3, но точки скрещивания отбираются по-другому.

7.2. Основы генетического программирования

Теперь, когда мы знаем, как объединять программы на Lisp, у нас есть инструменты для обобщения эволюционных алгоритмов на эволюционное формирование компьютерных программ. Рисунок 7.5 дает схематичное описание простой генетической программы. Мы видим, что она похожа на генетический алгоритм, но эволюционный алгоритм формирует решения оптимизационной задачи, в то время как генетическое программирование формирует компьютерные программы, которые сами могут решать оптимизационную задачу.

Прежде чем мы сможем реализовать генетическое программирование, нам нужно принять несколько базовых решений.

1. Какова мера приспособленности на рис. 7.5?
2. Каков критерий останова на рис. 7.5?
3. Каково множество терминальных символов для эволюционирующих компьютерных программ? То есть какие символы могут появляться в листьях синтаксических деревьев?
4. Каково множество функций для эволюционирующих компьютерных программ? То есть какие функции могут появляться в нетерминальных узлах синтаксических деревьев?
5. Как генерировать исходную популяцию компьютерных программ?
6. Какие еще параметры необходимо определить для управления выполнением генетической программы?

Некоторые из этих решений также необходимы для других эволюционных алгоритмов, но иные являются специфичными для генетического программирования. В следующих далее разделах поочередно обсуждается каждый из этих вопросов.

```
Родители ← {случайно сгенерированные компьютерные программы}
While not (критерий останова)
  Рассчитать приспособленность каждого родителя в популяции.
  Дети ← Ø
  While | Дети | < | Родители |
    Применить приспособленности для вероятностного отбора родителей p1 и p2.
    Спарить p1 и p2 для создания детей c1 и c2.
    Дети ← Дети U {c1, c2}
  Loop
  Случайно мутировать несколько детей.
  Родители ← Дети
Next поколение
```

Рис. 7.5. Концептуальный вид простой генетической программы

7.2.1. *Мера приспособленности*

Какова мера приспособленности на рис. 7.5? Это решение должно быть принято для всех эволюционных алгоритмов, но в случае с генетическим программированием решение сложнее. Компьютерная программа должна хорошо работать для широкого спектра входных данных, разных исходных условий и разных сред. Например, программа поиска топливосберегающей траектории спутника с одной орбиты на другую должна хорошо работать для разных параметров спутника и

разных орбит. Следовательно, при определении приспособленности компьютерной программы необходимо использовать много разных условий. Для конкретной программы каждое вводимое в компьютер множество и оперативное состояние возвращает свою «подприспособленность». Как совместить эти приспособленности, для того чтобы получить для компьютерной программы единую меру приспособленности? Должны ли мы использовать среднюю результативность? Должны ли мы попытаться максимизировать результативность по худшему варианту? Должны ли мы использовать какую-то комбинацию этих двух условий? Эти вопросы естественным образом приводят к многокритериальной оптимизации (глава 20), несмотря на то что в генетическом программировании нет необходимости использовать многокритериальную оптимизацию.

7.2.2. *Критерии останова*

Каков критерий останова на рис. 7.5? На этот вопрос необходимо ответить для абсолютно всех эволюционных алгоритмов (см. раздел 8.2), но это может быть особенно важно для генетического программирования. Это связано с тем, что мера приспособленности обычно более требовательна к вычислениям именно в генетическом программировании, чем в других эволюционных алгоритмах. Выбор критерия останова может определить, является ли генетическая программа успешной. Как и в случае других эволюционных алгоритмов, критерий останова для генетической программы может включать такие факторы, как число итераций, число оцениваний приспособленности, время выполнения, лучшее значение приспособленности, изменение лучшей приспособленности в течение нескольких поколений или стандартное отклонение значений приспособленности по всей популяции.

7.2.3. *Множество терминальных символов*

Каким является множество терминальных символов для эволюционирующих компьютерных программ? Данное множество описывает символы, которые могут появляться на листьях синтаксических деревьев. Множество терминальных символов – это множество всех возможных входов в эволюционирующие компьютерные программы. Это множество включает в себя переменные, которые вводятся в компьютерную программу, а также константы, которые, по нашему мнению, могли бы представлять важность. Константы могут включать элементарные целые числа, такие как 0 и 1, а также константы, которые мо-

гут быть важны для конкретной оптимизационной задачи (π, e и т. д.). Синтаксические деревья на рис. 7.3 имеют три терминальных символа: x, y и z. Некоторые константы могут быть получены неявным образом; например, $x - x = 0$ и $x/x = 1$. Поэтому при условии, что у нас есть функции вычитания и деления, нам действительно не нужны константы 0 и 1. Однако большинство реализаций генетических программ должно включать в свои множества терминальных символов константы.

Во множестве терминальных символов мы также можем использовать случайные числа, но обычно мы не хотим, чтобы случайное число менялось после его генерации. Такие случайные числа называются эфемерными случайными константами [Koza, 1992, глава 26]. Эфемерные случайные константы получаются путем указания величины, обозначаемой во множестве терминальных символов, как $\mathcal{R}$. Если величина $\mathcal{R}$ выбрана в качестве терминального символа во время инициализации популяции, то мы генерируем случайное число r_1 между заданными пределами и вставляем r_1 в особи генетической программы. С этого момента конкретное значение r_1 не изменяется. Однако если $\mathcal{R}$ выбрана снова для инициализации еще одной особи или для мутации, то мы генерируем новую случайную константу r_2 для этой реализации. Выбор пределов для генерации эфемерных случайных констант является в генетическом программировании еще одним проектным решением.

Определение терминального символа для приложения генетического программирования – это настоящая эквилибристика. Если мы используем множество, которое слишком мало, то генетическая программа не сможет эффективно решить нашу задачу. Однако если мы используем слишком большое множество терминальных символов, то для генетической программы может быть слишком сложно найти хорошее решение в разумные сроки. Дж. Коза изучает этот вопрос в своей публикации [Koza, 1992, глава 24] на примере простой задачи обнаружения программы $x^3 + x^2 + x$ на основе 20 тестовых случаев. Для этой задачи генетической программе требуется только один терминальный символ – x. Когда множество терминальных символов является минимальным $\{x\}$, генетическая программа находит правильную программу в течение 50 поколений в 99,8 % случаев. В табл. 7.1 показано, как уменьшается вероятность успеха при добавлении во множество терминальных символов генетической программы дополнительных членов (случайных чисел с плавающей точкой). В этой простой задаче вероятность успеха линейно уменьшается вместе с числом посторонних переменных[1] во

[1] Внешняя, или посторонняя, переменная (extraneous variable) – это нежелательная переменная, которая влияет на связь между переменными, изучаемыми экспериментатором. – *Прим. перев.*

множестве терминальных символов. Хорошей новостью является то, что даже когда 32 из 33 членов во множестве терминальных символов являются посторонними, генетическая программа по-прежнему в состоянии решить задачу в 35 % случаев.

Таблица 7.1. Вероятность успеха генетической программы после 50 поколений при обнаружении программы $x^3 + x^2 + x$. Размер популяции был 1000. Данные получены из публикации [Koza, 1992, глава 24]

Число дополнительных переменных	Вероятность успеха (процент)
0	99.8
1	96.6
4	84.0
8	67.0
16	57.0
32	35.0

7.2.4. *Множество функций*

Каким является множество функций для эволюционирующих компьютерных программ? Это множество описывает функции, которые могут отображаться в нетерминальных узлах синтаксических деревьев, например следующие.

- Во множество функций могут быть включены стандартные математические операторы (например, сложение, вычитание, умножение, деление, абсолютная величина).
- Во множество функций могут быть включены специфичные для конкретной задачи функции, которые мы считаем важными для нашей конкретной оптимизационной задачи (например, экспоненциальные, логарифмические, тригонометрические функции, фильтры, интеграторы, дифференциаторы).
- Во множество функций могут быть включены условные проверки (например, больше, меньше, равно).
- Во множество функций могут быть включены логические функции, если мы считаем, что они могут быть применимы к решению нашей конкретной оптимизационной задачи (например, and, nand, or, xor, nor, nor, not).

- Во множество функций могут быть включены функции присваивания переменных.
- Во множество функций могут быть включены инструкции цикла (например, циклы while, циклы for).
- Во множество функций могут быть включены вызовы подпрограмм, если у нас есть множество стандартных функций, которые мы создали для наших задач.

Синтаксические деревья на рис. 7.3 включают пять функций: сложение, вычитание, умножение, деление и абсолютную величину. Нам нужно найти правильный баланс в определении множества функций и множества терминальных символов. Эти множества должны быть достаточно большими, чтобы быть в состоянии представить решение нашей задачи, но если они слишком велики, то поисковое пространство будет настолько большим, что генетической программе будет трудно найти хорошее решение.

Некоторые функции должны быть модифицированы для генетической программы, поскольку синтаксические деревья эволюционируют и, возможно, не будет допустимых функциональных аргументов. Например, генетическая программа может эволюционно сформировать S-выражение (/ *x* 0), которое представляет собой деление на ноль. Это привело бы к ошибке Lisp, которая заставила бы генетическую программу завершиться. Поэтому вместо использования стандартного для языка Lisp оператора деления мы можем определить оператор деления DIV, который защищает от деления на ноль, а также от переполнения из-за деления на очень малое число:


```
(defun DIV (x y)                    ; определить защищенную функцию деления
(if (< (abs y) e) (return-from DIV 1)) ; вернуть 1, если делитель очень мал
return-from DIV (/ x y))            ; иначе вернуть x/y                    (7.4)
```

где *e* – очень малая положительная константа, например 10^{-20}. Уравнение (7.4) показывает синтаксис языка Lisp для определения защищенной подпрограммы деления[2]. Функция DIV возвращает 1, если делитель имеет очень малую величину. Возможно, нам придется переопределить другие функции аналогичным образом (логарифмические функции, обратные тригонометрические функции и т. д.), для того чтобы функции в нашем множестве функций гарантированно могли обрабатывать всевозможные входные данные.

[2] Обратите внимание, что в функции Lisp любой текст после точки с запятой интерпретируется как комментарий.

7.2.5. *Инициализация*

Каким образом генерировать исходную популяцию компьютерных программ? У нас есть два основных варианта инициализации, которые называются полным методом и методом выращивания. Мы также можем объединить эти варианты и получить третий вариант, который называется линейным методом «половина на половину» (ramped half-and-half, RHH) [Koza, 1992].

Полный метод создает программы, такие что число узлов от каждого терминального узла до узла верхнего уровня равняется D_c, задаваемой пользователем константе. D_c называется глубиной синтаксического дерева. В качестве примера родитель 1 на рис. 7.3 имеет глубину три, в то время как родитель 2 имеет глубину четыре. Родитель 1 на рис. 7.3 представляет собой полное синтаксическое дерево, поскольку от каждого терминального узла до узла сложения верхнего уровня имеется по три узла. Однако родитель 2 не является полным синтаксическим деревом, поскольку некоторые ветви программы имеют глубину четыре, а другие – только три.

Для генерации случайных синтаксических деревьев можно применять рекурсию. Например, если мы хотим создать синтаксическое дерево со структурой, подобной родителю 2 на рис. 7.3, мы сначала создадим узел вычитания на верхнем уровне и отметим, что он требует двух аргументов. Для первого аргумента мы генерируем узел умножения и отмечаем, что он требует двух аргументов. Этот процесс продолжается для каждого узла и каждого аргумента до тех пор, пока не будет создано достаточное число уровней для достижения нужной глубины. Когда мы достигаем желаемой глубины, то генерируем случайный терминальный узел для завершения этой ветви синтаксического дерева. На рис. 7.6 показана концепция рекурсивного алгоритма, генерирующего случайные компьютерные программы. Мы можем сгенерировать случайное синтаксическое дерево, вызвав процедуру GrowProgramFull(D_c, 1), где D_c – желаемая глубина синтаксического дерева. Процедура GrowProgramFull вызывает себя каждый раз, когда ей нужно добавить еще один уровень в растущем синтаксическом дереве.

Другой метод инициализации, *метод выращивания*, создает программы – такие, что число узлов от каждого терминального узла до узла верхнего уровня меньше или равно D_c. Если бы родители на рис. 7.3 были созданы путем случайной инициализации, то родитель 1 мог быть сгенерирован либо полным методом, либо методом выращивания, в то время как родитель 2 определенно был сгенерирован с помощью мето-

да выращивания, так как он не является полным синтаксическим деревом. Метод выращивания может быть реализован так же, как и полный метод, за исключением того, что при генерации случайного узла на глубине меньше D_c может быть сгенерирована либо функция, либо терминальный узел. Если создается узел функции, то синтаксическое дерево продолжает расти. Как и в полном методе, когда мы достигаем максимальной глубины D_c, мы генерируем случайный терминал для завершения этой ветви синтаксического дерева. Рисунок 7.7 иллюстрирует концепцию рекурсивного алгоритма, который генерирует случайные компьютерные программы с помощью метода выращивания.

```
function [SyntaxTree] = GrowProgramPull(Depth, NumArgs)
SyntaxTree ← Ø
For i = 1 to NumArgs
  If Depth = 1
    SyntaxTree ← Случайный терминальный символ
  else
    NewFunction ← Случайно выбранная функция
    NewNumArgs ← Число аргументов, требуемых функцией NewFunction
    SyntaxTree ← (NewFunction + GrowProgramFull(Depth–1, NewNumArgs))
  End
Next i
```

Рис. 7.6. Концептуальный вид рекурсивного алгоритма выращивания случайных синтаксических деревьев в форме S-выражения с помощью полного метода. Эта процедура первоначально вызывается в синтаксической форме GrowProgramFull (D_c, 1), где D_c – это требуемая глубина случайного синтаксического дерева. Оператор «+» обозначает конкатенацию строк. Обратите внимание, что этот алгоритм является концептуальным; он не включает все детали, необходимые для создания допустимого синтаксического дерева, например правильное размещение скобок

```
function [SyntaxTree] = GrowProgramGrow(Depth, NumArgs)
SyntaxTree ← Ø
For i – 1 to NumArgs
  If Depth = 1
    SyntaxTree ← Случайный терминальный символ
  else
    NewNode ← Случайно выбранная функция или терминальный символ
    If NewNode является терминальным символом
      SyntaxTree ← (SyntaxTree + NewNode)
    else
      NewNumArgs ← Число аргументов, требуемых узлом NewNode
```

```
        SyntaxTree ← (NewNode + GrowProgramGrow(Depth–1, NewNumArgs)
      End
    End
  Next i
```

Рис. 7.7. Концептуальный вид рекурсивного алгоритма выращивания случайных синтаксических деревьев в форме S-выражения с помощью метода выращивания. Эта процедура первоначально вызывается в синтаксической форме GrowProgramGrow(D_c, 1), где D_c – это требуемая глубина случайного синтаксического дерева. Как и на рис. 7.6, оператор «+» обозначает конкатенацию строк, но этот алгоритм не включает все детали, необходимые для его реализации

Метод «*половина на половину*» генерирует половину исходной популяции с помощью полного метода и половину с помощью метода выращивания. Кроме того, он генерирует равное число синтаксических деревьев для каждого значения глубиной между 2 и D_c. То есть максимальной допустимой глубиной, задаваемой пользователем. Рисунок 7.8 иллюстрирует концепцию инициализации синтаксического дерева методом «половина на половину».

```
D_c = максимальная глубина синтаксического дерева
N = размер популяции
For i = 1 to N
  Depth ← U[2, D_c]
  r ← U[0, 1]
  If r < 0.5
    SyntaxTree(z) ← GrowProgramGrow(Depth, 1)
  else
    SyntraxTree(i) ← GrowProgramFull(Depth, 1)
  End
Next i
```

Рис. 7.8. Алгоритм создания исходной популяции генетической программы методом «половина на половину». $U[2, D_c]$ – случайное целое число, равномерно распределенное на $[2, D_c]$, а $U[0,1]$ – случайное вещественное число, равномерно распределенное на [0, 1]. Данный алгоритм вызывает подпрограммы рис. 7.6 и 7.7

Дж. Коза экспериментировал с тремя разными описанными выше типами инициализации для нескольких простых задач генетического программирования [Koza, 1992, глава 25]. Он обнаружил разницу в вероятности успеха генетической программы в зависимости от того, какой метод инициализации был использован, как показано в табл. 7.2. Таблица говорит, что линейный метод инициализации «половина на половину» обычно намного лучше, чем два других метода инициализации.

Таблица 7.2. Вероятность успеха генетической программы для разных задач и разных методов инициализации. Эти данные получены из публикации [Koza, 1992, глава 25]

Задача	Полный	Выращивания	Половина на половину
Символическая регрессия	33	173	233
Булева логика	423	533	663
Искусственный муравей	143	503	463
Линейное уравнение	63	373	533

В заключение нашего обсуждения инициализации следует отметить, что часто бывает полезно посеять исходную популяцию эволюционного алгоритма несколькими известными хорошими особями. Эти хорошие особи могут быть сгенерированы пользователями, либо они могут поступить из какого-то другого оптимизационного алгоритма или иного источника. Вместе с тем посев необязательно улучшает результативность эволюционного алгоритма. Если в исходной популяции есть только несколько хороших особей, а остальные особи являются относительно слабыми случайно сгенерированными особями, то в процессе отбора малое число хороших особей может доминировать, а слабые особи могут быстро исчезнуть. Это может привести к эволюционному тупику и преждевременному схождению, известному как «выживание посредственности» [Koza, 1992, стр. 104]. Однако вероятность возникновения этого негативного события зависит от типа отбора, который мы используем (см. раздел 8.7). Если мы применим рулеточный отбор, то давление отбора велико, и несколько приспособленных особей, скорее всего, вскоре начнут в популяции доминировать. Если мы применим турнирный отбор, то давление отбора гораздо ниже, и вероятность того, что несколько приспособленных особей будет в популяции доминировать, соответственно, ниже.

7.2.6. *Параметры генетической программы*

Какие параметры управляют исполнением генетической программы? Эти параметры включают в себя те, которые используются для других эволюционных алгоритмов, а также включают параметры, специфичные для генетического программирования.

1. Нужно указать метод отбора, с помощью которого родители отбираются для участия в скрещивании. Можно использовать от-

бор пропорциональной приспособленности, турнирный отбор или какой-то другой метод. Фактически мы можем использовать любой из методов отбора, описанных в разделе 8.7.
Здесь также уместно упомянуть, что мы можем разумнее реализовать древовидное скрещивание, чем просто отбирать случайные точки скрещивания. Некоторые поддеревья полезнее, чем другие, и, возможно, мы не захотим разбивать эти поддеревья. Мы можем квантифицировать приспособленность поддеревьев, получив корреляции между точками скрещивания и приспособленностью дочерних программ, а затем использовать эти корреляции для смещения отбора будущих точек скрещивания [Iba и de Garis, 1996].

2. Нужно указать размер популяции. Поскольку в компьютерных программах существует очень много степеней свободы, генетическая программа обычно имеет более крупную популяцию, чем другие эволюционные алгоритмы. Генетическая программа обычно имеет размер популяции не менее 500 особей, и часто он переваливает за несколько тысяч особей.
3. Нужно указать метод мутации. На протяжении многих лет в генетическом программировании использовались различные методы мутации, некоторые из которых описаны ниже:
   a) мы можем отобрать случайный узел и заменить все, что находится ниже этого узла, случайным синтаксическим поддеревом. Это называется поддревесным мутированием [Koza, 1992, стр. 106]. Оно эквивалентно скрещиванию программы со случайно сгенерированной программой, а также называется скрещиванием безголового цыпленка (headless chicken crossover) [Angeline, 1997];
   b) мутация расширения заменяет терминал случайным поддеревом. Это эквивалентно мутации поддерева, если замененный узел в мутации поддерева является терминальным;
   c) мы можем заменить случайно отобранный узел либо терминальный символ новым сгенерированным узлом или терминальным символом. Это называется точечной мутацией, или мутацией с заменой узла, и требует, чтобы арность замещаемого узла была равна арности замещающего узла[3]. Например, мы можем заменить операцию сложения операцией

[3] Арность функции равна числу ее аргументов. Например, константа имеет арность 0, функция абсолютной величины имеет арность 1, и функция сложения может иметь арность 2 или более.

умножения или операцию абсолютной величины операцией синуса;

d) подъемная (hoist) мутация создает новую программу, которая является случайно отобранным поддеревом родительской программы [Kinnear, 1994];

e) усадочная (shrink) мутация заменяет случайно отобранное синтаксическое поддерево отобранным терминальным символом [Angeline, 1996a]; этот метод также называется мутацией со сверткой поддерева. Подъемная мутация и усадочная мутация первоначально были введены для сокращения раздувания программного кода (см. раздел 7.4);

f) перестановочная мутация случайно переставляет аргументы случайно отобранной функции [Koza, 1992]. Например, можно заменить аргументы *x* и *y* функции деления. Конечно, этот тип мутации никак не влияет на коммутативные функции;

g) мы можем случайно мутировать константы в программе [Schoenauer и соавт., 1996] .

Мы часто реализуем мутацию таким образом, что мутирующая программа повторно замещает исходную программу, только если она более приспособлена. Эта идея «заменить, только если более приспособлена» может быть применена к мутации в любом эволюционном алгоритме.

4. Нужно указать вероятность мутации p_m. Это аналогично другим эволюционным алгоритмам. Мутация в генетической программе с *n* особями часто реализуется с помощью метода, который похож на следующий ниже:

For each кандидатная компьютерная программа x_i, где $i \in [1, N]$
  Сгенерировать случайное число r, равномерно распределенное на [0, 1].
  If $r < p_m$
    Случайно отобрать узел *k* в компьютерной программе x_i.
    Заменить отобранное поддерево, начиная с узла *k*, на
      случайно сгенерированное поддерево.
  End
Next компьютерная программа

Большой размер популяции, используемый в генетической программе, наряду с большим числом возможных узлов, в которых

может произойти скрещивание, обычно означает, что хорошие результаты генетической программы не зависят от мутации [Koza, 1992, глава 25]. Часто мы можем получить хорошие результаты при $p_m = 0$. Тем не менее мутация по-прежнему может быть желательной в случае потери важного терминального символа или функции из популяции. Если это происходит, то мутация является единственным способом, которым они могут повторно войти в популяцию.

5. Нужно указать вероятность скрещивания p_c. Это аналогично эволюционным алгоритмам. После отбора двух родителей на рис. 7.5 мы можем либо применить скрещивание для их объединения, либо клонировать их для следующего поколения. Строка:

   Спарить p_1 и p_2 для создания детей c_1 и c_2.

   на рис. 7.5 тогда будет заменена на нечто вроде следующего:

   ```
   Сгенерировать случайное число r, равномерно распределенное на [0, 1].
   If r < p_c
     Спарить p_1 и p_2 для создания детей c_1 и c_2.
   else
     c_1 ← p_1
     c_2 ← p_2
   End
   ```

   Большинство опыта показывает, что скрещивание является важным аспектом генетического программирования и должно применяться с вероятностью $p_c \geq 0.9$ [Koza, 1992, глава 25].

6. Нужно решить, использовать элитарность или нет. Как и в любом другом эволюционном алгоритме, мы можем оставлять лучшие m компьютерных программ в генетической программе из поколения в поколение, для того чтобы они гарантированно не были потеряны в следующем поколении (см. раздел 8.4). Параметр m называется параметром элитарности. Элитарность может быть реализована несколькими способами. Например, можно заархивировать лучших m особей в конце поколения, обычным образом создать детей для следующего поколения, а затем заменить худших m детей элитарными особями из предыдущего поколения. Кроме того, мы можем копировать m элитарных особей в первые m детей каждого поколения, а затем создавать только $(N - m)$ детей в каждом поколении (где N – это размер популяции).

7. Нужно указать D_i, максимальный размер программы исходной популяции. Размер программы можно квантифицировать по ее глубине, которая измеряет максимальное число узлов между самым высоким и самым низким уровнями (включительно). Например, родитель 1 на рис. 7.3 имеет глубину три, в то время как родитель 2 имеет глубину четыре.
8. Также нужно указать D_c, максимальную глубину дочерних программ. Во время работы генетической программы дочерние программы могут расти все больше и больше с каждым последующим поколением. Если максимальная глубина не применяется, то дочерние программы могут стать неоправданно длинными, тратя пространство и время исполнения; это называется раздуванием генетической программы (раздел 7.4). Максимальная глубина D_c может обеспечиваться несколькими способами. Один из способов – заменить ребенка одним из родителей, если глубина ребенка превышает D_c. Другой способ – повторить операцию скрещивания, если глубина ребенка превышает D_c. Еще один способ – исследовать родительские синтаксические деревья перед отбором их точек скрещивания и ограничить случайно отобранные точки скрещивания так, чтобы глубина D_c не была превышена глубинами детей.
9. Нужно решить, хотим ли мы позволить заменять терминальный узел в синтаксическом дереве поддеревом во время скрещивания. Рисунок 7.4 показывает, что терминальный узел 2 в родителе 1 отобран для скрещивания и заменен поддеревом в ребенке 1. Мы используем p для обозначения вероятности скрещивания во внутреннем узле. При выборе точки скрещивания генерируется случайное число r, равномерно распределенное на [0, 1]. Если r меньше p_i, то для скрещивания мы отбираем терминальный узел, то есть мы отбираем символ в синтаксическом дереве, которому непосредственно не предшествует левая круглая скобка. Однако если r больше p_i, то для скрещивания мы отбираем S-выражение, то есть для скрещивания мы отбираем поддерево, окруженное соответствующими левой и правой скобками.
10. Нужно решить, стоит ли беспокоиться о дублирующих особях в популяции. Дубликаты – пустая трата компьютерных ресурсов. В эволюционных алгоритмах с относительно небольшим поисковым пространством или небольшой популяцией дубликаты могут возникать довольно часто, и работа с дубликатами может быть

важным аспектом эволюционного алгоритма (см. раздел 8.6.1). Однако в генетическом программировании поисковое пространство настолько велико, что дубликаты встречаются редко. Поэтому в генетическом программировании нам обычно не нужно беспокоиться о дублирующих особях.

7.3. Генетическое программирование: управление объектом за минимальное время

В этом разделе, который мотивирован публикацией [Koza, 1992, раздел 7.1], мы демонстрируем применение генетического программирования для управления за минимальное время ньютоновской системой второго порядка. Ньютоновская система второго порядка – это простая система, характеризуемая положением, скоростью и ускорением, которая удовлетворяет уравнениям

$$\dot{x} = v$$
$$\dot{v} = u \tag{7.5}$$

где x – это положение, v – скорость и u – заданное ускорение. То есть производной от положения является скорость, а производной от скорости является ускорение. Мы рассматриваем движение только в одном измерении. Задача состоит в том, чтобы найти профиль ускорения $u(t)$ для перевода системы из некоторого исходного положения $x(0)$ и скорости $v(0)$ в $x(t_f) = 0$ и $v(t_f) = 0$ за минимальное время t_f. Интуиция подсказывает нам, как приблизительно это сделать: мы разгоняемся максимально быстро в одном направлении, пока не достигнем определенного положения, а затем разгоняемся максимально быстро в противоположном направлении, пока не достигнем $x(0) = v(0) = 0$[4].

Для простоты и не теряя общности, мы исходим из того, что максимальная величина разгона равна 1 и что мы можем разгоняться в любом направлении. Задача управления за минимальное время показана в верхней части рис. 7.9. Мы разгоняемся в положительном направлении (вправо) до достижения стратегической точки с надписью «Смена курса». Затем мы разгоняемся в отрицательном направлении (влево) до достижения цели. Обратите внимание, что скорость транспортного

[4] На самом деле это то, что мы наблюдаем у водителей-подростков на каждом светофоре: максимально ускоряться, когда свет становится зеленым, а затем в тщательно выверенной точке перед следующим стоп-сигналом хлопать по тормозам. Если подросток выбрал время правильно, то автомобиль остановится ровно на следующем красном сигнале светофора, и время в пути между стоп-сигналами будет сведено к минимуму.

средства находится справа в течение всего периода времени. Если время подобрано правильно, то мы достигнем цели с нулевой скоростью.

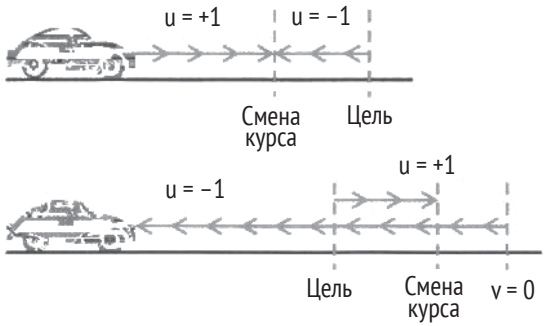


Рис. 7.9. Иллюстрация управления автомобилем за минимальное время. В верхнем рисунке мы разгоняемся вправо до точки смены курса, затем разгоняемся влево и достигаем цели с нулевой скоростью. На нижнем рисунке наша исходная скорость настолько высока, что мы неизбежно проскочим мимо цели. В этом случае мы разгоняемся влево, проскакиваем мимо цели, переключаемся на разгон вправо в точке смены курса и достигаем цели с нулевой скоростью

Может оказаться так, что у нас будет настолько высокая исходная скорость, что мы не сможем остановиться перед целью. В этом случае мы неизбежно проскочим мимо цели, и поэтому должны вернуться обратно к цели, достигнув ее с нулевой скоростью. Эта ситуация немного менее интуитивно понятна и показана внизу рис. 7.9. Решение задачи за минимальное время состоит в том, чтобы сначала как можно больше разогнаться в отрицательном направлении (влево). Машина проскакивает мимо цели. В итоге автомобиль достигнет нулевой скорости, после чего он начнет движение влево. Мы продолжаем разгоняться в отрицательном направлении до достижения стратегической точки с надписью «Смена курса», в это время мы начинаем разгоняться в положительном направлении (вправо) до возвращения к цели. Опять же, если время подобрано правильно, мы достигнем цели с нулевой скоростью.

Задача управления за минимальное время является классической задачей оптимального управления во многих аэрокосмических приложениях и подробно изучается во многих книгах по оптимальному управлению [Kirk, 2004]. Ее решение называется двухпозиционным управлением «на полном газу» (bang-bang control), потому что для любого начального условия $x(0)$ и $u(0)$ решение состоит из одного временного периода максимального разгона в одном направлении, за которым следует период времени максимального разгона в другом направлении.

Задача управления за минимальное время может быть представлена в графической форме с помощью диаграммы фазовой плоскости, как показано на рис. 7.10. Для простоты будем считать, что масса автомобиля равна 2. В этом случае кривая, изображенная на рис. 7.10, которая называется кривой смены курса, имеет вид

$$x = -v|v|/2. \tag{7.6}$$

Цель состоит в том, чтобы достичь исходной точки $x = 0$ и $v = 0$ за минимальное время из любой начальной точки в фазовой плоскости. Если положение и скорость выше кривой смены курса, то мы должны применить максимальный разгон в отрицательном направлении. Если положение и скорость ниже кривой смены курса, то мы должны применить максимальный разгон в положительном направлении. Это приведет нас к траектории, которая достигает кривой смены курса, и в этот момент мы изменим направление разгона. Затем мы будем следовать за кривой смены курса к началу фазовой плоскости.

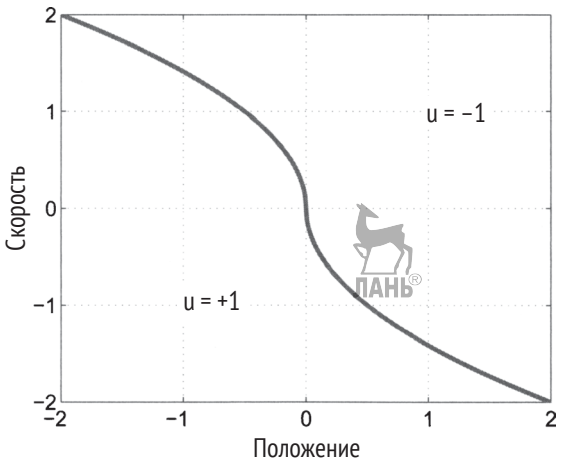


Рис. 7.10. Кривая смены курса, или переключения, для управления за минимальное время. Если положение и скорость лежат выше кривой смены курса, то разгон должен быть максимальным в отрицательном направлении. Если положение и скорость лежат ниже кривой, то разгон должен быть максимальным в положительном направлении

Рисунок 7.11 иллюстрирует оптимальную траекторию для исходного условия $x(0) = -0.5$ и $v(0) = 1.5$. Это соответствует нижнему рисунку на рис. 7.9. Автомобиль движется слишком быстро, чтобы остановиться перед достижением $x = 0$. Поэтому мы применяем максимальный раз-

гон в отрицательном направлении до достижения кривой смены курса; обратите внимание, что автомобиль проходит через $v = 0$ во время максимального отрицательного разгона. Когда автомобиль достигает кривой смены курса, мы применяем максимальный разгон в прямом направлении. Траектория достигает исходной точки фазовой плоскости ($x = 0$ и $v = 0$) за минимально возможное время.

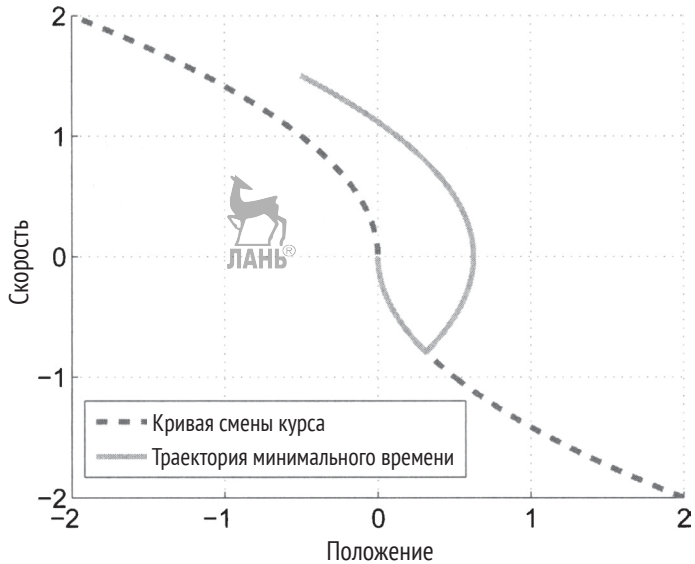


Рис. 7.11. Траектория минимального времени для исходного условия $x(0) = -0.5$ и $v(0) = 1.5$. Разгон равен −1 выше кривой смены курса и +1 после того, как траектория достигает кривой смены курса

Теперь мы применим генетическое программирование, для того чтобы попытаться эволюционно сформировать программу для задачи управления за минимальное время. Для этой задачи мы определим две специальные функции языка Lisp. Первая – защищенный оператор деления, показанный в уравнении (7.4), а второй – оператор «больше»:

```
(defun GT (x y)
(if (> x y) (return-from GT 1) (return-from GT -1)))        (7.7)
```

Функция GT возвращает 1, если $x > y$, и возвращает −1 в противном случае.

Для того чтобы оценить стоимость программы, возьмем 20 случайных исходных точек в фазовой плоскости (x, v), где $|x| < 0.75$ и $|v| < 0.75$, и посмотрим, сможет ли программа довести каждую из пар (x, v) до ис-

ходной точки в течение 10 секунд. Если программа успешна для исходного условия, то вкладом стоимости этой симуляции является время, необходимое для приведения (x, v) к исходной точке. Если программа не будет успешной в течение 10 секунд, то вклад стоимости этой симуляции равен 10. Общая стоимость компьютерной программы является средним всех 20 вкладов. В табл. 7.3 обобщены параметры генетической программы для этой задачи, которые в основном основаны на публикации [Koza, 1992, раздел 7.1].

Таблица 7.3. Параметры генетической программы для задачи управления автомобилем за минимальное время

Параметр генетической программы	Значение параметра
Цель	Найти программу управления автомобилем за минимальное время
Множество терминальных символов	x (положение), v (скорость), –1
Множество функций	+, –, *, DIV, GT, ABS
Стоимость	Время, необходимое для приведения автомобиля в исходную точку фазовой плоскости усредненно по 20 случайным исходным условиям
Лимит поколений	50
Размер популяции	500
Максимальная исходная глубина дерева	6
Метод инициализации	Половина на половину
Максимальная глубина дерева	17
Вероятность скрещивания	0.9
Вероятность мутации	0
Число элитарных особей	2
Метод отбора	Турнир (см. раздел 8.7.6)

На рис. 7.12 показана стоимость лучшего решения генетической программы в зависимости от номера поколения. Лучшая компьютерная программа найдена после менее чем 10 поколений для определенного прогона, но средняя результативность всей популяции в целом про-

должает уменьшаться во время всех 50 поколений. В большинстве задач генетического программирования, для того чтобы найти лучшее решение, требуется гораздо дольше, чем 10 поколений. Причина, по которой этот конкретный прогон был быстрее, чем средний прогон генетической программы, возможно, была вызвана относительной простотой задачи, либо это, возможно, было просто статистической случайностью. Лучшее решение, полученное генетической программой, выглядит так:

$$u = (* (\mathrm{GT}\ (- (\mathrm{DIV}\ x\ v)\ (- -1\ v))\ (\mathrm{GT}\ (+ v\ x)\ (\mathrm{DIV}\ x\ v))) \\ (\mathrm{DIV}\ (\mathrm{GT}\ (+ x\ v)\ (+ v\ x))\ (\mathrm{GT}\ (+ v\ x)\ x))))) \quad (7.8)$$

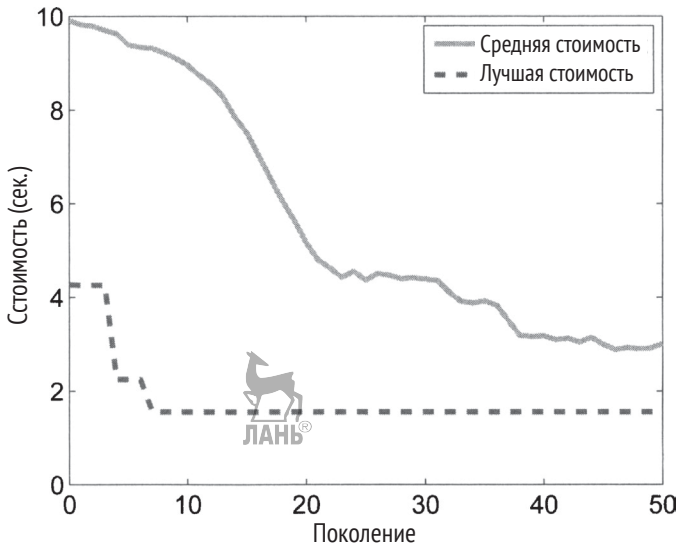


Рис. 7.12. Результативность генетической программы для задачи управления за минимальное время. Лучшее решение найдено по прошествии менее чем 10 поколений для конкретного прогона

Кривая смены курса для этого управления показана на рис. 7.13 вместе с теоретически оптимальной кривой смены курса. Для $v < 0$ эти две кривые очень похожи. Для $v > 0$ существует большая разница между кривыми, но общая форма по-прежнему аналогична.

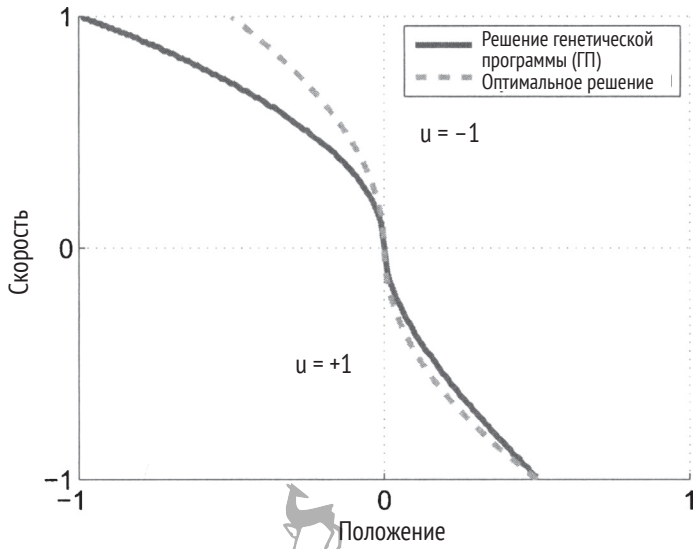


Рис. 7.13. Лучшая кривая смены курса, полученная генетической программой для задачи управления за минимальное время вместе с теоретически оптимальной кривой смены курса

Время, которое требуется автомобилю для достижения исходой точки фазовой плоскости, усредненно по 10 000 случайным исходным условиям в пространстве состояний $x \in [-0.75, +0.75]$ и $v \in [-0.75, +0.75]$, составляет около 1.53 секунды для оптимальной кривой смены курса и 1.50 секунды для кривой смены курса, полученной генетической программой. Интересно, что полученная генетической программой смена курса фактически работает немного лучше, чем оптимальная кривая смены курса! Теоретически это невозможно, но практика и теория не всегда совпадают. На практике возникают вопросы реализации, которые позволяют выполнять стратегию лучше теоретически оптимальной. Например, мы завершили нашу симуляцию, когда $|x| < 0.01$ и $|u| < 0.01$, и посчитали такие малые значения полным успехом. Теоретически мы можем достичь начальной точки с нулевой ошибкой, но на практике это не так. Также мы применили шаг $\tau = 0.02$ секунды для симулирования динамической системы. Вместо вычисления точного непрерывного во времени решения

$$\begin{aligned} \dot{v} &= u \\ \dot{x} &= v \end{aligned} \tag{7.9}$$

мы аппроксимировали решение как

$$v_{k+1} = v_k + \tau u_k$$
$$x_{k+1} = x_k + \tau(v_k + v_{k+1})/2 \quad (7.10)$$

где k – это индекс времени в диапазоне от 0 до 500 (то есть от 0 до 10 секунд). То есть мы применили прямоугольное интегрирование для получения скорости и трапециевидное интегрирование для получения положения [Simon, 2006, глава 1]. Эти разницы между теорией и практикой могут привести к тому, что управление будет больше болтаться вдоль оптимальной кривой смены курса, чем сейчас вдоль сгенерированной генетической программой кривой смены курса[5]. Читатель может воспроизвести результаты этого раздела, выполнив действия, описанные в задаче 7.13.

Теория и практика

Превосходство кривой смены курса, сгенерированной генетической программой, над теоретически оптимальным решением поднимает важный вопрос относительно разницы между теорией и практикой. Инженерные решения часто генерируются на основе теории, но, как известно любому практикующему инженеру, теоретические результаты должны быть модифицированы с учетом реальных соображений. Этот пример показывает, что генетическая программа может быть способной принимать во внимание реальные соображения и в результате находить решение, которое лучше, чем теоретически оптимальное решение задачи.

Может оказаться проще изучить теорию оптимального управления и решить задачу управления объектом за минимальное время более традиционным способом, вместо того чтобы научиться использовать генетическое программирование. Но это не так. Данный пример показывает нам, что генетическое программирование способно находить решения задач, для решения которых своими силами нам не хватает опыта. Он также показывает возможность нахождения «более качественных» решений, когда в расчет принимаются практические соображения.

7.4. Раздувание генетической программы

Генетическое программирование может привести к тому, что программы станут неоправданно длинными и потребуют больших вычислительных усилий. Дополнительный программный код, который фор-

[5] Дж. Коза сообщает о среднем времени 2.13 секунды для теоретически оптимальной кривой смены курса [Koza, 1992, раздел 7.1], снова указывая на различия в реализации уравнений динамической системы.

мируется во время работы генетической программы, имеет несколько разных имен, включая интроны, нежелательный код, пух, неэффективный код, код автостопом и невидимый код [Langdon и Poli, 2002, глава 11]. Некоторые примеры интронов включают следующее:

$$\begin{array}{l}(\text{not } (\text{not } x)) \\ (+\, x\ 0\ ) \\ (\text{if } (>\ 1\ 2)\ x\ y)\end{array} \qquad (7.11)$$

Любая серьезная реализация генетической программы должна защищать от раздувания с целью предотвращения неконтролируемого увеличения длины кода. Существует несколько способов защиты от раздувания.

Первый способ борьбы с раздуванием кода – использовать параметр максимальной глубины D_c, как описано в разделе 7.2.6. Однако это уравновешивающий акт. Если глубина D_c слишком мала, то мы ограничиваем поисковое пространство в генетической программе и тем самым можем уменьшить приспособленность самой лучшей программы, которую она может найти.

Второй способ борьбы с раздуванием кода – настроить наши реализации скрещивания и мутации для борьбы с раздуванием. Например, справедливое по размеру скрещивание отбирает точки скрещивания, которые уравновешивают размер родительских фрагментов кода, и это, следовательно, приводит к детям, которые не больше родителей [Langdon, 2000]. Если используется поддревесная мутация, то ее можно настроить так, чтобы размер мутирующей программы гарантированно был ограничен [Kinnear, 1993]. Подъемная мутация удаляет код для уменьшения длины программы [Kinnear, 1994], одноточечное скрещивание – это метод, который мы в этой главе не обсуждали, но он автоматически ограничивает глубину детей глубиной самого большого родителя [Poli и соавт., 2008, глава 5].

Третий способ борьбы с раздуванием – наказывать длинные программы в операциях отбора, воспроизведения и скрещивания. Эту идею можно реализовать несколькими способами. Например, мы можем добавить штраф в большие программы:

$$\textit{Штрафная стоимость} \leftarrow \textit{Стоимость} + \textit{Размер программы}. \qquad (7.12)$$

В общем случае, если будут штрафоваться большие программы, то, для того чтобы отыскать хорошее решение, потребуется больше оцениваний приспособленности [Koza, 1992, глава 25]. С другой стороны, эти оценивания приспособленности будут быстрее, поскольку программы

будут меньше. Этот подход существенно смещает отбор в сторону более коротких программ. Данная идея получила название парсимонического давления, бритвы Оккама и минимальной длины описания [Langdon и Poli, 2002, глава 11]. Еще одним подходом к штрафованию за длинные программы является метод Тарпейской скалы [Poli, 2003], который принимает нулевую селекционную вероятность случайно отобранных программ, чей размер длиннее средней, равной нулю. По мере того как мы изменяем частоту, с которой это делается, мы корректируем способность метода противостоять раздуванию. Этот метод имеет дополнительное преимущество в том, что сокращается время исполнения программы, так как программы с нулевой селекционной вероятностью не нужно вычислять.

Есть и другие способы борьбы с раздуванием программного кода, такие как использование многокритериальной оптимизации с двумя целевыми критериями – приспособленность программы и длина программы (см. главу 20, автоматическое удаление избыточного кода и использование автоматически определяемых функций (ADF)). Эти методы являются более сложными и не гарантируют предотвращения от раздувания программного кода [Langdon и Poli, 2002, глава 11].

Наконец, мы упомянем, что в некоторых обстоятельствах раздувание программного кода может быть полезно. Раздувание имеет биологическую аналогию, и это может помочь компьютерным программам защитить своих детей от последствий вредного скрещивания [Angeline, 1996b], [Nordin и соавт., 1996]. Это дает начало термину *эффективная приспособленность,* который показывает не только то, насколько приспособленной является компьютерная программа, но и насколько приспособленными, скорее всего, окажутся ее дети [Banzhaf и соавт., 1998, глава 7]. Приспособленная родительская компьютерная программа, скорее всего, не сможет производить приспособленных детей, если родительская программа хрупкая, но приспособленный родитель с большим объемом раздувшегося кода может с большей вероятностью произвести приспособленных детей. Рассмотрим операцию скрещивания между двумя компьютерными программами. Хорошая программа с большим объемом неиспользуемого кода, скорее всего, приведет к приспособленным детям после скрещивания, потому что точка скрещивания, скорее всего, произойдет в неиспользуемой части родителя.

Можно также поговорить об *эффективной сложности*. Абсолютная сложность компьютерной программы – это функция ее длины и структуры, в то время как ее эффективная сложность – это функция длины и структуры нераздутой части (то есть активной части) программы.

7.5. Эволюционирующие сущности помимо компьютерных программ

Достижения в области генетического программирования впечатляют. В публикации [Koza, 1992] приводятся примеры генетических программ для обнаружения тригонометрических тождеств, открытия научных законов, решения математических уравнений в символической форме, индуцирования символической формы последовательности и поиска программ для сжатия изображений. Также приводятся примеры задач управления, включая балансировку перевернутого маятника на движущейся тележке (так называемая задача стабилизации перевернутого маятника) и задний ход тягача с прицепом.

Но генетическое программирование до сих пор не получило широкого распространения, по сравнению с другими эволюционными алгоритмами. Существуют различные причины этого отставания [Koza, 1992, глава 1].

1. Инженеры натренированы находить правильные решения задач. В школе часто существует только один правильный ответ на задачу домашнего задания. Генетическое программирование находит решения, которые являются лишь приближенно правильными, и это сдерживает его использование на практике. Однако в реальном инженерном мире все наши решения являются приближенными, если по какой-либо другой причине ничего нет, кроме допущений, которые мы принимаем (явным или неявным образом) при выводе наших решений. Отсутствие корректности является теоретическим препятствием для использования генетического программирования, но этот факт не обязательно должен быть практическим препятствием.
2. Инженеры натренированы находить решения путем постепенных улучшений. Генетическое программирование следует этому подходу, но оно также поощряет поиск в тупиках. Слабые программы должны пройти эволюционное развитие, прежде чем будут достигнуты хорошие программы. В реальном мире неудача – это клеймо позора, но наиболее успешные инженеры признают, что неудача является предпосылкой успеха [Petroski, 1992]. Это справедливо не только для человека, но и для генетического программирования.
3. Инженеры натренированы решать задачи дедуктивно. Мы знакомимся с задачей и строим решения шаг за шагом. Грубо го-

воря, в генетическом программировании существует некоторая дедукция; в конце концов, высоко приспособленные программы объединяются, чтобы получить программы с надеждой на более высокую приспособленность. Однако компьютерные программы, генерируемые генетической программой, не создаются логически путем добавления функциональности один шаг за раз.

4. Инженеры натренированы решать задачи детерминистски. Чем больше случайности мы можем удалить из нашей среды, тем больше контроля мы можем получить. Чем больше контроля мы можем получить, тем лучше мы можем продвинуться в нашем методе решения. Однако генетическое программирование, как и другие эволюционные алгоритмы, для отыскания хороших решений опирается на случайность.
5. Инженеры натренированы решать задачи экономно. Короткое и простое решение лучше, чем сложное. Однако генетическая программа эволюционно формирует компьютерные программы с ветвями, которые никогда не выполняются, с терминальными символами, которые не способствуют конечному результату, и с неэффективными структурами. Это как процессы решения задач, которые мы видим в природе. Многие животные обычно имеют сотни младенцев на каждого выжившего. Эволюция – это заведомо расточительный и неэффективный процесс.
6. Инженеры натренированы решать задачи с учетом специфических критериев успеха. У нас есть задачи, подзадачи, этапы и планы. Валидационные процессы сообщают нам о нашем успехе или неудаче. Однако генетическое программирование не имеет четко определенной точки останова. Этот вопрос более подробно обсуждался в разделе 7.2.2.

Многие из этих факторов относятся к эволюционным алгоритмам, помимо генетического программирования, но они, по-видимому, особо применимы к генетическому программированию. Разница в том, что эволюционные алгоритмы находят решения, тогда как генетические программы – методы решения. Мы, кажется, лучше способны терпеть небрежность в наших решениях, чем в наших методах решения. Когда мы находим хорошее решение задачи, мы часто не слишком беспокоимся о том, откуда пришло решение, покуда оно работает. Однако, когда мы находим метод решения, мы, вероятно, будем недоверчивы к такому методу, если мы его не понимаем, даже если он работает.

В этой главе мы увидели, что генетическое программирование может эволюционно формировать компьютерные программы. Вместе с тем компьютерные программы по-прежнему гораздо чаще пишутся людьми, чем генетическими программами. Это вызвано тем, что компьютерные программы, как правило, могут быть спланированы, структурированы и организованы настолько, насколько они поддаются пониманию экспертами-людьми. Компьютерные программы могут быть модуляризованы, а главные задачи могут быть разделены на подзадачи и назначены отдельным программистам. Крупномасштабные программные проекты имеют слишком много степеней свободы, чтобы можно было рассчитывать на успех генетического программирования, процесса, основанного на случайном поиске. Если бы даже генетическое программирование все-таки действительно преуспело, результирующая программа, возможно, была бы неэффективной и трудной в обслуживании. Одним словом, простые задачи компьютерного программирования слишком просты для генетического программирования, потому что люди могут выполнять такие задачи без особых усилий; но сложные задачи компьютерного программирования для него слишком сложны и, следовательно, требуют человеческой изобретательности. Это поднимает важные вопросы относительно генетического программирования. Какие типы задач подходят для генетического программирования, которые действительно трудны для людей? Есть ли у генетического программирования какие-либо практические приложения?

Для обсуждения этих вопросов мы рассмотрим области применения эволюционных алгоритмов. Эволюционные алгоритмы хорошо умеют находить решения сложных многомерных задач мультимодальной оптимизации. Компьютерное программирование может быть сложным, многомерным и мультимодальным. Однако компьютерное программирование – это задача, в которой преуспевают многие люди. Оптимизация параметров – это задача, в которой люди не преуспели. Практически все нетривиальные задачи оптимизации параметров решаются с помощью компьютерных программ. Мы видим, что эволюционные алгоритмы получили широкое распространение, потому что они преуспевают в задачах, трудных для человека.

Поскольку многие люди являются квалифицированными программистами, генетическое программирование вряд ли будет широко применяться в реальных задачах компьютерного программирования. Тем не менее генетическое программирование может широко применяться к задачам, подобным компьютерному программированию, в котором люди не преуспевают. Существует ряд инженерных (и других) задач,

кандидатные решения которых могут быть представлены в виде древовидных структур и в которых люди не преуспевают. К таким задачам можно отнести проектирование систем линз [Koza и соавт., 2008], фотонных кристаллов [Preble и соавт., 2005], алгоритмов классификации протеина [Koza, 1997], клеточных автоматов [Андре и соавт., 1996], алгоритмов нахождения численного решения сложных уравнений [Balasubramaniam и Kumar,, 2009], алгоритмов решения головоломок и отыскания игровых стратегий [Hauptman и Sipper, 2007], [Hauptman и соавт., 2009], алгоритмов электрических цепей [McConaghy и соавт., 2008], программируемых пользователем вентильных матриц [Koza и соавт., 1999] и антенн [Lohn et al, 2004].

Ключевой признак этих задач состоит в том, что они не могут быть легко сведены к задачам параметрической оптимизации, и поэтому они не могут быть решены с помощью генетических алгоритмов, эволюционных стратегий, эволюционного программирования или аналогичных методов. Они могут быть решены с помощью генетического программирования, которое эволюционно формирует алгоритм или эволюционно формирует программу проекта. Результаты генетического программирования часто улучшаются на существующих патентах, используя при этом очень мало специфичной для конкретной задачи информации [Koza, 2010]. В настоящее время термин «компьютерный интеллект» используется так много, что он уже мало что значит; он стал почти бессодержательным заезженным жаргонизмом. Но когда компьютерная программа создает патентоспособное изобретение, многие, вероятно, согласятся с тем, что программа обладает значительной степенью интеллекта.

7.6. Математический анализ генетического программирования

Мы можем математически анализировать генетические программы, как и другие эволюционные алгоритмы (см. главу 4). Этот раздел расширяет теорию схем генетических алгоритмов (раздел 4.1) до генетического программирования [Langdon и Poli, 2002, глава 4]. Наш основной подход в этом разделе заключается в использовании простых примеров, позволяющих получить общее представление о том, как теория схем работает в генетическом программировании, а затем использовать эти примеры, для того чтобы представить общую формулу схем для генетического программирования.

7.6.1. *Определения и обозначения*

В качестве нашего первого примера предположим, что наше множество терминальных символов равняется {*x*, *y*}. Мы используем # для обозначения терминального символа «все равно какой». Рассмотрим схему

$$H = ((+ (- \# y) \#). \tag{7.13}$$

Если *x* и *y* являются единственными двумя доступными терминальными символвами, то эта схема имеет четыре экземпляра:

$$\begin{aligned} &(+ (- x\ y)\ x), \quad (+ (- x\ y)\ y), \\ &(+ (- y\ y)\ x), \quad (+ (- y\ y)\ y). \end{aligned} \tag{7.14}$$

Мы говорим, что схема соответствует S-выражению, если S-выражение является экземпляром схемы. Например, схема (+ # *y*) соответствует (+ *x y*), а также соответствует (+ 2 *y*), но она не соответствует (– *x y*).

Теперь мы определим три важных термина, связанных со схемами для генетического программирования.

1. Порядок *H*, *o*(*H*), – это число определенных символов в *H*, включая и функции, и терминальные символы. В уравнении (7.13) *o*(*H*) = 3.
2. Длина *H*, *n*(*H*), – это общее число символов в *H*, включая определенные функции и терминальные символы, а также функции и терминальные символы «все равное какие». В уравнении (7.13) *n*(*H*) = 5.
3. Определяющая длина *H*, *L*(*H*), – это минимальное число связей в синтаксическом поддереве, которое включает все определенные символы. Определение длины трудно установить непосредственно из S-выражения, но его можно легко увидеть, посмотрев на соответствующее синтаксическое дерево.

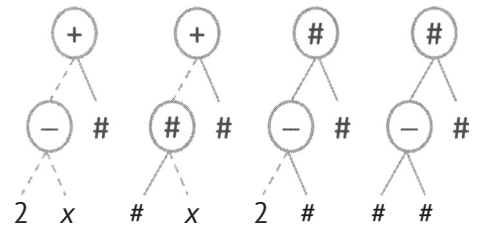
На рис. 7.14 показаны некоторые примеры синтаксических деревьев, их порядки, длины и определяющие длины.

Теперь рассмотрим, сколько схем соответствует S-выражению длины *n*. В качестве примера рассмотрим S-выражение

$$((+ (- 2\ x)\ (* 3\ y)). \tag{7.15}$$

Схема может соответствовать этому S-выражению с функцией + или символом # в верхнем узле. Аналогичную инструкцию можно сделать для всех остальных узлов в S-выражении. Следовательно, схема соответствует S-выражению, если схема имеет либо конкретный символ S-выражения, либо символ # в каждом узле. Мы видим, что существует

2^n схем, которые соответствуют S-выражению длины n. Например, существует 27 = 128 схем, которые соответствуют S-выражению уравнения (7.15).



(+ (–2*x*)#)	(+ (##*x*)#)	(#(–2#)#)	(#(–##)#)
L = 3	*L* = 2	*L* = 1	*L* = 0
o = 4	*o* = 2	*o* = 2	*o* = 1
n = 5	*n* = 5	*n* = 5	*n* = 5

Рис. 7.14. Четыре схемы генетической программы в виде синтаксического дерева и в форме S-выражения. Под каждой схемой показаны определяющая длина *L*, порядок *o* и длина *n*. Связи, которые используются для определения определяющей длины, находятся в наименьшем поддереве, которое включает все символы, отличные от #; эти связи показаны в виде пунктирных линий

Теперь определим структуру схемы. Мы используем *G* для обозначения структуры схемы и получаем *G*, заменяя все символы в *H* на символы #. Например, схема уравнения (7.13) имеет структуру:

$$\mathrm{G} = ((\#\ (\#\ \#\ \#)\ \#). \tag{7.16}$$

7.6.2. *Отбор и скрещивание*

Рассмотрим генетическую программу, в которой используются рулеточный отбор и скрещивание. Мы используем $m(H, t)$ для обозначения числа экземпляров схемы H в популяции генетической программы в t-м поколении. Мы применяем $m(H, t + 1/2)$ для обозначения числа экземпляров схемы в популяции после отбора. Тогда $m(H, t + 1)$ – это число экземпляров схемы после отбора, скрещивания и мутации. Если мы используем рулеточный отбор, то в среднем

$$m(H, t + 1/2) = m(H, t)\, f(H, t) / f_{\mathrm{ave}}(t), \tag{7.17}$$

где $f(H, t)$ – это средняя приспособленность всех экземпляров H в t-м поколении, $f_{\mathrm{ave}}(t)$ – средняя приспособленность всех особей в t-м поколении.

Теперь рассмотрим влияние скрещивания на популяцию. Скрещивание вполне может разрушить экземпляр H, то есть если родитель является экземпляром H, то он вполне может скреститься и создать детей, из которых ни один не является экземпляром H. В результате это приведет к тому, что число экземпляров H в следующем поколении будет на один меньше. Скрещивание может разрушить экземпляр H двумя разными способами. Рассмотрим родителя p_1 – такого, что $p_1 \in H$ и $p_1 \in G$, где G – это структура H. Во-первых, экземпляр H вполне может быть разрушен скрещиванием p_1 с особью $p_2 \notin G$; мы называем это событие D_1. Во-вторых, экземпляр H вполне может быть разрушен скрещиванием p_1 с особью p_2 – такой, что $p_2 \in G$, но $p_2 \notin H$; мы называем это событие D_2. Поскольку D_1 и D_2 являются взаимоисключающими событиями, вероятность того, что скрещивание разрушит экземпляр H, задается формулой

$$\Pr(D) = \Pr(D_1) + \Pr(D_2). \tag{7.18}$$

■ Пример 7.1

На рис. 7.15 показан пример события D_1. Родители (+ (– 2 *x*) (– 3 *y*)) и (+ *x y*) отобраны для скрещивания друг с другом. Случайно отобранными точками скрещивания являются крайне левая функция вычитания в родителе 1 и терминальный символ в родителе 2. Скрещивание приводит к детям (+ *y* (– 3 *y*)) и (+ *x* (– 2 *x*)). Мы видим, что ни один ребенок не имеет такой же структуры, как у любого из родителей. Скрещивание разрушило все схемы: схему H_1, экземпляром которой является родитель 1, и схему H_2, экземпляром которой является родитель 2.

■

■ Пример 7.2

Рассмотрим возможность события D_2 на примере. Рассмотрим схему H = (# *x y*). Предположим, что два родителя имеют

$$\begin{aligned} p_1 &= (+\ x\ y) \in H \\ p_2 &= (-\ y\ x) \notin H \end{aligned} \tag{7.19}$$

Оба родителя имеют одинаковую структуру G, но p_1 принадлежит схеме H, а p_2 – нет. Если точки скрещивания отобраны в верхней связи в p_1 и p_2, то детьми будут

$$\begin{aligned} c_1 &= (+\ y\ x) \in H \\ c_2 &= (-\ x\ y) \in H \end{aligned} \tag{7.20}$$

Мы видим, что экземпляр H для следующего поколения сохраняется.

Теперь рассмотрим схему $H = (+\ x\ \#)$ и тех же двух родителей, как показано в уравнении (7.19). Как и прежде, $p_1 \in H$ и $p_2 \notin H$. Однако в этом случае ни c_1, ни c_2 из уравнения (7.20) не принадлежат H, поэтому экземпляр H разрушается.

■

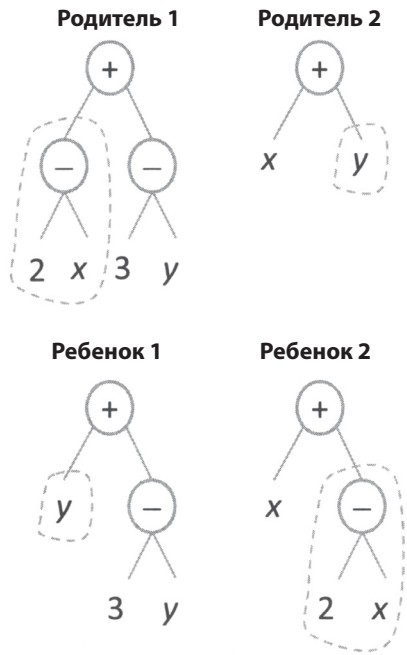


Рис. 7.15. Скрещивание между этими родителями приводит к детям, которые не имеют структуру ни одного из родителей. Это пример события D_1, при котором скрещивание между особями с разными структурами приводит к разрушению схем

Теперь рассмотрим влияние D_1 на число экземпляров схемы в популяции. Напомним, что D_1 является разрушением экземпляра схемы H, который имеет структуру G, из-за скрещивания между родителем p_1 – таким, что $p_1 \in H$ и $p_1 \in G$, – с родителем $p_2 \notin G$. То есть

$$D_1 = D \cap (p_2 \notin G), \tag{7.21}$$

где D – это событие, при котором экземпляр H разрушается из-за скрещивания. Теорема Байеса говорит нам, что

$$\Pr(D_1) = \Pr(D \mid p_2 \notin G) \Pr(p_2 \notin G). \tag{7.22}$$

Но вероятность того, что $p_2 \notin G$, пропорциональна числу особей в популяции, которые не принадлежат к G после отбора; то есть

$$\Pr(p_2 \notin G) = (N - m(G, t + 1/2) / N, \tag{7.23}$$

где N – это размер популяции. Объединив это с уравнением (7.22), получим

$$\Pr(D_1) = \Pr(D \mid p_2 \notin G) \frac{N - m(G, t+1/2)}{N}. \tag{7.24}$$

Теперь рассмотрим вероятность события H_2. Напомним, что D_2 является разрушением экземпляра схемы H, который имеет структуру G, из-за скрещивания между родителем p_1 – таким, что $p_1 \in H$ и $p_1 \in G$, – с родителем $p_2 \in G$. То есть

$$\begin{aligned} D_2 &= D \cap (p_2 \in G) \\ &= D \cap (p_2 \in G) \cap (p_2 \notin H), \end{aligned} \tag{7.25}$$

где второе равенство исходит из того факта, что разрушение схемы не может произойти, если не будет истинным $p_2 \notin H$. Теорема Байеса говорит нам, что

$$\Pr(D_2) = \Pr(D \mid p_2 \in G, p_2 \notin H) \Pr(p_2 \in G, p_2 \notin H). \tag{7.26}$$

Второй член в правой части уравнения (7.26) является вероятностью того, что $p_2 \in G$ и $p_2 \notin H$. Однако событие $p_2 \in H$ является подмножеством $p_2 \in G$, поэтому

$$\Pr(p_2 \in G, p_2 \notin H) = \Pr(p_2 \in G) - \Pr(p_2 \in H), \tag{7.27}$$

как показано на рис. 7.16.

Вероятности в правой части уравнения (7.27) соответственно пропорциональны числу экземпляров G и H после отбора; поэтому

$$\Pr(p_2 \in G, p_2 \notin H) = \frac{m(G, t+1/2) - m(H, t+1/2)}{N}. \tag{7.28}$$

Теперь рассмотрим первый член в правой части уравнения (7.26), то есть вероятность D, при условии что $p_2 \in G$ и $p_2 \notin H$. Экземпляр схемы может быть разрушен только в том случае, если скрещивание находится в одной из определяющих связей схемы. Например, на рис. 7.14 схема

слева будет разрушена только в том случае, если скрещивание происходит в одной из трех пунктирных связей; это верно для всех схем, изображенных на рисунке. Даже если скрещивание происходит в одной из этих связей, схема, возможно, не будет разрушена, в зависимости от содержимого поддерева, пристыкованного к экземпляру схемы. Поскольку есть $L(H)$ связей, в которых должно произойти скрещивание для экземпляра разрушаемой схемы, и есть всего $n(H) - 1$ связей, существует вероятность $L(H)/(n(H) - 1)$, что скрещивание произойдет в одной из определяющих связей. Поскольку скрещивание на одной из этих связей является необходимым, но не достаточным условием для разрушения экземпляра схемы, вероятность D, при условии что $p_2 \in G$, ограничена этой вероятностью сверху; то есть

$$\Pr(D \mid p_2 \in G, p_2 \notin H) \leq \frac{L(H)}{n(H)-1}. \tag{7.29}$$

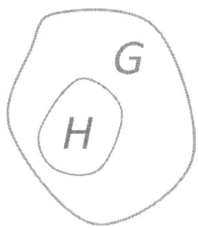


Рис. 7.16. Множество программ, принадлежащих схеме H, является подмножеством множества программ, принадлежащих схеме G, Следовательно, $\Pr(p \in G, p \notin H) = \Pr(p \in G) - \Pr(p \notin H)$

Правая часть приведенного выше уравнения называется хрупкостью узловой композиции схемы H [Langdon и Poli, 2002, раздел 4.4]. Оно дает верхнюю границу вероятности того, что экземпляр схемы H будет разрушен, если другой родитель имеет ту же структуру, что и H.

Объединив уравнения (7.18), (7.24), (7.28) и (7.29), получив верхнюю границу для вероятности разрушения схемы H:

$$\Pr(D) \leq \Pr(D \mid p_2 \notin G)\frac{N - m(G,t+1/2)}{N} + \frac{L(H)}{n(H)-1} + \frac{m(G,t+1/2) - m(H,t+1/2)}{N}, \tag{7.30}$$

в результате получим общее выражение вероятности разрушения экземпляра схемы H из-за скрещивания.

Учитывая тот факт, что скрещивание происходит с вероятностью p_c, мы принимаем допущение, что популяция генетической программы достаточно велика, чтобы использовать закон больших чисел [Grinstead

и Snell, 1997] для $m(H, t + 1)$, то есть числа экземпляров H в поколении $t + 1$. Это число задается суммой двух членов: (1) числа экземпляров H после отбора, умноженного на вероятность того, что скрещивание не произойдет, и (2) числа экземпляров H после отбора, умноженного на вероятность скрещивания, умноженную на вероятность того, что скрещивание не разрушит экземпляр H. Это дает

$$\begin{aligned} m(H, t + 1) &= m(H, t + 1/2)(1 - p_c) + m(H, t + 1/2)p_c(1 - \Pr(D)) \\ &= m(H, t + 1/2)(1 - p_c\Pr(D)), \end{aligned} \tag{7.31}$$

где $m(H, t + 1/2)$ показано в уравнении (7.17). Уравнение (7.31) дает приближенное число экземпляров схемы после скрещивания.

7.6.3. *Мутация и конечные результаты*

Теперь рассмотрим вероятность разрушения схемы из-за мутации. Предположим, что вероятность мутации в каждом узле равна PM. Тогда вероятность того, что мутация не произойдет в каждом узле, равна $1 - p_m$. Вероятность того, что мутация не произойдет ни в одном из определенных узлов, – $(1 - p_m)^{o(H)}$, где $o(H)$ – это порядок схемы (то есть число определенных узлов). Таким образом, вероятность того, что мутация произойдет в определенном узле, равна

$$\begin{aligned} \Pr(D_m) &= 1 - (1 - p_m)^{o(H)} \\ &\approx p_m o(H), \end{aligned} \tag{7.32}$$

где аппроксимация основана на разложении в ряд Тейлора $\Pr(D_m)$ вокруг $p_m = 0$. Обратите внимание, что мутация в определенном узле является необходимым, но недостаточным условием для разрушения схемы. Поэтому мы объединим уравнения (7.31) и (7.32) и получим

$$m(H, t + 1) \geq m(H, t + 1/2)(1 - p_c\Pr(D))\,(1 - p_m o(H)). \tag{7.33}$$

Мы объединим эти уравнения с уравнениями (7.17) и (7.30) и теперь получим

$$\begin{aligned} m(H, t + 1) \geq\ & \frac{m(H,t)f(H,t)}{f_{\text{ave}}(t)}\left[1 - p_m o(H)\right] \times \\ & \left\{1 - p_c\left[\Pr(D \mid p_2 \notin G)\left(1 - \frac{m(G,t)f(G,t)}{N f_{\text{ave}}(t)}\right) + \right.\right. \\ & \left.\left. \frac{L(H)}{n(H)-1}\,\frac{m(G,t)f(G,t) - m(H,t)f(H,t)}{N f_{\text{ave}}(t)}\right]\right\}. \end{aligned} \tag{7.34}$$

В результате этого мы получим нижнюю границу для числа экземпляров схемы H в поколении $t + 1$, где мы приняли во внимание и скрещивание, и мутацию. Как и ожидалось, это немного сложнее, чем теория схем в генетическом алгоритме, которую мы получили в разделе 4.1. Генетическое программирование сложнее генетического алгоритма из-за переменных размера и формы особей генетической программы. Однако мы можем сделать несколько упрощений в уравнении (7.34). Например, в начале выполнения генетической программы у нас широкое многообразие в популяции, поэтому маловероятно, что две особи с разными формами скрестятся в точке, которая приводит к сохранению схемы. Следовательно, в начале прогона генетической программы $\Pr(D \mid g \notin G) \approx 1$. Кроме того, при большом многообразии общая приспособленность особей, принадлежащих к структуре G, будет небольшой просто потому, что будет небольшое число таких программ; то есть $m(G, t) f(G, t)/(Nf_{\text{ave}}) \ll 1$. Следовательно, уравнение (7.34) может быть аппроксимировано в начале выполнения генетической программы как

$$m(H, t+1) \geq \frac{m(H,t) f(H,t)}{f_{\text{ave}}(t)} \left[1 - p_m o(H)\right] \times \left\{1 - p_c \left[1 + \frac{L(H)}{n(H)-1}\left(\frac{-m(H,t) f(H,t)}{N f_{\text{ave}}(t)}\right)\right]\right\}. \quad (7.35)$$

В результате получим аппроксимацию уравнения (7.34) в начале выполнения генетической программы. Мы можем выполнить дальнейшую аппроксимацию для короткой схемы, и в этом случае $L(H)/(n(H) - 1) \ll 1$:

$$m(H, t+1) \geq \frac{m(H,t) f(H,t)}{f_{\text{ave}}(t)} \left[1 - p_m o(H)\right](1 - p_c). \quad (7.36)$$

Данная аппроксимация схемы генетической программы, которая действительна для короткой схемы в начале выполнения генетической программы, аналогична выражению схемы генетической программы в уравнении (4.6). Теория схем, которую мы вывели в этом разделе, дает не равенство, а нижнюю границу для $m(H, t + 1)$. Это вызвано тем, что в нашем преобразовании мы применили аппроксимации. Поэтому данная теория схем называется пессимистической.

К теории схем можно подходить по-разному. Другие способы моделирования отбора, скрещивания и мутации в генетическом программировании дают теории схем, которые отличаются от теории, полученной в этом разделе. Некоторые из этих теорий являются точными, а не

пессимистичными, и в этом случае мы получаем уравнение вместо неравенства, такого как уравнение (7.34) [Altenberg, 1994], [Langdon и Poli, 2002, главы 3, 5]. Уна-Мэй О'Рейли разработал теорию схем с нижней границей, опираясь на работу Дж. Коза, которая определяла схемы иначе, чем в этой главе [Koza, 1992], [O'Reilly и Oppacher, 1995]. Юстиниан Роска разработал теорию схем генетического программирования, используя еще одно определение схемы [Rosca, 1997]. Теория схем также была разработана для систем генетического программирования, которые используют программные представления, отличные от синтаксических деревьев [Whigham, 1995].

Помимо теории схем, есть и другие методы математического анализа генетического программирования. Например, в 2001 году для генетического программирования были введены марковские модели [Poli и соавт., 2001], [Poli и соавт., 2004]. Кроме того, для математического моделирования генетического программирования может быть применена теорема отбора и ковариации Прайса [Langdon и Poli, 2002, глава 3].

7.7. Заключение

Эта глава была ограничена генетическим программированием (ГП) с использованием языка Lisp и синтаксических деревьев. Генетическое программирование также было реализовано с помощью многих других структур и на многих других языках. Например, программы могут быть представлены в виде линейной последовательности инструкций, являющейся форматом программирования, к которому большинство из нас привыкло в повседневной жизни. Этот формат называется линейным генетическим программированием [Poli и соавт., 2008, глава 7], которое в особенности подходит для программ на языке ассемблера. Эволюционное формирование программы на языке ассемблера для встраиваемой системы с использованием древовидного синтаксиса трудноосуществимо, потому что в этом случае нам нужно сначала создать компилятор с трансляцией по типу «дерево–ассемблер». Однако если мы эволюционно формируем программы непосредственно с использованием ассемблерного кода, оценивание стоимости выполнить гораздо проще.

Декартово (прямоугольное) генетическое программирование – это способ представления программ в виде множества массивов. Каждый массив содержит элемент, который определяет операцию этого массива, и элементы, которые определяют массивы, из которых следует получать входные данные [Miller и Smith, 2006]. Графовое генетическое

программирование – это способ представления программ с помощью вершин и ребер [Poli и соавт., 2008]. Для представления компьютерных программ в генетическом программировании также использовались и другие различные структуры [Banzhaf и соавт., 1998, глава 9].

Генетическое программирование получает все более широкое распространение, поскольку его область расширяется от компьютерного программирования к более общей эволюции инженерных алгоритмов и проектов. Дж. Коза перечисляет 76 результатов применения генетического программирования, которые конкурируют с результатами, генерируемыми человеком [Koza, 2010]. Он также утверждает, что скорость производства результатов, полученных с помощью генетического программирования, которые конкурируют с результатами человека, пропорциональна вычислительной мощности. В будущем этот факт предвещает резкое увеличение инженерных проектов, которые генерируются с помощью компьютерных программ, а также в значимом сотрудничестве между людьми и компьютерами.

Существует ряд задач, для которых генетическое программирование не подходит. Поисковым пространством в генетической программе является множество всех компьютерных программ в пределах, определяемых пользователем синтаксических ограничений. Эта широкая поисковая область является как сильной, так и слабой стороной генетического программирования. Широкая поисковая область позволяет генетической программе тщательнее искать оптимумы, чем это делается в других эволюционных алгоритмах, но также указывает на то, что в генетическом программировании обычно не используется много информации о предметной области от человека-программиста. В общем случае, если мы знаем структуру решения раньше времени, более стандартный эволюционный алгоритм должен превзойти генетическую программу, потому что специфичную для конкретной задачи информацию можно с легкостью встроить в задачу параметрической оптимизации, чем в задачу программной оптимизации. Однако если обнаружение структуры решения является серьезной проблемой оптимизационной задачи, то генетическая программа, возможно, будет адекватным подходом. Кроме того, обратите внимание, что мы можем посеять исходную популяцию генетической программы известными хорошими кандидатными решениями. Затем генетическая программа будет улучшать эти решения по мере поступательного развития. Следовательно, задачи, для которых постепенное улучшение существующих решений крайне желательно, также очень подходят для генетического программирования [Koza, 2010].

Интересным направлением будущей работы является генерация компьютерных программ, эволюционно формирующих компьютерные программы, которые, в свою очередь, эволюционно формируют компьютерные программы. Это направление можно было бы назвать генетическим метапрограммированием. Генетическая программа может эволюционно формировать компьютерные программы, но как обнаружить лучшую генетическую программу? Возможно, что генетическое метапрограммирование сможет сформировать генетическую программу. Генетическое метапрограммирование, вероятно, как минимум учетверит вычислительную мощность, требуемую для одной генетической программы. Юрген Шмидхубер впервые предложил генетические метапрограммы в своей диссертации 1987 года [Schmidhuber, 1987]. Генетические метапрограммы – это тип метасамообучения (то есть самообучения тому, как учиться) [Anderson и Oates, 2007]. Генетическое метапрограммирование можно рассматривать как поиск поиска, таким образом, оно попадает в категорию вертикальной теоремы об отсутствии бесплатных обедов [Dembski и Marks, 2010],

Обратите внимание, что генетическое программирование можно комбинировать с другими эволюционными алгоритмами. Например, мы можем объединить генетическое программирование и алгоритм оценивания вероятностного распределения (EDA) для нахождения вероятностных описаний эффективных программ, которые, в свою очередь, могут направлять наш поиск лучших программ. Это было впервые предложено под именем вероятностной инкрементной эволюционной программы [Salustowicz и Schmidhuber, 1997]. В этом алгоритме каждый узел в синтаксическом дереве имеет вероятность быть равным определенной функции или терминальному символу, и эти вероятности зависят от значений приспособленности отдельных программ. Новые эволюционные алгоритмы предлагаются в литературе довольно часто, и было бы интересно изучить, какие из этих новых эволюционных алгоритмов в особенности подходят для реализации в качестве генетической программы (см. главу 17).

Наконец, отметим, что серьезный исследователь генетического программирования должен овладеть искусством автоматически определяемых функций (ADF). Автоматически определяемые функции – это подпрограммы, которые автоматически и динамически эволюционируют в генетической программе. Учитывая тот факт, что программист-человек естественным образом использует подпрограммы, имеет смысл в том, что генетическая программа тоже должна создавать и использовать подпрограммы. Автоматически определяемые функции

могут значительно уменьшить вычислительные усилия генетического программирования при их применении к сложным задачам. Автоматически определяемые функции подробно обсуждаются в публикациях [Koza, 1994, глава 4], [Koza и соавт., 1999] и в других книгах, посвященных генетическому программированию.

Для дальнейшего изучения генетического программирования читатель может найти несколько отличных книг, посвященных данной теме. Книга Вольфганга Банжафа и других соавторов является легким для восприятия введением [Banzhaf et al, 1998]. Исчерпывающие издания Джона Козы представляют собой стандартный справочный материал по данной области, который безусловно заслуживает своей звездной репутации, в особенности его первая книга [Koza, 1992], [Koza, 1994], [Koza и соавт., 1999], [Koza и соавт., 2005]. Полевой справочник по генетическому программированию (Field Guide to Genetic Programming) является свободно доступной книгой, которая обеспечивает хорошее общее представление о генетическом программировании [Poli и соавт., 2008]. Подробный анализ схем можно изучить в книге Уильяма Лэнгдона и Риккардо Поли [Langdon и Poli, 2002].

Задачи

Письменные упражнения

7.1. Напишите S-выражение и синтаксическое дерево для положительного решения квадратного уравнения: $(\sqrt{b^2 - 4ac} - b) / (2a)$. Какова глубина синтаксического дерева? Ваше синтаксическое дерево является полным?

7.2. Напишите S-выражение, которое возвращает 8, если $x > 2$, и 9 в противном случае.

7.3. Предположим, что вы оцениваете кандидатное решение генетической программы f на n разных входах $\{u_i\}$. Напишите функцию приспособленности, которая дает в два раза больше веса средней результативности f, чем результативности f для худшего случая.

7.4. Определите защищенную функцию квадратного корня на языке Lisp, которая возвращает 0, если входные данные отрицательные.

7.5. Предположим, что мы используем метод выращивания для генерации случайного S-выражения с максимальной глубиной D_c. Предположим, что в каждом узле существует 50%-ный шанс от-

бора функции или терминального символа. Какова вероятность того, что конкретная ветвь достигнет максимально возможной глубины?

7.6. Предположим, что мы используем метод выращивания для генерации случайного S-выражения с максимальной глубиной D_c. Предположим, что в каждом узле существует 50%-ный шанс отбора функции или терминального символа. Предположим, что каждая функция принимает два аргумента. Какова вероятность того, что S-выражение будет представлять собой полное синтаксическое дерево с глубиной D_c?

7.7. Перечислите все программы, которые могут быть созданы в результате подъемной мутации синтаксического дерева на рис. 7.1.

7.8. Перечислите уникальные программы, которые могут быть созданы путем перестановочной мутации синтаксического дерева на рис. 7.2.

7.9. Напишите следующее уравнение в упрощенном виде (например, (+ 1 1) можно заменить на 2). Получив упрощенный вариант, напишите уравнение в более привычном виде:

(defun Pgm (*x*) (+ (DIV (– (+ (+ 1 1) (+ (DIV *x* 1) (+ 1 1))) (– (* x 1) (+ 1 1))) (abs *x*)) (DIV (+ (– *x* 1) (abs *x*)) (– (– (* *x* 1) (+ 1 1)) (– (+ 1 1) (* 1 1)))))).

7.10. Какова определяющая длина, порядок и длина схемы (if (# *x* #) 8#)? Какова структура схемы?

7.11. Как варьируется нижняя граница числа экземпляров схемы вместе с вероятностью мутации? Как она зависит от порядка схемы? Как она варьируется вместе с вероятностью скрещивания? Объясните свои ответы.

Компьютерные упражнения

7.12. Упражнения на языке Lisp:

a) скачайте и установите последнюю версию языка CLISP (Common Lisp);

b) скачайте и установите интегрированную среду разработки (IDE) для языка CLISP. Примечание: задача управления объектом за минимальное время в разделе 7.3 была реализована в программе под названием LispIDE;

c) запустите среду Lisp IDE и в командной строке введите следующую строку:

```
(print (* 5 (+ 3 2)))
```

В результате Lisp напечатает 25 в вашем терминале дважды: один раз из-за команды print и второй раз из-за значения, возвращаемого функцией print.

7.13. Упражнение на управление объектом за минимальное время:

a) скачайте программу `GPCartControl.lisp` и связанные с ней файлы с веб-сайта книги и запустите ее на компьютере. Она продублирует результаты генетического программирования задачи управления за минимальное время из раздела 7.3. Если вы используете среду разработки LispIDE, то это можно сделать следующим образом.

- Запустить среду LispIDE.
- Открыть программу `GPCartControl.lisp` в среде LispIDE.
- Изменить строку 15 программы `GPCartControl.lisp`, так чтобы путь указывал на каталог вашего компьютера, который содержит файлы Lisp.
- Выбрать весь файл `GPCartControl.lisp` (используйте компьютерную мышь или **Ctrl-A**).
- Выбрать пункт меню **Edit** → **Send to Lisp** (Правка → Отправить в Lisp). Это определит функцию `GPCartControl` в Lisp.
- Набрать `(GPCartControl)` в командной строке среды LispIDE. В результате программа будет запущена;

b) после выполнения программы `GPCartControl.lisp` она выведет два файла. Один файл – `[Строка DateTime].txt` – будет содержать номер поколения, лучшую стоимость и среднюю стоимость. Другой файл – `[Строка DateTime].lisp` – будет содержать лучшую программу, найденную генетической программой (обратите внимание, что `[Строка DateTime]` – это текстовая строка, представляющая дату и время создания файла);

c) выполните команду Lisp (`setf LispPgm [Лучшая программа]`), где `[Лучшая программа]` – это текстовая строка, которая определяет лучшую программу, найденную генетической программой. Вы должны получить эту текстовую строку из `[DateTimeStringJ.lisp`. Например

(setf LispPgm

"(defun CartControl (*x v*) (if (> (* – ! *x*) (* *v* (abs *v*))) 1 – 1))")

d) выполните функцию Lisp `(PhasePlane LispPgm)`. В результате будет создано два файла. Один файл – `PhasePlane.txt` – будет представлять собой список `(x, v, control)` значений, сгенерированных программой `LispPgm`. Другой файл – `PhasePlane1.txt` – будет являться списком `(x, v)` значений, при которых управление переключается между –1 и +1. При этом предполагается, что управление, генерируемое программой `LispPgm`, всегда насыщенно. Если это допущение неверно, то файл `PhasePlane1.txt` будет бесполезен;

e) запустите программу на языке Matlab `PlotPhasePlane.m` с входной строкой «PhasePlane». Это позволит сгенерировать график фазовой плоскости управления в зависимости от *x* и *v*, используя сгенерированные выше файлы `PhasePlane.txt` и `PhasePlane1.txt`. Однако график будет бесполезным, если указанное выше допущение не было удовлетворено, то есть если сгенерированное программой LispPgm управление не является всегда насыщенным;

f) приспособленность программы управления тележкой на языке Lisp может быть оценена выполнением программы `EvalCartControl.lisp`. Если вы определяете `CartControl`, как описано выше в подзадаче (c), то вы можете открыть программу `EvalCartControl.lisp`, вычислить ее в среде Lisp IDE, чтобы `EvalCartControl` была определенной в Lisp функцией, а затем выполнить ее, введя следующую команду: `(EvalCartControl #' CartControl)`.

7.14. Модифицируйте некоторые параметры в программе `GPCartControl.lisp`, чтобы увидеть, какое влияние они оказывают на результативность. Среди параметров, которые вы можете модифицировать, следующие:

- `Dinitial`, максимальная исходная глубина дерева;
- `Dcreated`, максимальная глубина дерева;
- `Pcross`, вероятность скрещивания;
- `Preproduce`, вероятность размножения;
- `Pinternal`, вероятность скрещивания во внутренних (функциональных) узлах;
- `NumEvals`, число оцениваний функций на особь;

- `NumElites`, число элитарных особей в каждом поколении;
- `GenLimit`, лимит поколений;
- `PopSize`, размер популяции;
- `SelectionMethod`, метод отбора.

7.15. Модифицируйте программу `GPCartControl.lisp` и связанные с ней файлы, для того чтобы генетическая программа смогла найти отображение $y(x)$, которое близко совпадает с целью $y(x)$, где $y(x)$ задано следующим образом:

$$y(0) = 3, y(1) = 5, y(2) = 1, y(3) = 2, y(4) = 9$$
$$y(5) = 8, y(6) = 3, y(7) = 4, y(8) = 1, y(9) = 6.$$

Постройте график схождения генетической программы, показывающий развитие минимальной и средней стоимостей популяции как функций числа поколений, лучшей программы, найденной генетической программой, и постройте график, показывающий целевые значения $y(x)$, по сравнению с аппроксимированными генетической программой значениями $\hat{y}(x)$.

Глава 8

Вариации эволюционных алгоритмов

Существует большое число вариантов.

– Дэвид Фогель [Fogel, 2000]

В предыдущих главах обсуждались четыре популярных и основополагающих подхода к эволюционным вычислениям. Однако были представлены только основные идеи и алгоритмы. В тех алгоритмах мы можем реализовать целый ряд вариаций. Эти вариации применимы не только к эволюционным алгоритмам, рассмотренным в предыдущих главах, но и к алгоритмам, которые будут проанализированы в этой книге далее. Поэтому данная глава имеет большую применимость к широкому кругу эволюционных алгоритмов. Некоторые вариации, которые мы обсуждаем в этой главе, могут сильно влиять на результативность эволюционного алгоритма. Когда мы сравниваем два эволюционных алгоритма, алгоритмы *A* и *B*, часто решающим фактором, влияющим на их уровни результативности, является не существенная разница между *A* и *B*, а эти якобы незначительные вариации и детали реализации.

Краткий обзор главы

В разделе 8.1 обсуждаются разные способы инициализации эволюционно-алгоритмической популяции. В разделе 8.2 рассматриваются разные способы принятия решения о прекращении работы эволюционного алгоритма. В разделе 8.3 раскрывается вопрос представления кандидатных эволюционно-алгоритмических решений и то, как выбранное

представление может существенно влиять на результаты. В разделе 8.4 обсуждается элитарность, которая первоначально была предложена для генетических алгоритмов, но теперь обычно реализуется во всех эволюционных алгоритмах из-за присущих ей преимуществ. В разделе 8.5 рассматривается разница между поколенческими эволюционными алгоритмами и стационарными эволюционными алгоритмами.

Эволюционно-алгоритмические популяции часто имеют тенденцию сходиться к одной высоко приспособленной особи, то есть вся популяция становится клонами одного кандидатного решения. Это сильно снижает способность эолюционного алгоритма отыскивать оптимальное решение, поэтому в разделе 8.6 обсуждается вопрос поддержки видового многообразия в эволюционно-алгоритмической популяции.

До сих пор мы фокусировались на рудеточном отборе родителей. Однако есть и другие подходы к отбору, и мы обсудим некоторые из них в разделе 8.7. Одним из важных вариантов отбора является племенной отбор (раздел 8.7.7), который первоначально был предложен для генетических алгоритмов, но также может быть применен с другими эволюционными алгоритмами. В разделе 8.8 рассматриваются разные способы объединения родителей для получения детей, а в разделе 8.9 обсуждаются разные способы реализации мутации.

8.1. Инициализация

Обычно мы инициализируем эволюционный алгоритм случайной популяцией. Это простейший и популярный метод инициализации. Однако инициализация может существенно влиять на успех эволюционного алгоритма. Немного дополнительных усилий, потраченных на инициализацию, может принести большие дивиденды.

Предположим, мы хотим выполнить эволюционный алгоритм с N особями. Один из подходов к инициализации – сгенерировать более N особей и просто оставить лучших N в качестве нашей исходной популяции. Например, в публикации [Bhattacharya, 2008] создается $5N$ случайных особей и N в качестве исходной популяции.

Мы можем также случайно сгенерировать особей, а затем локально оптимизировать каждую особь, чтобы получить нашу исходную эволюционно-алгоритмическую популяцию. Например, мы можем сгенерировать N случайных особей, выполнить оптимизацию методом градиентного спуска для подмножества этих особей, а затем использовать получившихся особей в качестве исходной эволюционно-алгоритмической популяции. Мы можем реализовать это по-разному; например, в

нашей исходной популяции можно использовать только лучших особей, либо можно выполнить градиентный спуск лишь на лучших особях, но использовать всех особей в нашей исходной популяции.

Еще один вариант – применить для инициализации эволюционно-алгоритмической популяции экспертные решения. Например, предположим, что мы хотим применить эволюционный алгоритм для точной регулировки алгоритма управления. Мы можем применить экспертные знания, позволяющие выполнить оценивание разумных решений управления, и посеять исходную эволюционно-алгоритмическую популяцию этими решениями. Либо для того, чтобы посеять нашу исходную эволюционно-алгоритмическую популяцию, мы можем воспользоваться кандидатными решениями, которые находим из любого другого источника (иные алгоритмы, другие опубликованные результаты и т. д.). Использование зависимой от конкретной задачи информации для посева исходной эволюционно-алгоритмической популяции часто носит название направленной инициализации.

■ Пример 8.1

В данном примере рассматривается влияние дополнительных исходных особей на эволюционную программу для оптимизации 10-мерной функции Розенброка (см. приложение С.1.4). Мы используем размер популяции 10 и выполняем эволюционную программу для 50 поколений. В нашей стандартной реализации эволюционной программы мы случайно инициализируем 10 исходных особей. В нашей реализации дополнительных исходных особей мы случайно генерируем 20 особей и используем лучших 10 для нашей исходной популяции эволюционной программы. На рис. 8.1 приведены результаты двух алгоритмов, усредненно по 20 симуляциям Монте-Карло. Инициализация дополнительных особей требует вдвое больше вычислительных усилий на этапе инициализации, но дополнительные усилия приводят к гораздо более высокой результативности. Рисунок показывает, что если мы инициализируем дополнительных особей, то в течение первых 40 поколений получаем значительно более высокие результаты, хотя оба алгоритма демонстрируют примерно одинаковые результаты к тому времени, когда они достигают 50-го поколения.

■

Подход с использованием направленной инициализации или дополнительных исходных особей имеет смысл для задач с несколькими по-

колениями и несколькими особями, то есть задач, которые ограничены несколькими вычислениями функции приспособленности из-за их высокой стоимости (см. раздел 21.1). При таком типе задачи дополнительные усилия, затрачиваемые во время инициализации, могут иметь преимущества, которые длятся в течение многих поколений.

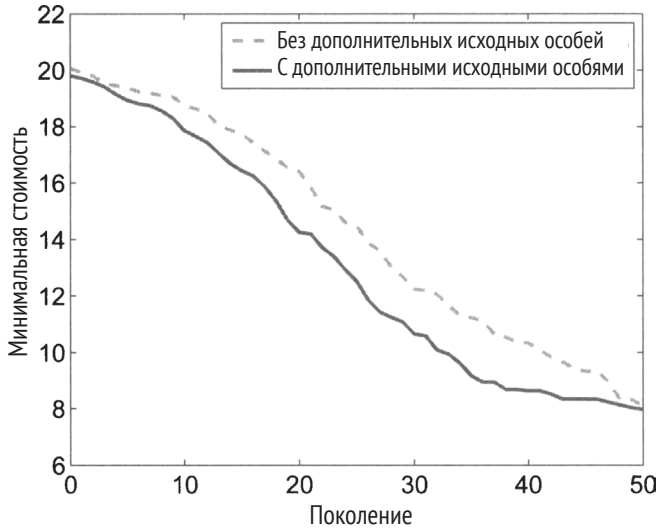


Рис. 8.1. Пример 8.1: стоимость в сопоставлении с номером поколения для эволюционной программы, которая минимизирует 10-мерную функцию Розенброка, усредненно по 20 симуляциям Монте-Карло. Дополнительная работа по созданию еще 10 исходных особей, по-видимому, стоит дополнительных усилий, по крайней мере в течение первых 40 поколений

8.2. Критерии сходимости

Одна вещь, которую мы должны решить, когда реализуем эволюционный алгоритм, – это когда остановиться. Мы замалчивали эту проблему в алгоритмах предыдущих глав, используя типичную инструкцию «While not (критерий останова)» (например, см. рис. 3.6, 5.1, 6.1 и 7.5). Какой критерий останова следует использовать? Как долго должен работать эволюционный алгоритм, прежде чем мы остановим программу? Для определения сходимости мы можем использовать несколько критериев.

1. Мы можем остановить эволюционный алгоритм после заданного числа поколений. Такой критерий имеет преимущество просто-

ты и предсказуемости, и, вероятно, это наиболее часто используемый критерий останова для эволюционных алгоритмов. Однако если мы сравниваем разные эволюционные алгоритмы, то должны останавливать эволюционные алгоритмы после заданного числа оцениваний целевой функции, а не заданного числа поколений. Это связано с тем, что в разных эволюционных алгоритмах используется разное число оцениваний функций на поколение, поэтому мы можем выполнить справедливое сравнение эволюционных алгоритмов, только завершив работу после достижения общего для них предела вычислений функций.

2. Мы можем остановить эволюционный алгоритм, после того как решение станет «достаточно хорошим». Такой критерий останова зависит от конкретной задачи, потому что критерий «достаточно хорошо» варьируется от одной задачи к другой. Данный критерий останова имеет свою привлекательность; если мы находим решение, которое обеспечивает удовлетворительную результативность, то зачем продолжать искать лучшее решение? Однако большинство решений реальных задач никогда не бывает достаточно хорошим. Мы всегда попытаемся найти решение, которое будет еще лучше. С другой стороны, удвоение времени выполнения при повышении результативности всего на мизерную величину во многих случаях является пустой тратой ресурсов. Компромисс между временем выполнения и результативностью является зависящей от конкретной задачи проблемой, которая требует инженерного суждения.
3. Мы можем остановить эволюционный алгоритм, после того как приспособленность лучшей особи перестанет изменяться в течение определенного числа поколений. Это может означать, что эволюционный алгоритм застрял в локальном минимуме. Локальный минимум также может оказаться глобальным минимумом, но мы никогда этого не узнаем, если не найдем более оптимальный локальный минимум.
4. Мы можем остановить эволюционный алгоритм, после того как средняя популяционная приспособленность не изменится заметно в течение определенного числа поколений. Данный критерий аналогичен вышеуказанному критерию; такая ситуация говорит о том, что популяция в целом перестала улучшаться.
5. Мы можем остановить эволюционный алгоритм, после того как стандартное отклонение популяционной приспособленности

перестанет уменьшаться или опустится ниже некоторого порога. Эта ситуация свидетельствует о том, что популяция достигла определенного уровня единообразия.

6. Мы можем применить некоторую комбинацию вышеуказанных критериев.

Пункты 3–5 подразумевают (но не гарантируют), что популяция больше не улучшается, поэтому мы с таким же успехом можем остановить эволюционный алгоритм. Однако такой подход сопряжен с опасностью. Даже когда эволюционный алгоритм, по-видимому, сошелся, статистически невероятное мутационное или рекомбинационное событие может привести к значительному улучшению решения, как показано на рис. 8.2. Мы не можем работать вечно; в конце концов, мы должны остановиться. Но независимо от того, когда остановимся, мы рискуем, что завершаем как раз перед особо удачной мутацией или рекомбинацией.

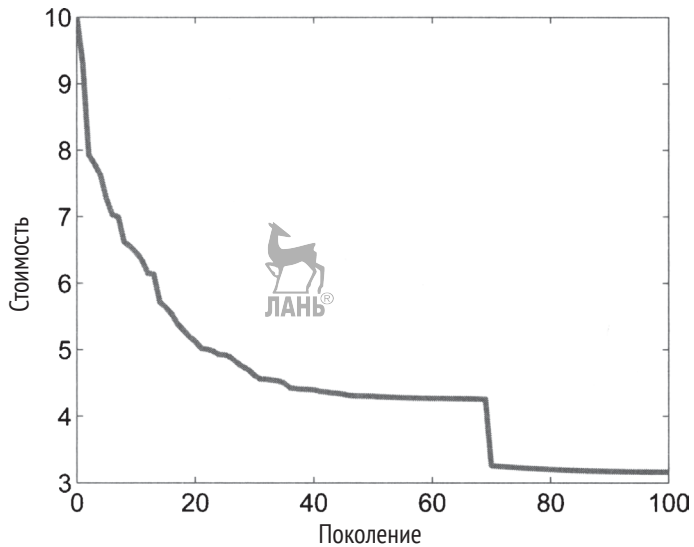


Рис. 8.2. График стоимости (лучшей или средней) в сравнении с номером поколения для симуляции гипотетического эволюционного алгоритма. Если мы прекратим работу эволюционного алгоритма, после того как стоимость перестанет улучшаться, то пропустим значительное улучшение в 70-м поколении, которое, возможно, будет вызвано статистически невероятной, но непредсказуемой мутацией. Однако если значение стоимости, равное 5, клиента удовлетворяет, то улучшение на 70-м поколении нас, скорее всего, волновать не будет

8.3. Представление задачи с помощью кода Грея

В этом разделе описывается, как реализовывать бинарные эволюционные алгоритмы с использованием кодирования Грея. Код Грея[1], так называемый отраженный двоичный код, представляет собой способ представления чисел таким образом, что коды соседних чисел различаются только одним битом [Doran, 2007]. Рассмотрим представление чисел 0–7 в двоичном коде:

$$\begin{aligned} &000 = 0, && 001 = 1,\\ &010 = 2, && 011 = 3,\\ &100 = 4, && 101 = 5,\\ &110 = 6, && 111 = 7. \end{aligned} \tag{8.1}$$

Мы видим, что двоичные коды для соседних чисел могут отличаться более чем на один бит. Например, код для 3 равен 011, и код для 4 равен 100. Двоичные коды для 3 и 4 отличаются во всех трех битах. Обратное также верно, то есть двоичные коды, которые очень похожи друг на друга, иногда представляют собой числа, которые очень далеки друг от друга. Например, 000 представляет число 0; если мы изменим один бит, получив код 100, то мы теперь будем иметь представление числа 4, которое далеко от 0 относительно диапазона чисел, которые мы представляем.

Теперь рассмотрим представление чисел 0–7 в кодах Грея:

$$\begin{aligned} &000 = 0, && 001 = 1,\\ &011 = 2, && 010 = 3,\\ &110 = 4, && 111 = 5,\\ &101 = 6, && 100 = 7. \end{aligned} \tag{8.2}$$

Мы видим, что коды Грея для соседних чисел всегда отличаются ровно на один бит. Например, код 3 равен 010, и код для 4 равен 110. Коды Грея для 3 и 4 отличаются только самым левым битом. Кодирование с помощью кодов Грея удаляет скалы Хэмминга, то есть большие изменения целочисленных значений между представлениями, которые отличаются только одним битом [Deep и Thakur, 2007].

■ Пример 8.2

Рассмотрим функцию $y = f(x)$ примера 2.2. Если мы закодируем значения $x \in [-4, +1]$ шестнадцати равноотстоящих значений из горизон-

[1] См. https://ru.wikipedia.org/wiki/Код_Грея. – *Прим. перев.*

тальной оси рис. 2.1 четырехбитным двоичным кодированием, то мы получим

двоичное кодирование:

$$\begin{aligned}
&0000 = -4.00, && 0001 = -3.67,\\
&0010 = -3.33, && 0011 = -3.00,\\
&0100 = -2.67, && 0101 = -2.33,\\
&0110 = -2.00, && 0111 = -1.67,\\
&1000 = -1.33, && 1001 = -1.00,\\
&1010 = -0.67, && 1011 = -0.33,\\
&1100 = +0.00, && 1101 = +0.33,\\
&1110 = +0.67, && 1111 = +1.00.
\end{aligned} \tag{8.3}$$

С другой стороны, мы можем также закодировать значения четырехбитным кодом Грея и в результате получим

кодирование Грея:

$$\begin{aligned}
&0000 = -4.00, && 0001 = -3.67,\\
&0011 = -3.33, && 0010 = -3.00,\\
&0110 = -2.67, && 0111 = -2.33,\\
&0101 = -2.00, && 0100 = -1.67,\\
&1100 = -1.33, && 1101 = -1.00,\\
&1111 = -0.67, && 1110 = -0.33,\\
&1010 = +0.00, && 1011 = +0.33,\\
&1001 = +0.67, && 1000 = +1.00.
\end{aligned} \tag{8.4}$$

Если мы построим график y против x таким образом, что соседние значения x отличаются в своих двоичных кодах на один бит, то мы получим верхний график на рис. 8.3. Если построить график y против x таким образом, чтобы соседние значения x отличались в своих кодах Грея на один бит, то получим нижний график. Мы видим, что при кодировании Грея график сохраняет свою первоначальную форму (сравните с рис. 2.1). Это позволяет упрощать оптимизацию гладких функций, поскольку небольшие изменения в кодах приводят к небольшим изменениям значений функций.

■

■ Пример 8.3

В данном примере мы тестируем влияние двоичного кодирования и кодирования Грея на результативность генетического алгоритма. Мы используем генетический алгоритм для оптимизации двумерной функции Экли, описанной в примере 3.3, где каждая размерность ко-

дируется шестью битами. Мы используем популяцию размером 20 и скорость мутации 2% на бит на поколение. Рисунок 8.4 показывает среднюю стоимость 20 генетико-алгоритмических особей в каждом поколении, усредненно по 50 симуляциям Монте-Карло. Кодирование Грея демонстрирует заметно более высокую результативность, чем двоичное, благодаря гладкой, регулярной поверхности функции Экли (см. рис. 2.5).

■

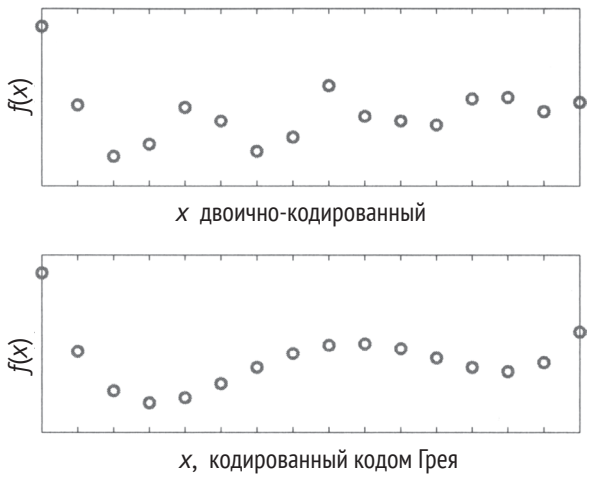


Рис. 8.3. Пример 8.2: в верхней части графика ось x расположена так, что однобитные изменения в x смежны, когда мы используем двоичное кодирование. Нижний график показывает то же самое, когда применяем кодирование Грея. Гладкая функция теряет свою гладкость, если мы используем двоичное кодирование

Хотя кодирование Грея, по-видимому, работает лучше в большинстве практических приложений, можно доказать, что кодирование Грея лучше работает в худших случаях [Whitley, 1999]. Худшим случаем является дискретная задача, для которой половина точек в поисковом пространстве является локальными минимумами. Наконец, стоит упомянуть и то, что в эволюционных алгоритмах мы можем использовать ряд других представлений, помимо двоичного кодирования и кодирования Грея. Применяемое нами представление может оказать значительное влияние на результативность эволюционного алгоритма, и поэтому, хотя в этой книге мы не рассматриваем представления подробно, их не

следует игнорировать в приложениях на основе эволюционных алгоритмов. Изучение представлений может оказаться весьма сложным, и мы отсылаем читателя к таким публикациям, как [Choi и Moon, 2003] и [Rothlauf и Goldberg, 2003], для дальнейшего изучения.

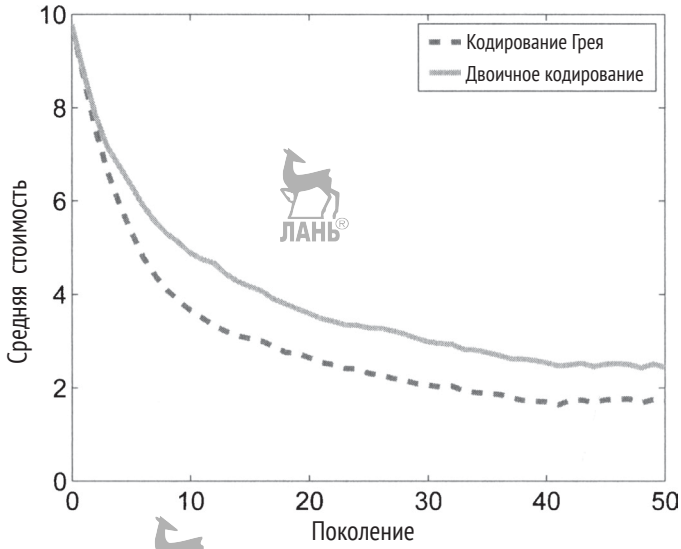


Рис. 8.4. Пример 8.3: результаты минимизации двумерной функции Экли, где каждая размерность кодируется шестью битами. График показывает среднюю стоимость всех генетико-алгоритмических особей в каждом поколении, усредненно по 50 симуляциям Монте-Карло. Генетический алгоритм с кодированием Грея работает заметно лучше, чем генетический алгоритм с двоичным кодированием

■ Пример 8.4

Предположим, что мы имеем задачу, соответствующую худшему случаю, для которой четные значения двоичного кода имеют стоимость 1, а нечетные значения – 2. Если особи представлены двоичными кодами, то значения стоимости для трехбитной задачи, соответствующей худшему случаю, будут

$$\begin{aligned} f(000) &= 1, \quad f(001) = 2, \\ f(010) &= 1, \quad f(011) = 2, \\ f(100) &= 1, \quad f(101) = 2, \\ f(110) &= 1, \quad f(111) = 2. \end{aligned} \tag{8.5}$$

Если особи представлены кодами Грея, то значения стоимости для трехбитной задачи, соответствующей худшему случаю, записанные в том же порядке, что и двоичные представления выше, будут:

$$\begin{aligned} f(000) &= 1, \quad f(001) = 2, \\ f(011) &= 1, \quad f(010) = 2, \\ f(110) &= 1, \quad f(111) = 2, \\ f(101) &= 1, \quad f(100) = 2. \end{aligned} \tag{8.6}$$

Если мы посмотрим на значения стоимостной функции уравнения (8.5), то увидим, что скрещивание между двоично-кодированными высоко приспособленными особями приведет к детям, которые тоже будут высоко приспособленными. Это вызвано тем, что все высоко приспособленные особи имеют 0 в крайне правой битной позиции, поэтому любые дети от высоко приспособленных особей тоже будут иметь 0 в крайне правой битной позиции. Это означает, что они будут четными, а значит, будут высоко приспособленными, как и их родители. Вместе с тем уравнение (8.6) показывает, что скрещивание между кодированными кодом Грея высоко приспособленными особями может привести к низко приспособленным детям. Этот простой пример дает нам интуитивное понимание того, что для задач со многими локальными минимумами двоичное кодирование, возможно, будет справляться лучше, чем кодирование Грея.

■

■ Пример 8.5

В данном примере мы тестируем результативность генетического алгоритма на 20-битной задаче, соответствующей худшему случаю, для которой четные значения двоичного кодирования имеют стоимость 1, а нечетные – 2. Мы используем популяцию размером 20 и скорость мутации 2 % на бит на поколение. На рис. 8.5 показана средняя стоимость 20 генетико-алгоритмических особей в каждом поколении, усредненно по 50 симуляциям Монте-Карло. Двоичное кодирование работает гораздо лучше, чем кодирование Грея. Это указывает на то, что в задачах со многими локальными минимумами при поиске широкого многообразия локальных минимумов двоичное кодирование, возможно, будет справляться лучше, чем кодирование Грея. Стоит запомнить, что во многих практических оптимизационных задачах мы

хотим найти многообразие хороших решений, а не только одно хорошее решение.

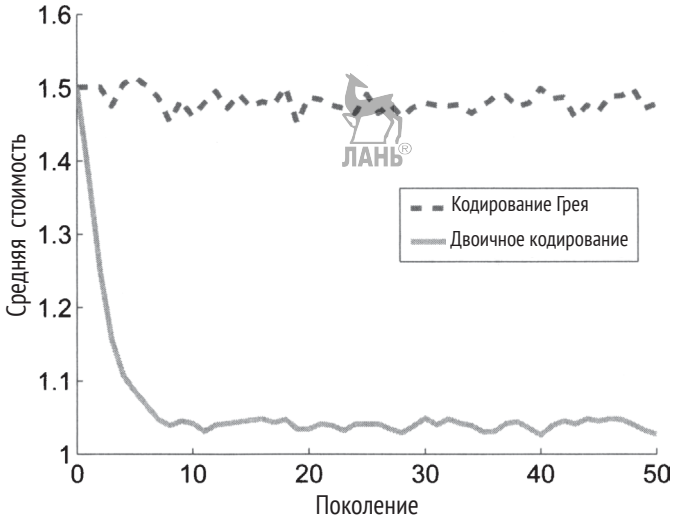


Рис. 8.5. Результаты примера 8.5. График показывает среднюю стоимость всех генетико-алгоритмических особей в каждом поколении 20-битной задачи, которая имеет много локальных минимумов, усредненно по 50 симуляциям Монте-Карло. При поиске нескольких локальных минимумов двоичное кодирование работает лучше, чем кодирование Грея

■

8.4. Элитарность

В этом разделе обсуждается тема элитарности. Элитарность является способом гарантировать, что самые приспособленные особи в эволюционном алгоритме оставляются в популяции из поколения в поколение. Хотя показанные в этой книге результаты выглядят довольно хорошо, существует опасность, что из поколения в поколение мы, возможно, будем терять некоторых наших лучших особей. Элитарность этому препятствует.

Рассмотрим схематичное описание генетического алгоритма на рис. 3.6. Мы видим, что лучшие родители рекомбинируют, производя детей. Однако если в i-м поколении для нашей оптимизационной задачи существует превосходное кандидатное решение x_e, то не будет никакой гарантии, что лучшая особь в $(i + 1)$-м поколении будет улучшением,

по сравнению с x_e, или даже будет такой же хорошей, как и x_e. Особь x_e будет рекомбинировать с другими родителями и производить детей, но x_e не будет частью следующего поколения. Как сохранить хорошие результаты рекомбинации, избегая при этом выпадения лучшей особи из популяции?

Ответ на этот вопрос заключается в том, чтобы оставлять лучших особей в эволюционном алгоритме из поколения в поколение. Эта идея, впервые предложенная в публикации [De Jong, 1975], называется элитарностью и обычно улучшает результативность эволюционного алгоритма. Мы можем реализовать элитарность, по крайней мере, несколькими способами.

1. Мы можем реализовать элитарность, производя только ($N - E$) детей в каждом поколении, где N – это размер популяции, а E – определяемое пользователем число элитарных особей. Предположим, что мы хотим оставлять из поколения в поколение лучших E особей из общей популяции N. В этом случае мы будем применять рекомбинацию и мутацию для получения ($N - E$) детей, а затем мы объединим лучших E особей с детьми, получив следующее поколение из N особей. Рисунок 8.6, являющийся модификацией рис. 3.6, показывает этот вариант для элитарного генетического алгоритма. Мы можем легко применить эту идею и в других эволюционных алгоритмах.

```
Родители ← {случайно сгенерированная популяция}
While not (критерий останова)
    Рассчитать приспособленность каждого родителя в популяции.
    Элитарные особи ← Лучшие E родителей
    Дети ← ∅
    While | Дети | < | Родители | – E
        Применить приспособленности для вероятностного отбора пары
            родителей с целью их спаривания.
        Спарить родителей для создания детей c1 и c2.
        Дети ← Дети ∪ {c1, c2}
    Loop
    Случайно мутировать нескольких детей.
    Родители ← Дети ∪ Элитарные особи
Next поколение
```

Рис. 8.6. Вариант элитарности 1: простой генетический алгоритм, модифицированный для элитарности. N – это размер популяции, E – число элитарных особей, которые оставляются из поколения в поколение, и каждое поколение производит ($N - E$) детей

2. Мы можем реализовать элитарность, производя N детей и заменяя худших детей лучшими особями предыдущего поколения. На рис. 8.7 этот вариант показан для элитарного генетического алгоритма, и мы можем легко применить эту идею и в других эволюционных алгоритмах. Обычно с этим вариантом можем рассчитывать на более высокую результативность, чем с вышеуказанным вариантом элитарности, однако он требует дополнительного шага сортировки.

```
Родители ← {случайно сгенерированная популяция}
While not (критерий останова)
    Рассчитать приспособленность каждого родителя в популяции.
    Элитарные особи ← Лучшие E родителей
    Дети ← ∅
    While | Дети | < | Родители |
        Применить приспособленности для вероятностного отбора пары
            родителей с целью их спаривания.
        Спарить родителей для создания детей c1 и c2.
        Дети ← Дети ∪ {c1, c2}
    Loop
    Случайно мутировать нескольких детей.
    Родители ← Дети ∪ Элитарные особи
    Родители ← Лучшие N родителей
Next поколение
```

Рис. 8.7. Варинат элитарности 2: простой генетический алгоритм, модифицированный для элитарности. N – это размер популяции, E – число элитарных особей, которые оставляются из поколения в поколение, и каждое поколение производит N детей

3. Есть и другие варианты элитарности. Например, мы можем производить N детей и применять некий тип алгоритма обратного рулеточного отбора, отбирать E из них, где худшие дети имеют наибольшую вероятность отбора. Затем можем заменять этих детей лучшими E особями предыдущего поколения. Мы можем также реализовать другие вариации на эту тему.

■ Пример 8.6

В данном примере мы тестируем влияние элитарности на результативность генетического алгоритма. Мы снова оптимизируем двумерную функцию Экли, описанную в примере 3.3, где каждая размерность

кодируется шестью битами. Мы используем популяцию размером 20 и скорость мутации 2 % на бит на поколение. На рис. 8.8 показана минимальная стоимость 20 генетико-алгоритмических особей в каждом поколении, усредненно по 20 симуляциям Монте-Карло. Элитарный генетический алгоритм оставляет двух лучших особей каждое поколение и использует вариант элитарности, показанный на рис. 8.6. Мы видим, что встраивание элитарности значительно улучшает результативность генетического алгоритма.

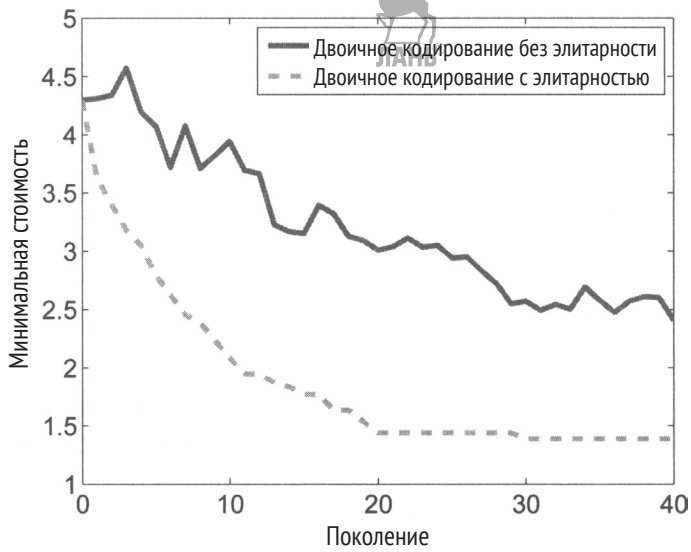


Рис. 8.8. Результаты примера 8.6 для минимизации двумерной функции Экли, где каждая размерность кодируется шестью битами. График показывает минимальную стоимость всех генетико-алгоритмических особей в каждом поколении, усредненно по 20 симуляциям Монте-Карло. Элитарный генетический алгоритм работает значительно лучше, чем неэлитарный генетический алгоритм

■

В наших эволюционных алгоритмах мы должны почти повсеместно применять элитарность, потому что она стоит очень мало, но приносит высокие дивиденды. Вместе с тем могут возникать некоторые типы задач с дорогими или динамическими функциями стоимости, для которых неэлитарный эволюционный алгоритм работает лучше, чем элитарный (см. главу 21).

8.5. Стационарные и поколенческие алгоритмы

Иногие эволюционные алгоритмы, которые мы обсуждали до сих пор, являются поколенческими эволюционными алгоритмами. Это означает, что каждое поколение целиком заменяется, за исключением, возможно, элитарных особей, как описано в разделе 8.4. Однако в природе эволюция происходит не так. В природе поколения перемежаются, и смерть и рождение происходят непрерывно. Этот тип эволюции называется стационарным. Наше наблюдение за природой мотивирует нас внедрять стационарные версии наших эволюционных алгоритмов. На рис. 8.9 дается схематичное описание стационарного генетического алгоритма.

Родители ← {случайно сгенерированная популяция}
Рассчитать приспособленность каждого родителя в популяции.
While not (критерий останова)
    Применить приспособленности для отбора пары родителей p_1 и p_2
        с целью их рекомбинации.
    Рекомбинировать родителей для создания детей c_1 и c_2.
    Случайно мутировать c_1 и c_2.
    Рассчитать приспособленность c_1 и c_2.
    $p_1 \leftarrow c_1$ и $p_2 \leftarrow c_2$
Next поколение

Рис. 8.9. Стационарный генетический алгоритм

Рисунок 8.9 показывает, что в каждом поколении мы создаем только двух детей, и двое детей замещают родителей в популяции. Сравните это с поколенческим генетическим алгоритмом на рис. 3.6, в котором мы создаем всех N детей, после чего они заменяют своих родителей. В стационарном эволюционном алгоритме можно реализовать различные варианты.

- Мы можем заменять p_1 на c_1 только в том случае, если c_1 имеет лучшую приспособленность, чем p_1, и мы можем делать то же самое с p_2 и c_2. Данный вариант похож на элитарность, как описано в разделе 8.4, потому что лучшая особь никогда не будет утеряна из популяции. Этот вариант также похож на эволюционную стратегию $(\mu + \lambda)$ рис. 6.10, в которой ребенок выживает до следующего поколения, только если он является одним из μ лучших из $(\mu + \lambda)$ особей.

- Мы можем создавать и заменять более двух особей в каждом поколении. Обратите внимание, что поколенческий генетический алгоритм на рис. 3.6 каждое поколение заменяет все *N* особей, в то время как стационарный генетический алгоритм на рис. 8.9 каждое поколение заменяет только двух особей. Мы можем спроектировать генетический алгоритм, который находится где-то между этими двумя крайностями, заменяя в каждом поколении четырех особей, либо шесть особей, либо случайное число особей, либо любое число по нашему выбору. Кеннет Де Йонг для обозначения числа особей, которых мы заменяем каждое поколение, использует термин «разрыв поколений» [De Jong, 1975].

Обратите внимание, что мы не можем сравнивать результативность поколенческого генетического алгоритма рис. 3.6 с результативностью стационарного генетического алгоритма рис. 8.9 путем выполнения обоих алгоритмов на одинаковом числе поколений. Поколение рис. 3.6 производит *N* детей, в то время как поколение рис. 8.9 производит только двух детей. Поэтому поколенческий генетический алгоритм всегда будет превосходить стационарный для заданного числа поколений. Такое сравнение было бы несправедливым. Мы можем сделать справедливое сравнение, только выполнив оба алгоритма для одинакового числа оцениваний функции приспособленности. Таким образом, *NG*/2 поколений стационарного генетического алгоритма на рис. 8.9 будут вычислительно эквивалентны *G* поколениям поколенческого генетического алгоритма на рис. 3.6.

Рисунки 3.6 и 8.9 иллюстрируют поколенческие и стационарные стратегии для генетических алгоритмов, но мы можем легко распространить эти идеи практически на любой другой эволюционный алгоритм. Как мы видели на предыдущих страницах этой книги и как мы также увидим позже, некоторые эволюционные алгоритмы, по всей видимости, более естественно укладываются в поколенческий подход, а другие более естественно укладываются в стационарный подход. Однако стандартная реализация любого эволюционного алгоритма может быть изменена по желанию пользователя для получения либо поколенческого, либо стационарного алгоритма.

8.6. Популяционное многообразие

В данном разделе обсуждается вопрос, как поступать с дублирующими особями в популяции и как отбор и рекомбинация могут модифици-

роваться для поощрения многообразия в мультимодальных задачах. В разделе 8.6.1 мы сначала рассмотрим проблему дублирующих особей. Затем мы обсудим два метода обеспечения многообразия в эволюционно-алгоритмических популяциях: ограничения на рекомбинацию в разделе 8.6.2 и методы поддержания популяционных ниш в разделе 8.6.3, которые включают в себя обмен информацией о приспособленности, очистку и скучивание.

8.6.1. *Дублирующие особи*

В популяции, которая из поколения в поколение многократно рекомбинируется, часто возникает единообразие. Это означает, что вся популяция целиком становится популяцией клонов. Единообразие встречается чаще в задачах с дискретной областью, чем в задачах с непрерывной областью, но она может возникать в обоих типах задач. Единообразие ограничивает возможности эволюционного алгоритма по дальнейшему разведыванию поискового пространства. Хотя кандидатное решение, к которому сходится эволюционный алгоритм, обычно является хорошим решением, в других участках поискового пространства могут существовать гораздо более качественные решения; поэтому даже после того, как эволюционный алгоритм найдет хорошее решение, мы надеемся, что он будет продолжать разведывать пространство в попытке найти еще более качественные решения. Когда единообразие возникает до того, как мы нашли удовлетворительное решение нашей оптимизационной задачи, оно называется преждевременным схождением [Ronald, 1998]. Мы можем предотвратить его с помощью более высокой скорости мутации, но если мы используем слишком высокую скорость мутации, то наш эволюционный алгоритм выродится в случайный поиск. Одним из распространенных способов предотвращения преждевременного схождения является постоянный поиск в популяции наличия дублирующих особей и их замена. Мы можем делать это несколькими разными способами, как описано ниже.

1. Всякий раз, когда мы создаем ребенка, мы можем сканировать популяцию, чтобы убедиться, что не создаем дубликат. Если мы создали дубликат, то можем повторить операцию рекомбинации с разными родителями или разными параметрами скрещивания и в результате получить другого, недублирующего ребенка. Либо для получения недубликата мы можем мутировать ребенка.

2. Всякий раз, когда мы мутируем особь, мы можем сканировать популяцию, чтобы убедиться, что мы не создаем дубликат. Если мы создали дубликат, то можем повторить операцию мутации.
3. В конце каждого поколения мы можем сканировать популяцию на наличие дубликатов. Мы можем заменять дубликаты разными способами. Например, можем заменять дубликаты случайно сгенерированными особями, либо мутировать дубликаты, либо выполнять операцию рекомбинации для замены каждого дубликата.
4. Мы можем допускать наличие в популяции нескольких дубликатов, но не более чем задаваемый пользователем порог D. Дубликаты, скорее всего, будут высоко приспособленными особями, иначе они вряд ли будут встречаться в популяции. Следовательно, мы, возможно, не будем возражать против дубликатов, поскольку они обеспечивают более высокую вероятность участия высоко приспособленных особей в рекомбинации. Поэтому мы можем заменять дубликаты, только если их больше D. Либо мы можем вероятностно заменять их, в зависимости от того, насколько они приспособлены или сколько есть дубликатов.

Сканирование на дубликаты, вероятно, выглядит дорогостоящим с точки зрения вычислений, поскольку требует вложенного цикла и, следовательно, вычислительных усилий порядка N^2, где N – это размер популяции. В эталонных задачах на него может уходить значительная часть вычислительных усилий эволюционного алгоритма. Однако мы должны помнить, что реальные задачи, как правило, на порядок сложнее, чем эталонные. В реальных задачах оценивание функции приспособленности составляет подавляющую часть вычислительных усилий (см. главу 21), а вычислительные усилия на операцию поиска и замены дубликатов будут незначительными. Вместе с тем если вычислительные усилия на эталонное тестирование эволюционного алгоритма вызывают озабоченность, то мы можем сократить усилия, сканируя популяцию на наличие дубликатов каждые G поколений, вместо того чтобы сканировать каждое поколение, где G – это определяемый пользователем параметр.

8.6.2. *Нишевая и видовая рекомбинация*

В типичной реализации эволюционного алгоритма отбираются родительские особи, которые затем объединяются для получения детей, никак не учитывая то, насколько похожи или различны родители. Однако в биологии мы часто видим, что родители друг на друга похожи,

но похожи не слишком. Например, мы редко видим спаривание между особями разных видов, но так же редко видим спаривание между близкими родственниками. На рис. 8.10 дана иллюстрация проблемы с рекомбинацией между двумя сильно отличающимися друг от друга особями. Во-первых, получившийся ребенок может быть слабым решением оптимизационной задачи, потому что срединная точка между двумя хорошо приспособленными особями может иметь слабую приспособленность. Во-вторых, скрещивание может привести к потере генетической информации, которая может быть важна для решения задачи.

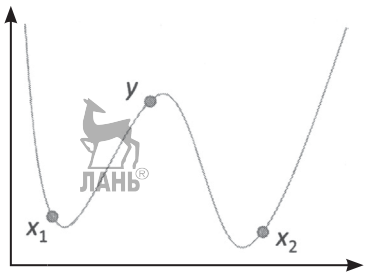


Рис. 8.10. Данный рисунок демонстрирует задачу мультимодальной минимизации. Этот пример функции иллюстрирует проблему с рекомбинацией двух сильно отличающихся друг от друга особей. Родители x_1 и x_2 имеют низкую стоимость, но их ребенок y имеет высокую стоимость. Кроме того, если ребенок y заменяет одного из родителей (например, x_2), то эволюционному алгоритму будет трудно найти глобальный оптимум, близкий к x_2

Рисунок 8.11 иллюстрирует проблему рекомбинации между двумя слишком похожими друг на друга особями. Если мы не допустим рекомбинации между особями, которые друг от друга отличаются, то эволюционный поисковый процесс может застрять в колее.

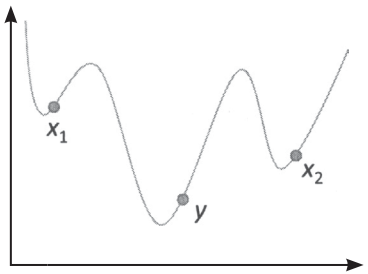


Рис. 8.11. Данный рисунок демонстрирует задачу мультимодальной минимизации. Этот пример функции иллюстрирует проблему, возникающую, когда сильно отличающимся друг от друга особям не разрешается рекомбинировать. Если x_1 и x_2 не разрешается рекомбинировать из-за их несходства, то может оказаться трудно найти глобальный минимум, близкий к y

Рассмотренные выше проблемы мотивируют создание нишевых и видовых стратегий рекомбинации.

- Стратегии ниширования препятствуют рекомбинации между сильно отличающимися друг от друга особями, в пространстве области [Mahfoud, 1995b], [Mahfoud, 1995a]. Они не только помогают отыскивать оптимальные решения, как показано на рис. 8.10, но и помогают отыскивать несколько локальных оптимумов, что важно для многих задач. Ниширование также может быть полезно для многокритериальной оптимизации (см. главу 20) и для динамической оптимизации (см. раздел 21.2).
- Видовые стратегии препятствуют рекомбинации между особями, которые очень похожи друг на друга в пространстве области [Banzhaf et al, 1998, раздел 6.4]. Они поощряют разведывание, поощряя рекомбинацию между особями, которые сильно отличаются друг от друга.

Обратите внимание, что для одной и той же задачи нишевые и видовые стратегии являются противоположными стратегиями. Философия нишевой рекомбинации заключается в том, что когда мы рекомбинируем приспособленных особей, то не можем рассчитывать, что ребенок будет приспособленным, если родители не похожи друг на друга. Философия видовой рекомбинации заключается в том, что когда мы рекомбинируем приспособленных особей, то должны убедиться, что родители друг от друга отличаются, благодаря чему мы можем эффективно разведывать поисковое пространство. Выбор того, какой из этих подходов использовать, является решением, которое зависит от конкретной задачи.

8.6.3. *Ниширование*

В этом разделе мы используем термин *ниширование* (niching) иначе, чем в предыдущем разделе. Ниширование в этом разделе – это метод, который позволяет эволюционно-алгоритмическим особям выживать в отдельных карманах поискового пространства. Ниширование здесь имеет такую же обусловленность, как и в предыдущем разделе; однако в нишировании предыдущего раздела специально рассматривается отбор родителей, а ниширование в этом разделе связано с регулировкой значений приспособленности.

Ниширование обусловливается мультимодальными задачами, для которых может оказаться важным поддерживать особей вблизи мно-

гих локальных оптимумов или для которых может оказаться важным находить несколько хороших решений. Самый ранний метод нишировання – это обмен информацией о приспособленности [Holland, 1975, С. 164]. Мы обсудим три стратегии нишировання: обмен информацией о приспособленности в разделе 8.6.3.1, очистку в разделе 8.6.3.2 и скучивание в разделе 8.6.3.3. Дополнительное обсуждение этих идей представлено в публикации [Sareni и Krahenbiihl, 1998].

8.6.3.1. Обмен информацией о приспособленности (fitness sharing)

Иногда эволюционно-алгоритмические особи в хорошем участке поискового пространства могут начать доминировать над популяцией. Это может привести к преждевременному схождению. Мы хотели бы оставлять хороших особей в популяции, но мы также хотим сохранять многообразие, благодаря которому у нас будет возможность разведывать новые участки поискового пространства по мере поступательного развития эволюционной программы из поколения в поколение. Обмен информацией о приспособленности особенно полезен для мультимодальных задач, в которых мы хотим найти несколько решений в разных участках поискового пространства.

Рассмотрим рис. 8.12. Мы видим, что x^* является глобальным максимумом, но особи возле x^* вряд ли выживут до следующего поколения из-за их низких значений приспособленности, по сравнению с другими особями в поисковом пространстве. Для того чтобы поощрять многообразие в популяции, мы можем искусственно увеличивать значения приспособленности относительно уникальных особей и уменьшать значения приспособленности относительно распространенных особей.

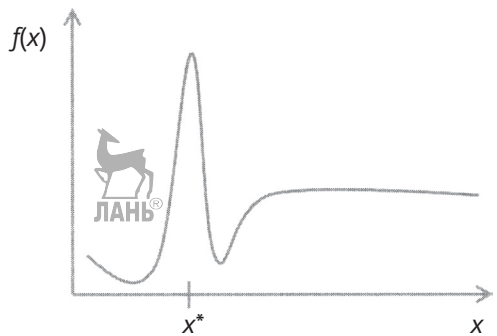


Рис. 8.12. Эта функция имеет глобальный максимум в x^*, но особи возле x^* вряд ли будут выбраны для рекомбинации из-за их низких значений приспособленности по отношению к другим особям в поисковом пространстве

Обмен информацией о приспособленности уменьшает приспособленности особей, которые находятся рядом друг с другом в поисковом пространстве [Sareni и Krahenbiihl, 1998]. Биологическое обоснование этой идеи заключается в том, что подобные друг другу особи конкурируют за подобные ресурсы. Поэтому даже если особь – высоко приспособленная, она не сможет размножаться, если в том же географическом регионе есть много других подобных ей особей.

Предположим, что у нас есть эволюционно-алгоритмическая популяция $\{x_i\}$ из N особей и что f_i является приспособленностью x_i. Метод обмена информацией о приспособленности вычисляет модифицированные значения приспособленности следующим образом:

$$f_i' = f_i / m_i, \tag{8.7}$$

где число m_i, так называемое число ниш в x_i, связано с числом особей, подобных x_i. Число ниш вычисляется формулой

$$m_i = \sum_{j=1}^{N} s(d_{ij}), \tag{8.8}$$

где $s(\cdot)$ – это функция обмена, а d_{ij} – расстояние между особями x_i и x_j. Для получения d_{ij} мы часто используем евклидово расстояние. Общепринято использовать функцию обмена

$$s(d) = \begin{cases} 1 - (d/\sigma)^{\alpha}, & \text{если } d < \sigma \\ 0 & \text{в противном случае} \end{cases}, \tag{8.9}$$

где σ – это определяемый пользователем параметр, называемый порогом несходства, порогом разнородности, расстоянием отсечения или нишевым радиусом, и α – определяемый пользователем параметр. Мы обычно используем $\alpha = 1$, в результате получая треугольную функцию обмена. Исследователи предложили разные методы установки порога несходства [Deb и Goldberg, 1989]. Например,

$$\begin{aligned} \sigma &= rq^{-1/n}, \\ r &= \frac{1}{2}\sum_{k=1}^{n}\left(\max_i x_i(k) - \min_i x_i(k)\right)^2, \end{aligned} \tag{8.10}$$

где n – это размерность задачи, $x_i(k)$ – k-й элемент x_i и q – ожидаемое число локальных оптимумов в функции приспособленности. В обмене информацией о приспособленности при отборе родителей для рекомбинации мы используем модифицированные значения приспособленности из уравнения (8.7). Обратите внимание, что если у нас минимизи-

зационная задача, то перед применением уравнения (8.7) нам нужно конвертировать значения стоимости в значения приспособленности, а затем конвертировать модифицированные значения приспособленности в модифицированные значения стоимости (см. задачу 8.7).

8.6.3.2. Очистка (clearing)

Очистка аналогична обмену информацией о приспособленности, но, вместо того чтобы делиться значениями приспособленности между особями, которые разделяют одну и ту же нишу, мы уменьшаем приспособленность некоторых из этих особей [Pétrowski, 1996], [Sareni и Krahenbiihl, 1998]. Данную идею можно реализовать несколькими способами, в том числе следующими. Во-первых, мы определяем множество ниш D_i каждой особи в популяции:

$$D_i = \{x_j : d_{ij} < \sigma\}, \tag{8.11}$$

где d_{ij} – это такое же расстояние, как и в уравнении (8.8), а σ – определяемый пользователем параметр. Затем мы ранжируем особей в каждой нише в соответствии с их приспособленностью:

$$r_{ki} = \text{rank of } x_k \text{ in } D_i, \tag{8.12}$$

где лучшая особь в каждой нише имеет ранг 1, вторая лучшая особь имеет ранг 2 и т. д. Наконец, мы определяем параметр R как число особей, которые по нашему желанию должны выжить в каждой нише, и получаем модифицированные значения приспособленности следующим образом, где N – это размер популяции:

```
For i = 1 to N
    For k = 1 to | D_i |
        If r_ki ≤ R then
            f'_k ← f_k
        else
            f'_k ← −∞
        End if
    Next ниша
Next особь
```

Приведенный выше алгоритм гарантирует, что наименее приспособленные особи в каждой нише недоступны для отбора или рекомбинации. Тем не менее это не гарантирует, что наиболее приспособленные особи будут доступны для отбора, потому что особи могут принадлежать более чем к одной нише. Например, особь x_m, возможно, будет наиболее приспособленной в своей нише, но она, возможно, также будет входить

в другую нишу, в которой она является не самой приспособленной, и в этом случае ее модифицированная приспособленность могла бы быть принята равной $-\infty$. Множество особей, которые выживают по вышеуказанному алгоритму, зависит от порядка, в котором мы обрабатываем ниши $\{D_i\}$.

8.6.3.3. Скучивание (crowding)

Скучивание работает путем замены отдельных особей в популяции на подобных особей, которые недавно были произведены путем рекомбинации. Метод скучивания был введен Кеннетом Де Йонгом [De Jong, 1975] для имитации конкуренции за ресурсы в природе. Ниже мы рассмотрим три типа скучивания: стандартное скучивание, детерминированное скучивание и ограниченный турнирный отбор.

Стандартное скучивание

Стандартное скучивание используется в сочетании со стационарной рекомбинацией (см. раздел 8.5). Стандартное скучивание производит M детей в каждом поколении, а затем сравнивает этих детей со случайно отобранными родителями C_f, где M и C_f – это задаваемые пользователем параметры. C_f называется фактором скученности. Каждый ребенок заменяет наиболее похожую особь из группы случайно отобранных родительских особей. Обычно используются значения параметров $M = N/10$ и $C_f = 3$, где N – это размер популяции [Mahfoud, 1992]. На рис. 8.13 показана реализация стандартого скучивания.

Родители $\{p_k\} \leftarrow$ {случайно сгенерированная популяция из N особей}
Рассчитать приспособленность каждого родителя p_k, $k \in [1, N]$.
While not (критерий останова)
  Применить приспособленности для вероятностного отбора M родителей
    с целью рекомбинации.
  Рекомбинировать родителей для создания M детей c_i, $i \in [1, M]$.
  Случайно мутировать каждого ребенка c_i, $i \in [1, M]$.
  Рассчитать приспособленность каждого ребенка c_i, $i \in [1, M]$.
  For $i = 1$ **to** M
    Случайно отобрать C_f особей I из родительской популяции $\{p_k\}$.
    $p_{\min} = \arg\min_p \| p - c_i \| : p \in I$
    $p_{\min} \leftarrow c_i$
  Next ребенок
Next поколение

Рис. 8.13. Стационарный эволюционный алгоритм со стандартным скучиванием. M и C_f – это отобранные пользователем параметры, а $\|p - c_i\|$ – определяемая пользователем функция расстояния

Детерминированное скучивание

Детерминированное скучивание предусматривает турниры между детьми и родителями [Mahfoud, 1995b]. Родители объединяются для производства детей, и каждый ребенок заменяет собой самого похожего на него родителя, но только если ребенок имеет более высокую приспособленность, чем этот родитель. На рис. 8.14 показана реализация детерминированного скучивания.

Родители ← {случайно сгенерированная популяция}
Рассчитать приспособленность каждого родителя в популяции.
While not (критерий останова)
  Применить приспособленности для вероятностного отбора пары родителей p_1 и p_2.
  Рекомбинировать родителей для создания детей c_1 и c_2.
  Случайно мутировать c_1 и c_2.
  Рассчитать приспособленность c_1 и c_2.
  For $i = 1$ **to** 2
    If $||p_1 - c_i|| < ||p_2 - c_i||$ и приспособленность(c_i) > приспособленность (p_1) **then**
      $p_1 \leftarrow c_i$
    else if $||p_2 - c_i|| < ||p_1 - c_i||$ и приспособленность (c_i) > приспособленность (p_2) **then**
      $p_2 \leftarrow c_i$
    End if
  Next ребенок
Next поколение

Рис. 8.14. Стационарный эволюционный алгоритм с детерминированным скучиванием. Каждый ребенок заменяет своего ближайшего родителя, если ребенок больше приспособлен, чем родитель

Ограниченный турнирный отбор

Ограниченный турнирный отбор имеет общие черты как со стандартным, так и с детерминированным скучиванием [Harik, 1995]. *M* родителей рекомбинируют для производства двух детей. Затем мы сравниваем детей со случайно отобранными особями. Каждый ребенок заменяет самую похожую особь из группы случайно отобранных особей, но только в том случае, если ребенок имеет более высокую приспособленность. См. рис. 8.15, где приведена реализация ограниченного турнирного отбора.

Родители ← {случайно сгенерированная популяция}
Рассчитать приспособленность каждого родителя в популяции.
While not (критерий останова)

```
Применить приспособленности для вероятностного отбора M родителей
    с целью рекомбинации.
Рекомбинировать родителей для создания M детей c_i, i ∈ [1, M].
Случайно мутировать каждого ребенка c_i, i ∈ [1, M].
Рассчитать приспособленность каждого ребенка c_i, i ∈ [1, M].
Случайно отобрать C_f особей I из родительской популяции.
For i = 1 to M
    p_min = arg min_p || p – c_i || : p ∈ I
    If приспособленность(c_i) > приспособленность(p_min) then
        p_min ← c_i
    End if
Next ребенок
Next поколение
```

Рис. 8.15. Стационарный эволюционный алгоритм с ограниченным турнирным отбором. M и C_f – это отбираемые пользователем параметры, а $||p - c||$ – определяемая пользователем функция расстояния

8.7. Варианты отбора

Прежде чем мы сможем скомбинировать эволюционно-алгоритмических особей для создания детей, мы должны отобрать особей для использования в качестве родителей. Рулеточный отбор, о котором мы говорили в разделе 3.4.2, является стандартным для генетического алгоритма методом отбора; он применяется и в других эволюционных алгоритмах. Вместе с тем существует большое число других алгоритмов отбора, и в этом разделе описаны семь таких алгоритмов. Почти все методы отбора смещены в сторону приспособленных особей в популяции. То есть независимо от того, какой метод отбора мы используем, более приспособленная особь почти всегда будет иметь больше шансов быть отобранной, чем менее приспособленная.

Если метод отбора чрезмерно смещен в сторону отбора приспособленных особей, то популяция может слишком быстро сходиться к единому решению, не разведав поисковое пространство достаточно широко. Однако если метод отбора смещен в сторону приспособленных особей не достаточно сильно, то эволюционный алгоритм может быть не в состоянии правильно эксплуатировать информацию, которая присутствует у наиболее приспособленных особей.

Полезным метрическим показателем для квантификации разницы между различными алгоритмами отбора является селекционное давление 0, которое определяется как

$$\phi = \frac{\Pr(\text{отбор наиболее приспособленной особи})}{\Pr(\text{отбор средне приспособленной особи})}, \qquad (8.13)$$

где Pr(отбор x) – это вероятность того, что для рекомбинации будет отобрана особь x. Селекционный отбор квантифицирует величину относительной вероятности того, что высоко приспособленная особь будет принимать участие в рекомбинации. Ниже мы рассмотрим семь разных типов отбора.

8.7.1. *Стохастическая универсальная выборка*

Как отмечалось выше, стандартным методом отбора в генетических алгоритмах и во многих других эволюционных алгоритмах является рулеточный отбор, то есть отбор пропорционально приспособленности. На рис. 3.2, который для удобства воспроизводится ниже на рис. 8.16, показан рулеточный отбор для четырехчленной популяции.

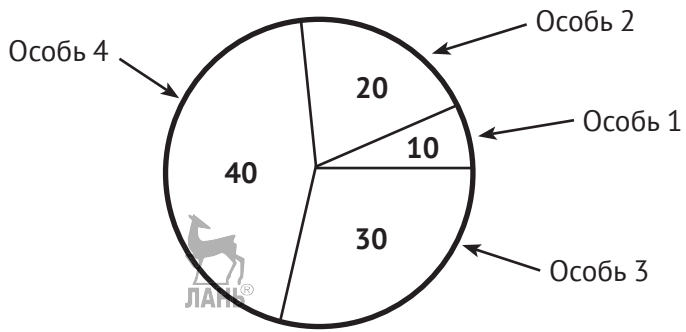


Рис. 8.16. Иллюстрация рулеточного отбора для четырехчленной популяции. Каждой особи присваивается сектор, площадь которого пропорциональна ее приспособленности. Отбор каждой особи в качестве родителя пропорционален площади ее сектора в колесе рулетки

Потенциальная проблема с рулеточным отбором заключается в том, что существует хороший шанс, что лучшая особь не будет отобрана для рекомбинации. Например, предположим, что мы вращаем колесо рулетки на рис. 8.16 четыре раза, для того чтобы отобрать четырех родителей для рекомбинации. Вероятность того, что лучшая особь, особь #4, не будет отобрана ни в одном из четырех вращений, равна $(0.6)^4 = 13\ \%$. Мы имеем шанс, равный примерно 1/7, что лучшая особь не будет отобрана для рекомбинации. Это может оказаться неприемлемо высокой вероятностью для потери информации о лучшей особи в популяции.

Стохастическая универсальная выборка [Baker, 1987] решает эту проблему, при этом по-прежнему используя рулеточный подход. Вместо того чтобы вращать колесо рулетки на рис. 8.16 четыре раза и отбирать четырех родителей, мы используем вертушку (спиннер) с четырьмя равноотстоящими указателями, помещаем ее на колесо рулетки и вращаем ее один раз. Это дает нам четырех родителей за одно вращение и гарантирует, что мы по крайней мере один раз выберем особь #3 и по крайней мере один раз выберем особь #4, так как они обе имеют доли приспособленности, которые превышают 25 % от общих суммированных значений приспособленности. Рисунок 8.17 демонстрирует эту идею.

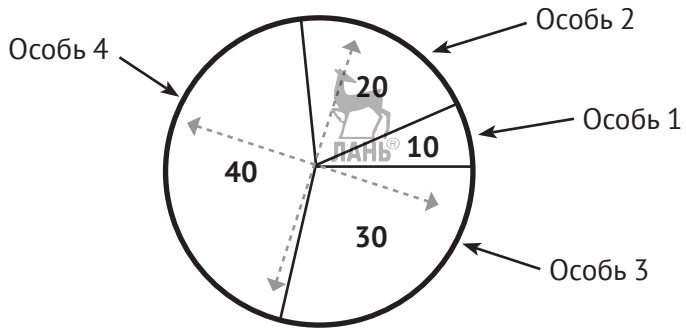


Рис. 8.17. Стохастическая универсальная выборка для четырехчленной популяции. Каждой особи назначается сектор, пропорциональный ее приспособленности. Вертушка с четырьмя равноотстоящими указателями вращается один раз, для того чтобы получить четырех родителей

Стохастическая универсальная выборка, примененная к значениям приспособленности на рис. 8.17, даст нам одну из следующих родительских выборок:

	Особь #1, #2, #3 и #4,
или	Особь #1, #3, #4 и #4,
или	Особь #2, #3, #3 и #4,
или	Особь #2, #3, #4 и #4.

(8.14)

На рис. 8.18 показан псевдокод для стохастической универсальной выборки. Сравните его с программным кодом рулеточного отобора на рис. 3.5. Рисунок 8.18 гарантирует, что особь x_i будет отобрана где-то между $N_{i,\min}$ и $N_{i,\max}$ раз, где

$$N_{i,\min} = \left\lfloor \frac{Nf_i}{f_{\text{sum}}} \right\rfloor,$$
$$N_{i,\max} = \left\lceil \frac{Nf_i}{f_{\text{sum}}} \right\rceil, \tag{8.15}$$

где $\lceil \alpha \rceil$ – это наибольшее целое число, которое меньше или равно α, и $\lfloor \alpha \rfloor$ – наименьшее целое число, которое больше или равно α.

$x_i = i$-й особь в популяции, $i \in [1, N]$
$f_i \leftarrow$ приспособленность x_i, для $i \in [1, N]$
$f_{\text{sum}} \leftarrow \sum_{i=1}^{N} f_i$
Сгенерировать равномерно распределеное случайное число $r \in [0, f_{\text{sum}}/N]$.
$f_{\text{accum}} \leftarrow 0$
Родители $\leftarrow \emptyset$
$k \leftarrow 0$
While | Родители | $< N$
    $k \leftarrow k + 1$
    $f_{\text{accum}} \leftarrow f_{\text{accum}} + f_k$
    While $f_{\text{accum}} > r$
        Родители $\leftarrow$ Родители $\cup\, x_k$
        $r \leftarrow r + f_{\text{sum}}/N$
    End while
Next родитель

Рис. 8.18. Псевдокод для отбора N родителей из N особей с помощью стохастической универсальной выборки. Этот код исходит из того, что $f_i \geq 0$ для всех $i \in [1, N]$

8.7.2. *Избыточный отбор*

Избыточный отбор (over-selection) – это метод, первоначально предложенный Дж. Козой в контексте генетического программирования [Koza, 1992, глава 6]. Избыточный выбор модифицирует рулеточный отбор путем непропорционального взвешивания значений приспособленности высоко приспособленных особей, для того чтобы увеличить их шансы на отбор. В версии Дж. Козы избыточного отбора лучшие 32 % популяции имеют 80%-ный шанс быть отобранными, и худшие 68 % популяции имеют 20%-ный шанс быть отобранными. Точные проценты не слишком важны; ключевой особенностью избыточного отбора является то, что приспособленные особи имеют значительно более высокую

вероятность быть отобранными. Это один из типов шкалирования по значению приспособленности [Goldberg, 1989a].

Дж. Коза протестировал три разных типа отбора для генетического программирования и обнаружил, что рулеточный отбор выполняется хуже, турнирный отбор выполняется лучше, а избыточный отбор выполняется лучше всех [Koza, 1992, глава 25]. Однако этот результат может быть связан с большими размерами популяции, которые обычно используются в генетическом программировании. Для более мелких популяций избыточный отбор мог бы оказывать слишком большое селекционное давление на ранних этапах эволюции, когда популяционная дисперсия приспособленности велика (см. уравнение (8.13)), но дополнительное селекционное давление могло бы быть полезным на более поздних этапах эволюции, когда дисперсия приспособленности мала.

8.7.3. *Сигма-шкалирование*

Сигма-шкалирование (sigma scaling) нормализует значения приспособленности относительно стандартного отклонения приспособленности всей популяции в целом. Шкалированные значения приспособленности следующие:

$$f'(x_i) = \begin{cases} \max[1 + (f(x_i) - \overline{f})/(2\sigma), \epsilon] & \text{если } \sigma \neq 0 \\ 1 & \text{если } \sigma = 0 \end{cases}, \tag{8.16}$$

где $f(x_i)$ – это приспособленность i-й особи в популяции, $\overline{f}$ – среднее значений приспособленности, σ – стандартное отклонение значений приспособленности и ϵ – неотрицательное определяемое пользователем минимально допустимое шкалированное значение приспособленности.

Обратите внимание на неоднозначность утверждения о том, что σ является стандартным отклонением значений приспособленности. Если значения приспособленности свободны от шума и мы хотим знать стандартное отклонение конкретных измеренных нами значений приспособленности, то стандартное отклонение определяется так:

$$\sigma = \left(\frac{1}{N} \sum_{i=1}^{N} \left(f(x_i) - \overline{f} \right)^2 \right)^{1/2}. \tag{8.17}$$

Однако если значения приспособленности включают шум или если мы рассматриваем значения приспособленности как выборки из рас-

пределения вероятностей, то несмещенная оценка стандартного отклонения значений приспособленности вычисляется следующим образом [Simon, 2006, задача 3.6]:

$$\sigma = \left(\frac{1}{N-1} \sum_{i=1}^{N} \left(f(x_i) - \overline{f} \right)^2 \right)^{1/2}. \tag{8.18}$$

■ Пример 8.7

Предположим, что у нас есть четырехчленная популяция со значениями приспособленности

$$f(x_1) = 10, \quad f(x_2) = 5, \quad f(x_3) = 40, \quad f(x_4) = 15. \tag{8.19}$$

Рулеточный отбор дает особям следующие селекционные вероятности:

$$\Pr(x_1) = 14\ \%, \quad \Pr(x_2) = 7\ \%, \quad \Pr(x_3) = 57\ \%, \quad \Pr(x_4) = 22\ \%. \tag{8.20}$$

Среднее и стандартное отклонение значений приспособленности уравнения (8.19) равны $\overline{f} = 17.5$ и $\sigma = 15.5$, где для вычисления σ мы используем уравнение (8.18). Уравнение (8.16) дает шкалированные значения приспособленности:

$$f'(x_1) = 0.76, \quad f'(x_2) = 0.60, \quad f'(x_3) = 1.72, \quad f'(x_4) = 0.92. \tag{8.21}$$

Если мы применим эти шкалированные значения приспособленности в алгоритме рулеточного отбора, то получим селекционные вероятности:

$$\Pr(x_1) = 19\ \%, \quad \Pr(x_2) = 15\ \%, \quad \Pr(x_3) = 43\ \%, \quad \Pr(x_4) = 22\ \%. \tag{8.22}$$

Сравнивая уравнения (8.20) и (8.22), мы видим, что сигма-шкалирование имеет тенденцию выравнивать селекционные вероятности широко отличающихся значений приспособленности.

■

■ Пример 8.8

В качестве еще одного примера предположим, что у нас есть четыре особи со следующими значениями приспособленности:

$$f(x_1) = 15, \qquad f(x_2) = 25,$$
$$f(x_3) = 20, \qquad f(x_4) = 10. \tag{8.23}$$

Рулеточный отбор дает этим особям следующие селекционные вероятности:

$$\Pr(x_1) = 21\ \%, \qquad \Pr(x_2) = 36\ \%,$$
$$\Pr(x_3) = 29\ \%, \qquad \Pr(x_4) = 14\ \%. \tag{8.24}$$

Среднее и стандартное отклонение значений приспособленности уравнения (8.23) равны $\overline{f} = 17.5$ и $\sigma = 6.5$. Уравнение (8.16) дает шкалированные значения приспособленности:

$$f'(x_1) = 0.81, \qquad f'(x_2) = 1.58,$$
$$f'(x_3) = 1.19, \qquad f'(x_4) = 0.42. \tag{8.25}$$

Если мы применим эти шкалированные значения приспособленности в алгоритме рулеточного отбора, то получим селекционные вероятности:

$$\Pr(x_1) = 20\ \%, \qquad \Pr(x_2) = 40\ \%,$$
$$\Pr(x_3) = 30\ \%, \qquad \Pr(x_4) = 10\ \%. \tag{8.26}$$

Сравнивая уравнения (8.24) и (8.26), мы видим, что сигма-шкалирование имеет тенденцию широко разбрасывать селекционные вероятности близко расположенных значений приспособленности.

■

8.7.4. *Ранговый отбор*

Ранговый отбор (rank-based selection), так называемое ранговое взвешивание, сортирует особей в популяции от лучшего к худшему и выполняет отбор, используя ранговое положение, а не абсолютные значения приспособленности [Whitley, 1989]. Например, предположим, что у нас в популяции есть четыре особи со значениями приспособленности уравнения (8.19) и рулеточными селекционными вероятностями уравнения (8.20). Ранговый отбор ранжирует особей в соответствии со значениями приспособленности, давая лучшей особи ранг N (где N – это размер популяции) и худшей особи – ранг 1:

$$R(x_1) = 2, \qquad R(x_2) = 1,$$
$$R(x_3) = 4, \qquad R(x_4) = 3. \tag{8.27}$$

Ранговый отбор затем выполняет отбор на основе рангов, а не на основе значений приспособленности.

■ Пример 8.9

Предположим, что мы используем ранговый отбор в сочетании с рулеточным отбором для рангов уравнения (8.27). В результате получаем следующие селекционные вероятности:

$$\begin{aligned} \Pr(x_1) &= 20\ \%, & \Pr(x_2) &= 10\ \%, \\ \Pr(x_3) &= 40\ \%, & \Pr(x_4) &= 30\ \%. \end{aligned} \tag{8.28}$$

Мы видим, что когда значения приспособленности друг от друга сильно отличаются, как в уравнении (8.19), ранговый отбор выравнивает селекционные вероятности. Это не дает выско приспособленным особям доминировать в популяции на ранних стадиях эволюции, то есть предотвращает преждевременное схождение.

■

■ Пример 8.10

В качестве еще одного примера предположим, что у нас есть четыре особи со значениями приспособленности уравнения (8.23) и рулеточными селекционными вероятностями уравнения (8.24). Ранговый отбор ранжирует особей следующим образом:

$$\begin{aligned} R(x_1) &= 2, & R(x_2) &= 4, \\ R(x_3) &= 3, & R(x_4) &= 1. \end{aligned} \tag{8.29}$$

Если мы применим ранговый отбор в сочетании с рулеточным отбором, то уравнение (8.29) приведет к следующим селекционным вероятностям:

$$\begin{aligned} \Pr(x_1) &= 20\ \%, & \Pr(x_2) &= 40\ \%, \\ \Pr(x_3) &= 30\ \%, & \Pr(x_4) &= 10\ \%. \end{aligned} \tag{8.30}$$

Мы видим, что когда значения приспособленности тесно сгруппированы вместе, как в уравнении (8.23), ранговый отбор широко разбрасывает селекционные вероятности. В результате мы получаем более широкое различие между похожими особями в конце выполнения эволюционного алгоритма, после того как популяция начала сходиться.

■

Мы можем отрегулировать разброс селекционных вероятностей, пропустив ранги через нелинейную функцию. Например, если мы хотим добиться более широкой разницы между селекционными вероятностя-

ми особей, то можем перед применением рулеточного отбора возвести ранги в квадрат. Уравнение (8.29) станет:

$$R^2(x_1) = 4, \qquad R^2(x_2) = 16,$$
$$R^2(x_3) = 9, \qquad R^2(x_4) = 1. \tag{8.31}$$

Использование вышеуказанных значений для рулеточного отбора дает селекционные вероятности:

$$\Pr(x_1) = 13\ \%, \qquad \Pr(x_2) = 53\ \%,$$
$$\Pr(x_3) = 30\ \%, \qquad \Pr(x_4) = 4\ \%. \tag{8.32}$$

Мы видим, что возведение рангов в квадрат разбрасывает селекционные вероятности, по сравнению с уравнением (8.30). Другие типы операций на рангах (например, операция взятия квадратного корня) могли бы привести к более единообразным селекционным вероятностям.

8.7.5. *Линейное ранжирование*

Линейное ранжирование (linear ranking) – это обобщение рангового отбора. В линейном ранжировании мы принимаем вероятность отбора особи x_i равной

$$\Pr(x_i) = \alpha + \beta R(x_i), \tag{8.33}$$

где $R(x_i)$ – это ранг x, как определено в разделе 8.7.4, и α и β – определяемые пользователем параметры. На рис. 8.19 показана вероятность отбора с популяцией размером N. По мере того как прямая становится более крутой, селекционное давление увеличивается.

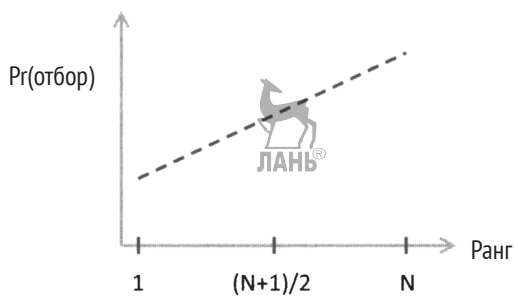


Рис. 8.19. На этом рисунке показан метод линейного ранжирования для отбора в эволюционном алгоритме с популяцией размером N. Худшая особь имеет ранг 1, а лучшая особь – ранг N

Поскольку лучшая особь имеет ранг N, а средняя особь имеет ранг $(N + 1) / 2$, селекционное давление уравнения (8.13) равняется

$$\phi = \frac{\alpha + \beta N}{\alpha + \beta(N+1)/2}. \tag{8.34}$$

Если нормализовать селекционные вероятности так, чтобы они в сумме составляли 1, то получим

$$\sum_{i=1}^{N}(\alpha + \beta i) = \alpha N + \beta N(N+1)/2 = 1. \tag{8.35}$$

Если мы хотим добиться определенного селекционного давления ϕ, то можем решить уравнения (8.34) и (8.35) для α и β, в результате получив

$$\alpha = \frac{2N - \phi(N+1)}{N(N-1)},$$
$$\beta = \frac{2(\phi - 1)}{N(N-1)}. \tag{8.36}$$

Эти формулы говорят нам о том, как устанавливать α и β, для того чтобы получить желаемое селекционное давление.

Поскольку $\Pr(x_i)$ является линейной функцией $R(x_i)$, как показано в уравнении (8.33), мы имеем

$$\Pr(\text{средний } x) = \tfrac{1}{2}[\Pr(\text{худший } x) + \Pr(\text{лучший } x)] \geq \tfrac{1}{2}\Pr(\text{лучший } x), \tag{8.37}$$

принимая допущение, что все вероятности неотрицательны. Объединив это с определением селекционного давления в уравнении (8.13), мы видим, что

$$\phi = \frac{\Pr(\text{лучший } x)}{\Pr(\text{средний } x)}. \tag{8.38}$$

Если попытаться установить $\phi > 2$, то $\Pr(\text{худший } x)$ будет меньше 0. Как правило, мы хотим добиться низкого селекционного давления на ранних стадиях эволюционного алгоритма, для того чтобы избежать преждевременного схождения, и более высокого селекционного давления на более поздних стадиях, для того чтобы эксплуатировать высоко приспособленных особей.

Линейное ранжирование и рулеточный отбор

Линейное ранжирование имеет преимущество в том, что даже если мы используем его в сочетании с рулеточным отбором, нам не прихо-

дится использовать в нашей компьютерной программе цикл, как показано на рис. 3.5. Для того чтобы увидеть, как можно избежать цикла, предположим, что мы генерируем случайное число $r \sim U[0, 1]$ для отбора. Это означает, что мы хотим отобрать m-ю особь, где

$$\begin{aligned} \sum_{i=1}^{m} \Pr(x_i) &\approx r, \\ \sum_{i=1}^{m} (\alpha + \beta R(x_i)) &\approx r, \\ \alpha m + \beta m(m+1)/2 &\approx r. \end{aligned} \tag{8.39}$$

Но это всего лишь простое квадратичное уравнение для m, которое мы можем решить в виде:

$$m = \frac{-2\alpha - \beta + \sqrt{(2\alpha+\beta)^2 + 8\beta r}}{2\beta}. \tag{8.40}$$

Разумеется, поскольку m ограничено множеством целых чисел, для получения m нам нужно округлить правую часть уравнения (8.40) до ближайшего целого числа. Следовательно, мы можем реализовать линейное ранжирование в сочетании с рулеточным отбором без цикла, в отличие от стандартного рулеточного алгоритма на рис. 3.5. На рис. 8.20 показан алгоритм рулеточного отбора с линейным ранжированием.

Выбрать задаваемое пользователем селекционное давление $\phi \in (1, 2)$.
Решить уравнение (8.36) для α и β.
$\{x_i\}$ = эволюционно-алгоритмическая популяция, отсортированная
в порядке приспособленности, $i \in [1, N]$,
где x_1 – худшая особь и x_N – лучшая особь
Сгенерировать равномерно распределенное случайное число $r \in (0, 1)$.
Решить уравнение (8.40) для m, индекса отобранного родителя.

Рис. 8.20. Псевдокод для отбора одного родителя из N особей с использованием линейного ранжирования и рулеточного отбора

Недостатком линейного ранжирования является то, что нам приходится выполнять сортировку популяционных значений приспособленности, которую нам не нужно делать в случае стандартного рулеточного отбора. Но если мы используем стационарный эволюционный алгоритм, в котором каждое поколение генерируем лишь несколько детей, то поддержка эволюционно-алгоритмической популяции в упорядоченном по приспособленности виде без полноценной сортировки в каждом поколении могла бы оказаться простой. Таким образом, явное

преимущество или недостаток от использования линейного ранжирования и уравнения (8.40) для рулеточного отбора отсутствует; все зависит от других аспектов реализации эволюционного алгоритма и предпочтений пользователя.

8.7.6. *Турнирный отбор*

Турнирный отбор (tournament selection) снижает вычислительные затраты, связанные с отбором. На рис. 3.5 показано, что рулеточный отбор N родителей из популяции N особей требует вложенных циклов, что может быть вычислительно затратно для больших популяций. В случае турнирного отбора мы случайно отбираем τ особей из популяции, где $\tau > 2$ – это определяемый пользователем размер турнира. Затем мы сравниваем значения приспособленности отобранных особей и отбираем наиболее приспособленных для рекомбинации.

В качестве анализа метода турнирного отбора рассмотрим селекционное давление, как определено в уравнении (8.13). Если для турнира отобрана наиболее приспособленная особь, то она будет отобрана для рекомбинации с вероятностью 100 %. Если для турнира отобрана средняя x, то она должна быть более приспособленной, чем $\tau - 1$ других особей в турнире, в котором они будут отобраны для рекомбинации. В этом случае существует 50%-ная вероятность того, что x будет более приспособлена, чем каждая особь, с которой она сравнивается[2], поэтому существует вероятность $(1/2)^{\tau-1}$, что x будет отобрана для рекомбинации. Тогда уравнение (8.13) становится

$$\phi = 2^{\tau-1}. \tag{8.41}$$

Мы видим, что по мере увеличения τ селекционное давление в турнирном отборе тоже увеличивается.

Вышеприведенный метод турнирного отбора называется строгим турниром, потому что лучшая особь в турнире выигрывает в 100 % случаев. Мягкий турнир – это турнир, в котором лучшая особь турнира побеждает с вероятностью $p < 1$ [Reeves и Rowe, 2003, раздел 2.3]. Другие, менее приспособленные особи тоже имеют некоторую вероятность выиграть турнир. При условии что размер турнира одинаковый, мягкие турниры имеют меньшее селекционное давление, чем строгие.

Одним из преимуществ турнирного отбора перед другими видами отбора является то, что он может работать только с субъективными сравнениями между особями. То есть для выполнения турнирного отбора

[2] Это приблизительно правильно, исходя из того, что $\tau \ll N$.

нам не нужно вычислять абсолютные значения приспособленности; нам нужно только знать относительные значения приспособленности особей в турнире.

8.7.7. *Племенные эволюционные алгоритмы*

Многие эволюционные алгоритмы вероятностно отбирают особей для рекомбинации, опираясь на их относительные значения приспособленности. Все методы, которые мы обсуждали до сих пор, следуют этому принципу. Однако в племенных эволюционных алгоритмах, то есть на основе племенного жеребца[3], мы каждое поколение для каждой операции рекомбинации всегда отбираем лучшую особь. Лучшая особь каждого поколения называется племенным жеребцом. Затем мы обычным способом (например, с помощью пропорционального отбора на основе приспособленности, рангового отбора, турнирного отбора и т. д.) отбираем других родителей, с которыми племенной жеребец комбинируется для создания ребенка. Эта идея была впервые применена к генетическим алгоритмам, и в этом случае она называется племенным генетическим алгоритмом [Khatib и Fleming, 1998]. Добавление племенной логики в генетический алгоритм приводит к модификации стандартного генетического алгоритма из рис. 3.6 и к племенному генетическому алгоритму на рис. 8.21.

Родители ← {случайно сгенерированная популяция}
While not (критерий останова)
  Рассчитать приспособленность каждого родителя в популяции.
  Дети ← ∅
  x_1 ← наиболее приспособленный родитель
  While |Дети| < |Родители|
    Применить приспособленности для вероятностного отбора
      второго родителя $x_2 : x_2 \neq x_1$.
    Спарить x_1 и x_2 для создания детей c_1 и c_2.
    Дети ← Дети ∪ {c_1, c_2}
  Loop
  Случайно мутировать нескольких детей.
  Родители ← Дети
Next поколение

Рис. 8.21. Приведенный выше псевдокод схематично описывает племенной генетический алгоритм

[3] Племенной эволюционный алгоритм (stud EA) – это эволюционный алгоритм, в котором вместо применения стохастического отбора используется наиболее приспособленная особь, племенной жеребец (stud). Такая особь делится своей генетической информацией со всеми другими, используя простые операторы генетического алгоритма. См. https://www.researchgate.net/publication/220702114_The_stud_GA_A_mini_revolution. – *Прим. перев.*

■ Пример 8.11

В данном примере мы симулируем непрерывный генетический алгоритм с племенной опцией и без нее на множестве 20-мерных эталонных задач из приложения *C*. Мы используем популяции размером 50, лимит поколений 50 и скорость мутации 1 % для каждого из 20 признаков в каждой особи в каждом поколении. Мы реализуем мутацию, заменяя независимую переменную на переменную, которую случайно отбираем из равномерного распределения между минимальным и максимальным значениями области. Мы также используем параметр элитарности, равный двум, то есть сохраняем двух лучших особей из поколения в поколение.

В табл. 8.1 показана лучшая результативность стандартного генетического алгоритма и племенного генетического алгоритма, усредненно по 50 симуляциям Монте-Карло. Таблица показывает, что племенной генетический алгоритм явно превосходит стандартный генетический алгоритм для показанных эталонов. При наличии племенной опции результативность отдельных эталонов драматически улучшается.

Таблица 8.1. Результаты примера 8.11, показывающие относительную результативность генетического алгоритма с племенной опцией и без нее. В таблице показан нормализованный минимум, найденный двумя версиями генетического алгоритма, усредненно по 50 симуляциям Монте-Карло. См. определения эталонных функций в приложении C

Эталон	Неплеменной ГА	Племенной ГА
Ackley	1.44	1
Fletcher	3.26	1
Griewank	3.96	1
Penalty #1	1.05×10^5	1
Penalty #2	160.8	1
Quartic	9.14	1
Rastrigin	1.92	1
Rosenbrock	3.89	1
Schwefel 1.2	1.24	1
Schwefel 2.21	1.65	1
Schwefel 2.22	3.70	1

Эталон	Неплеменной ГА	Племенной ГА
Schwefel 2.26	2.56	1
Sphere	4.47	1
Step	4.23	1

■

До сего момента племенной эволюционный алгоритм в основном (возможно, исключительно) применялся в генетических алгоритмах в исследовательской литературе, но мы можем его легко применить и ко многим другим эволюционным алгоритмам. Добавление племенной логики в эволюционный алгоритм сводится к внесению простого изменения в базовый эволюционный алгоритм, как мы видим из сравнения между рис. 3.6 и 8.21. Из-за своей легкости реализации и превосходной результативности, показанной в табл. 8.1, мы должны серьезно рассмотреть внедрение племенной логики в наши эволюционные алгоритмы. Еще одной интересной областью будущих исследований было бы выведение математических моделей для генетических алгоритмов (глава 4) и других эволюционных алгоритмов с учетом племенной логики.

8.8. Рекомбинация

В простом генетическом алгоритме используется одноточечное скрещивание. В этом разделе рассматриваются другие типы рекомбинации для бинарных и непрерывных эволюционных алгоритмов. Обратите внимание, что мы используем термины скрещивание и рекомбинация взаимозаменяемо. Некоторые из методов рекомбинации, которые мы представим в этом разделе, подробнее обсуждаются в публикации [Herrera и соавт., 1998].

Предположим, что у нас есть популяция особей $\{x_1, x_2, \ldots, x_N\}$. Каждая особь имеет n признаков. Обозначим k-й признак i-й особи как $x_i(k)$ для $k \in [1, n]$. Таким образом, мы можем представить x_i как вектор

$$x_i = [x_i(1) \quad x_i(2) \quad \ldots \quad x_i(n)]. \tag{8.42}$$

Мы обозначаем детскую особь, то есть ребенка, который является результатом рекомбинации, как y. Обозначим k-й признак ребенка как $y(k)$, поэтому

$$y = [y(1) \quad y(2) \quad \ldots \quad y(n)]. \tag{8.43}$$

8.8.1. *Одноточечное скрещивание (в бинарных или непрерывных эволюционных алгоритмах)*

Предположим, что у нас есть два родителя, x_a и x_b, где $a \in [1, N]$ и $b \in [1, N]$. Одноточечное скрещивание, так называемое простое скрещивание или дискретное скрещивание, является типом скрещивания, который впервые был использован в бинарном генетическом алгоритме (см. главу 3):

$$y(k) \leftarrow [x_a(1) \ \ldots \ x_a(m) \quad x_b(m+1) \ \ldots \ x_b(n)], \tag{8.44}$$

где m – это случайно отобранная точка скрещивания, то есть $m \sim U[0, n]$. Если $m = 0$, то y является клоном x_b. Если $m = n$, то y является клоном x_a. Одноточечное скрещивание часто реализуется для получения двух детей от пары родителей. Это делается путем отбора каждого признака второго ребенка y_2 у противоположного родителя, чем у того, у которого y_1 получил свой признак:

$$\begin{aligned} y_1(k) &\leftarrow [x_a(1) \ \ldots \ x_a(m) \quad x_b(m+1) \ \ldots \ x_b(n)] \\ y_2(k) &\leftarrow [x_b(1) \ \ldots \ x_b(m) \quad x_a(m+1) \ \ldots \ x_a(n)] \end{aligned} \tag{8.45}$$

8.8.2. *Многоточечное скрещивание (в бинарных или непрерывных эволюционных алгоритмах)*

Двухточечное скрещивание сводится к

$$\begin{aligned} y(k) \leftarrow [& x_a(1) \ \ldots \ x_a(m_1) \\ & x_b(m_1+1) \ \ldots \ x_b(m_2) \\ & x_a(m_2+1) \ \ldots \ x_a(n)] \end{aligned} \tag{8.46}$$

где двумя точками скрещивания являются $m_1 \sim U[0, n]$ и $m_2 \sim U[m_1 + 1, n]$. Если $m_1 = 0$ или $m_2 = n$, то двухточечное скрещивание сводится к одноточечному скрещиванию. Если $m_1 = n$, то y является клоном x_a. Уравнение (8.46) может быть расширено до трехточечного скрещивания или M-точечного скрещивания для любого $M > 2$. Как и в случае с одноточечным скрещиванием, многоточечное скрещивание часто реализуется для получения двух детей от пары родителей.

8.8.3. *Сегментированное скрещивание (в бинарных или непрерывных эволюционных алгоритмах)*

Сегментированное скрещивание [Michalewicz, 1996, раздел 4.6] можно представить как обощение многоточечного скрещивания. Ребенок 1

получает свой первый признак от родителя 1. Затем, с вероятностью ρ, мы переключаемся на родителя 2 и получаем второй для ребенка 1 признак; и с вероятностью $(1 - \rho)$ мы получаем второй для ребенка 1 признак от родителя 1. Каждый раз, когда мы получаем признак для ребенка 1, то переключаемся на другого родителя, чтобы получить следующий признак с вероятностью ρ. Ребенок 1 и ребенок 2 получают свои признаки от разных родителей, поэтому если ребенок 1 получает признак k от родителя 1, то ребенок 2 получает признак k от родителя 2, для $k \in [1, n]$. Аналогичным образом, если ребенок 1 получает признак k от родителя 2, то ребенок 2 получает признак k от родителя 1. Сегментированное скрещивание эквивалентно многоточечному скрещиванию, где число точек скрещивания является случайным числом. На рис. 8.22 показан алгоритм сегментированного скрещивания. Вероятность переключения ρ часто принимается равной примерно 0.2.

$S \leftarrow$ true
For $k = 1$ **to** n
  If S **then**
    $c_1(k) \leftarrow p_1(k)$
    $c_2(k) \leftarrow p_2(k)$
  else
    $c_1(k) \leftarrow p_2(k)$
    $c_2(k) \leftarrow p_1(k)$
  End if
  $r \leftarrow U[0, 1]$
  If $r < \rho$ **then** $S \leftarrow$ not S
Next признак решения

Рис. 8.22. Сегментированное скрещивание n-мерных особей. p_1 и p_2 являются двумя родителями, а c_1 и c_2 – их двумя детьми

8.8.4. *Равномерное скрещивание (в бинарных или непрерывных эволюционных алгоритмах)*

Предположим, что у нас есть два родителя, x_a и x_b. Равномерное скрещивание [Ackley, 1987a], [Michalewicz и Schoenauer, 1996] приводит к ребенку y, где k-й признак ребенка y задается следущим образом:

$$y(k) \leftarrow x_{i(k)}(k) \tag{8.47}$$

для каждого $k \in [1, n]$, где мы случайно отбираем $i(k)$ из множества $\{a, b\}$. То есть мы случайно отбираем каждый детский признак у одного из двух родителей, каждый с вероятностью 50 %.

8.8.5. *Многородительское скрещивание (в бинарных или непрерывных эволюционных алгоритмах)*

Многородительское скрещивание, которое мы здесь обсуждаем, является обобщением однородного скрещивания [Eiben, 2003], [Eiben и Bäck, 1998], [Eiben, 2000] и употребляется под несколькими другими названиями, в том числе генофондовая рекомбинация [Bäck, 1996], [Bäck и и соавт., 1997b], [Mtihlenbein и Voigt, 1995], сканирующее скрещивание [Eiben и Schippers, 1996], а также многополое скрещивание [Schwefel, 1995] (в отличие от двухродительского скрещивания, которое называется двуполым скрещиванием). В многородительском скрещивании мы случайно отбираем каждый детский признак от одного из родителей, где число родителей больше двух. Этот метод был предложен еще в 1966 году [Bremermann и соавт., 1966]. Многородительское скрещивание дает

$$y_k \leftarrow x_{i(k)}(k) \tag{8.48}$$

для каждого $k \in [1, n]$, где мы случайно отбираем $i(k)$ из подмножества $[1, N]$ (напомним, что в популяции есть N потенциальных родителей). При реализации многородительского скрещивания нам нужно принять несколько решений. Например, сколько особей должно быть в фонде потенциальных родителей? Как отбирать особей для родительского фонда? После того как фонд был определен, как отбирать родителей из родительского фонда? Наконец, отметим, что к многородительскому скрещиванию также предлагались и другие подходы [Eiben95].

8.8.6. *Глобальное однородное скрещивание (в бинарных или непрерывных эволюционных алгоритмах)*

Один из способов реализации многородительского скрещивания заключается в случайном отборе каждого детского признака у одного из родителей, где родительский фонд эквивалентен всей популяции. В результате получается глобальная равномерная рекомбинация:

$$y_k \leftarrow x_{i(k)}(k) \tag{8.49}$$

для каждой $k \in [1, n]$, где $i(k)$ отбирается случайно из равномерного распределения $[1, N]$ для каждой k. В качестве альтернативы $i(k)$ может быть отобрана на основе приспособленности. То есть вероятность того, что $i(k) = m$, может быть пропорциональна приспособленности x_m для всех $k \in [1, n]$ и $m \in [1, N]$.

8.8.7. *Перетасованное скрещивание (в бинарных или непрерывных эволюционных алгоритмах)*

Перетасованное скрещивание случайно переставляет признаки решения в родителях [Eshelman и соавт., 1989]. Во всех родителях, которые вносят вклад в данного ребенка, мы используем одну и ту же перестановку признаков решения. Затем мы выполняем один из упомянутых выше методов скрещивания (обычно одноточечное скрещивание) для получения детей. Потом отменяем перестановку признаков решения в ребенке. На рис. 8.23 показан алгоритм перетасованного скрещивания в сочетании с одноточечным скрещиванием.

$\{r_1, ..., r_n\} \leftarrow$ случайная перестановка элементов $\{1, ..., n\}$
Точка скрещивания $m \leftarrow U[1, n-1]$
For $k = 1$ **to** m
    $t_1(k) \leftarrow p_1(r_k)$
    $t_2(k) \leftarrow p_2(r_k)$
Next k
For $k = m + 1$ **to** n
    $t_1(k) \leftarrow p_2(r_k)$
    $t_2(k) \leftarrow p_1(r_k)$
Next k
For $k = 1$ **to** n
    $c_1(r_k) \leftarrow t_1(k)$
    $c_2(r_k) \leftarrow t_2(k)$
Next k

Рис. 8.23. Перетасованное скрещивание в сочетании с одноточечным скрещиванием для n-мерных родителей. p_1 и p_2 – это два родителя, t_1 и t_2 – два ребенка перед отменой перестановки, c_1 и c_2 – неперемешанные дети

8.8.8. *Плоское скрещивание и арифметическое скрещивание (в непрерывных эволюционных алгоритмах)*

Плоское скрещивание, так называемое арифметическое скрещивание, описывается следующим образом:

$$\begin{aligned} y(k) &\leftarrow U[x_a(k), x_b(k)] \\ &= \alpha x_a(k) + (1 - \alpha) x_b(k), \end{aligned} \tag{8.50}$$

где $\alpha \sim U[0, 1]$. То есть $y(k)$ – это случайное число, взятое из равномерного распределения между k-ми признаками двух его родителей. Это

равносильно утверждению, что ребенок представляет собой линейную комбинацию признаков двух своих родителей. Иногда плоское и арифметическое скрещивания отличаются тем, что плоское скрещивание дает одного ребенка, тогда как арифметическое скрещивание дает двух детей:

$$\begin{aligned} &\text{плоское скрещивание:} && y(k) = \alpha x_a(k) + (1-\alpha)x_b(k); \\ &\text{арифметическое скрещивание:} && \begin{cases} y_1(k) = \alpha x_a(k) + (1-\alpha)x_b(k) \\ y_2(k) = (1-\alpha)x_a(k) + \alpha x_b(k). \end{cases} \end{aligned} \tag{8.51}$$

Вместо равномерной функции плотности вероятности (PDF) для α мы можем также применить треугольную функцию плотности:

$$\text{PDF}(\alpha) = \begin{cases} 1+\alpha & \text{если } -1 \le \alpha < 0 \\ 1-\alpha & \text{если } 0 \le \alpha \le 1 \end{cases} \tag{8.52}$$

В этом случае уравнение (8.51) называется нечеткой рекомбинацией [Eshelman и Schaffer, 1993].

8.8.9. *Смешанное скрещивание (в непрерывных эволюционных алгоритмах)*

Смешанное скрещивание, которое также называется скрещиванием BLX-α и эвристическим скрещиванием [Houck и соавт., 1995], объединяет родителей x_a и x_b следующим образом:

$$\begin{aligned} x_{\min}(k) &\leftarrow \min(x_a(k),\ x_b(k)) \\ x_{\max}(k) &\leftarrow \max(x_a(k),\ x_b(k)) \\ \Delta x(k) &\leftarrow x_{\max}(k) - x_{\min}(k) \\ y(k) &\leftarrow U[x_{\min}(k) - \alpha\Delta x(k),\ x_{\max}(k) + \alpha\Delta x(k)] \end{aligned} \tag{8.53}$$

где α – определяемый пользователем параметр. Если $\alpha = 0$, то смешанное скрещивание эквивалентно плоскому скрещиванию. Если $\alpha < 0$ (с нижним пределом –0.5), то смешанное скрещивание сжимает поисковую область, что выгодно для эксплуатации текущей популяции. Если $\alpha > 0$, то смешанное скрещивание расширяет поисковую область, что полезно для разведывания. В публикации [Herrera и соавт., 1998] рекомендуется $\alpha = 0.5$.

8.8.10. *Линейное скрещивание (в непрерывных эволюционных алгоритмах)*

Линейное скрещивание создает трех детей от родителей x_a и x_b:

$$\begin{aligned} y_1(k) &\leftarrow (1/2)x_a(k) + (1/2)x_b(k) \\ y_2(k) &\leftarrow (3/2)x_a(k) + (1/2)x_b(k) \\ y_3(k) &\leftarrow (-1/2)x_a(k) + (3/2)x_b(k) \end{aligned} \tag{8.54}$$

В зависимости от конкретной реализации эволюционного алгоритма мы оставляем наиболее приспособленного либо двух наиболее приспособленных из трех детей для следующего поколения.

8.8.11. *Симулированное бинарное скрещивание (в непрерывных эволюционных алгоритмах)*

Симулированное бинарное скрещивание создает следующих двух детей от родителей x_a и x_b [Deb и Agrawal, 1995]:

$$\begin{aligned} y_1(k) &\leftarrow (1/2)[(1 - \beta_k)x_a(k) + (1 + \beta_k)x_b(k)] \\ y_2(k) &\leftarrow (1/2)[(1 + \beta_k)x_a(k) + (1 - \beta_k)x_b(k)] \end{aligned} \tag{8.55}$$

где β_k – это случайное число, генерируемое из следующей ниже функции плотности вероятности (PDF):

$$\text{PDF}(\beta) = \begin{cases} 1/2(\eta + 1)\beta^{\eta} & \text{если } 0 \le \beta \le 1 \\ 1/2(\eta + 1)\beta^{-(\eta+2)} & \text{если } \beta > 1 \end{cases} \tag{8.56}$$

где η – любое неотрицательное вещественное число. Публикация [Deb и Agrawal, 1995] содержит обсуждение эффекта η на оператор симулированного бинарного скрещивания. В ней в целом рекомендуются значения в диапазоне от 0 до 5. Мы можем сгенерировать β с помощью следующего алгоритма:

$$\begin{aligned} r &\leftarrow U[0, 1] \\ \beta &\leftarrow \begin{cases} (2r)^{1/(\eta+1)} & \text{если } r \le 1/2 \\ (2 - 2r)^{-1(\eta+1)} & \text{если } r > 1/2 \end{cases} \end{aligned} \tag{8.57}$$

Обратите внимание, что симулированное бинарное скрещивание эквивалентно арифметическому скрещиванию в уравнении (8.51), если $\beta = 2\alpha - 1$. Мы можем также реализовать симулированное бинарное

скрещивание со значениями β_k, которые имеют распределения, отличные от уравнения (8.56).

8.8.12. *Резюме*

Рассмотренные выше методы рекомбинации были первоначально предложены для генетических алгоритмов, но они также могут использоваться в других эволюционных алгоритмах. Исследователи предложили и другие методы скрещивания [Herrera и соавт., 1998], однако вышеуказанные подходы предоставляют основные идеи. Кроме того, мы можем объединять некоторые из этих подходов и создавать свои собственные алгоритмы рекомбинации. Среди этих методов скрещивания нет явного победителя. Один метод скрещивания мог бы показывать лучшие результаты на одной задаче, в то время как другой работает лучше всего на иной задаче. Тем не менее даже если мы не можем выделить лучший метод скрещивания, то обычно можем сказать, что одноточечное скрещивание является одним из худших.

8.9. Мутация

В бинарных эволюционных алгоритмах мутация является простой операцией. Если у нас есть популяция из N особей, где каждая особь имеет n бит, и наша скорость мутации равна ρ, то в конце каждого поколения мы переворачиваем каждый бит в каждой особи с вероятностью ρ, как показано в уравнении (3.6).

В непрерывных эволюционных алгоритмах у нас вариантов для мутации больше. Мы по-прежнему называем ρ скоростью мутации и по-прежнему модифицируем $x_i(k)$ с вероятностью ρ для каждого i и каждого k. Но если мы решим модифицировать $x_i(k)$, то тогда нам нужно решить, каким образом следует модифицировать $x_i(k)$. Одним из способов является создание $x_i(k)$ из равномерного или гауссова распределения, среднее значение которого находится в центре поисковой области. Другой способ – создавать $x_i(k)$ из равномерного или гауссова распределения, среднее значение которого находится на немутированном значении $x_i(k)$. Мы опишем эти варианты ниже, где будем использовать $x_{\min}(k)$ и $x_{\max}(k)$ для обозначения пределов поисковой области k-й размерности в нашей оптимизационной задаче.

8.9.1. *Равномерная мутация с центром в $x_i(k)$*

Равномерная мутация с центром в $x_i(k)$ может быть записана как

$$r \leftarrow U[0, 1]$$
$$x_i(k) \leftarrow \begin{cases} x_i(k) & \text{если } r \geq \rho \\ U[x_i(k) - \alpha_i(k),\ x_i(k) + \alpha_i(k)] & \text{если } r < \rho \end{cases} \quad (8.58)$$

для $i \in [1, N]$ и $k \in [1, n]$, где $\alpha_i(k)$ – это определяемый пользователем параметр, который задает величину мутации. Мы часто выбираем $\alpha_i(k)$ как можно больше, обеспечивая при этом, чтобы мутация оставалась в поисковой области:

$$\alpha_i(k) = \min(x_i(k) - x_{\min}(k),\ x_{\max}(k) - x_i(k)). \quad (8.59)$$

8.9.2. *Равномерная мутация с центром в середине поисковой области*

Равномерная мутация с центром в середине поисковой области может быть записана как

$$r \leftarrow U[0, 1]$$
$$x_i(k) \leftarrow \begin{cases} x_i(k) & \text{если } r \geq \rho \\ U[x_{\min}(k),\ x_{\max}(k)] & \text{если } r < \rho \end{cases} \quad (8.60)$$

для $i \in [1, N]$ и $k \in [1, n]$.

8.9.3. *Гауссова мутация с центром в $x_i(k)$*

Гауссова мутация с центром в $x_i(k)$ может быть записана как

$$r \leftarrow U[0, 1]$$
$$x_i(k) \leftarrow \begin{cases} x_i(k) & \text{если } r \geq \rho \\ \max[\min(x_{\max}(k), N(x_i(k), \sigma_i^2(k)),\ x_{\min}(k)] & \text{если } r < \rho \end{cases} \quad (8.61)$$

для $i \in [1, N]$ и $k \in [1, n]$, где $\sigma_i(k)$ – это определяемый пользователем параметр, пропорциональный величине мутации. Операции *min* и *max* гарантируют, что модифицированное значение $x_i(k)$ останется в поисковой области. Данный тип мутации подобен операторам поиска, которые мы используем в эволюционном программировании и эволюционных стратегиях.

8.9.4. *Гауссова мутация с центром в середине поисковой области*

Гауссова мутация с центром в середине поисковой области может быть записана как

$$r \leftarrow U[0, 1]$$
$$x_i(k) \leftarrow \begin{cases} x_i(k) & \text{если } r \geq \rho \\ \max[\min(x_{\max}(k), N(c_i(k), \sigma_i^2(k)), x_{\min}(k)] & \text{если } r < \rho \end{cases} \quad (8.62)$$

для $i \in [1, N]$ и $k \in [1, n]$, где $c_i(k) = (x_{\min}(k) + x_{\max}(k))/2$ – это центр поисковой области и где $\sigma_i(k)$ – определяемый пользователем параметр, пропорциональный величине мутации. Операции *min* и *max* гарантируют, что модифицированное значение $x_i(k)$ останется в поисковой области.

8.10. Заключение

В этой главе мы рассмотрели целый ряд вариаций эволюционных алгоритмов, но на самом деле ограничили наше обсуждение только наиболее распространенными вариациями. Существует ряд других способов модификации эволюционных алгоритмов, таких как использование разных размеров популяции [Hu и соавт., 2010], взаимодействующих субпопуляций [Li и соавт., 2009], диплоидии или полиплоидии, где каждая особь связывается с несколькими кандидатными решениями [Wang и соавт., 2009], и гендерного моделирования, которое ограничивает скрещивание родителями противоположного пола [Mitchell, 1998]. Исследователи также предложили иные модификации, но у нас здесь просто не хватит места, чтобы обсудить все вариации, которые были изучены за последние несколько десятилетий.

Учитывая все доступные вариации эволюционных алгоритмов, полезно проводить различие между эволюционным алгоритмом и экземпляром эволюционного алгоритма [Eiben и Smit, 2011]. Эволюционный алгоритм – это общий каркас, определяющий подход к оптимизации, который включает в себя популяцию кандидатных решений, отбор, рекомбинацию и мутацию; а экземпляр эволюционного алгоритма – это реализация данного каркаса, которая включает в себя конкретные подходы к решению этих задач и специфические регулировочные параметры (например, специфически настроенные генетические алгоритмы, эволюционные стратегии либо любой другой конкретный

алгоритм, рассмотренный в настоящей книге). В этом ракурсе все экземпляры эволюционного алгоритма рассматриваются как конкретные реализации общего эволюционно-алгоритмического каркаса, что полезно для унификации данной области и предотвращения его раскола на явно несвязанные фрагменты. Эта унифицированная перспектива эволюционных алгоритмов лежит в основе книг [De Jong, 2002], [Eiben и Smith, 2010].

Обратите внимание, что регулировку параметров эволюционных алгоритмов можно рассматривать в двух разных перспективах. Во-первых, мы можем регурировать параметры для оптимизации работы эволюционного алгоритма, а во-вторых, можем регулировать параметры для изучения того, как результативность зависит от настроек параметров [Eiben и Smit, 2011]. Эта вторая перспектива тесно связана с надежностью эволюционных алгоритмов, которую мы кратко обсудим в разделе 21.4.

В будущем, несомненно, появятся новые и изобретательные вариации эволюционных алгоритмов. Наиболее сложным аспектом таких исследований является сначала тщательное изучение прошлых исследований, чтобы убедиться, что предположительно новые идеи уже не были опубликованы. В современной литературе есть бесчисленные примеры того, как из-за незнания авторами и рецензентами прошлых исследований изобретается колесо. Иногда алгоритмы изобретаются и получают разные имена от разных авторов, а иногда новые алгоритмы изобретаются, но получают то же имя, что и совершенно другой алгоритм. Справедливости ради, трудно идти в ногу с происходящим в последние несколько десятилетий взрывным ростом количества литературы по эволюционным алгоритмам, и все мы в определенной степени незнакомы с прошлыми исследованиями. Тем не менее когда мы документируем наше исследование, то несем ответственность перед нашими читателями и прошлыми исследователями, состоящую в том, чтобы тщательно изучать опубликованную литературу, благодаря чему мы сможем поместить наше исследование в надлежащий контекст.

Задачи

Письменные упражнения

8.1. Предположим, что вы инициализируете популяцию из N особей, равномерно распределенных в одномерной поисковой области $[x_{\min}, x_{\max}]$, где $x_{\max} = -x_{\min}$.

a) Какова вероятность того, что хотя бы одна из этих особей окажется в пределах ϵ оптимальной точки в области?

b) Предположим, что $\epsilon/x_{\max} \ll 1$ и что N «не слишком велико». Примените аппроксимации на основе разложения в ряды Тейлора, чтобы найти коэффициент, на который ваш ответ на вопрос в части (а) увеличивается, если исходная популяция удваивается.

8.2. Коды Грея не являются уникальными. Уравнение (8.2) показывает кодировку Грея чисел 0–7. Дайте альтернативный код Грея.

8.3. Элитарность и эволюционные стратегии

a) Объясните, как элитарный генетический алгоритм похож на эволюционную стратегию $(\mu + \lambda)$.

b) Какие значения мы будем использовать для N и E на рис. 8.7, а также для μ и λ на рис. 6.10, для того чтобы получить максимально похожие элитарный генетический алгоритм и эволюционную стратегию $(\mu + \lambda)$?

8.4. Элитарность и стационарная эволюция

a) Как объединить вариант элитарности 1 на рис. 8.6 со стационарной эволюцией?

b) Как объединить вариант элитарности 2 на рис. 8.7 со стационарной эволюцией?

8.5. Предположим, что у нас есть эволюционный алгоритм с разрывом поколений k. Сколько поколений этого эволюционного алгоритма вычислительно эквивалентно G поколениям поколенческого эволюционного алгоритма?

8.6. Сколько сравнений необходимо выполнить в популяции размера N, чтобы выполнить полную проверку на наличие дубликатов?

8.7. Напишите последовательность уравнений для преобразования значений стоимости (где чем меньше, тем лучше) в модифицированные значения стоимости с использованием обмена информации о приспособленности.

8.8. Следует ли нам беспокоиться о возможном делении на ноль в уравнении (8.7)?

8.9. Метод очистки из раздела 8.6.3.2 может привести к тому, что самая приспособленная особь в нише станет недоступной для отбо-

ра и рекомбинации. Изобразите наглядный пример ситуации, в которой это может произойти.

8.10. Селекционное давление

а) Каково селекционное давление при рулеточном отборе со значениями приспособленности, показанными на рис. 8.16?

б) Каково селекционное давление стохастической универсальной выборки со значениями приспособленности, показанными на рис. 8.17?

8.11. Стохастическая универсальная выборка

а) Какова вероятность того, что наиболее приспособленная особь на рис. 8.17 будет отобрана дважды с использованием стохастической универсальной выборки?

b) Какова вероятность того, что наименее приспособленная особь на рис. 8.17 будет отобрана один раз с использованием стохастической универсальной выборки?

8.12. Предположим, что в эволюционно-алгоритмической популяции есть четыре особи со значениями приспособленности 10, 20, 30 и 40.

а) Каковы вероятности отбора каждой особи при наличии одного вращения колеса рулетки?

b) Предположим, что мы используем избыточный отбор так, что лучшие 50 % популяции имеют 75%-ную вероятность отбора, а худшие 50 % популяции имеют 25%-ную вероятность отбора. Каковы вероятности отбора каждой особи при наличии одного вращения колеса рулетки?

c) Каковы вероятности отбора, если мы используем сигма-шкалирование?

d) Каковы вероятности отбора, если мы используем ранговый отбор?

8.13. Предположим, что в популяции эволюционного алгоритма есть четыре особи со значениями приспособленности 10, 20, 30 и 40. Примените уравнение (8.36) для вычисления α и β, чтобы получить следующие значения селекционного давления при использовании линейного ранжирования.

a) $\phi = 1.4$.
b) $\phi = 1.6$.
c) $\phi = 1.8$.

8.14. Предположим, что вы используете мягкий турнир для отбора с размером турнира, равным 3, где вероятность отбора лучшей особи в турнире составляет 70 %, вероятность отбора второй лучшей особи равна 20 %, вероятность отбора худшей особи равна 10 %. Каково селекционное давление этого турнира?

8.15. Предположим, вы используете эволюционный алгоритм для решения 20-мерной задачи.

a) Какова вероятность того, что дети, произведенные одноточечным скрещиванием, будут клонами родителей?

b) Какова вероятность того, что дети, произведенные двухточечным скрещиванием, будут клонами родителей?

c) Какова вероятность того, что дети, произведенные сегментированным скрещиванием с $\rho = 0.2$, будут клонами родителей?

d) Какова вероятность того, что дети, произведенные равномерным скрещиванием, будут клонами родителей?

Компьютерные упражнения

8.16. Реализуйте непрерывный генетический алгоритм с элитарностью для минимизации 10-мерной функции Экли. Выполните генетический алгоритм для 50 поколений с популяцией размером 50 и вероятностью мутации 1 %. Выполните генетический алгоритм 20 раз и постройте график среднего значения (усредненно по 20 симуляциям) минимальной стоимости в зависимости от номера поколения. Выполните это для 0 элитарных особей, 2 элитарных особей, 5 элитарных особей, 10 элитарных особей. Поместите четыре графика в один рисунок, чтобы облегчить сравнение.

8.17. Реализуйте непрерывный генетический алгоритм с тремя типами скучивания, рассмотренными в разделе 8.6.3.3, для того чтобы минимизировать 10-мерную функцию Экли. Используйте параметры скучивания, рекомендованные в том разделе, и популяцию размером 40, скорость мутации 2 %, параметр элитарности 2 и заменяйте дубликаты в конце каждого поколения случайно сгенерированными особями. Выполните каждый генетический алгоритм для 1000 оцениваний функций. (Обратите внимание, что разные типы скучивания дают разное число оцениваний функ-

ций на поколение. Следовательно, для того чтобы выполнить справедливое сравнение между типами скучивания, нам нужно выполнить генетические алгоритмы для разных лимитов поколений.) Сообщите о минимальной стоимости, достигнутой генетическим алгоритмом для случая без скучивания и для рассмотренных в данной главе трех типов скучивания, усредненно по 20 симуляциям Монте-Карло.

8.18. Напишите программу для численного подтверждения ваших ответов на вопросы задачи 8.11.

8.19. Предположим, что у вас есть особи задачи 8.13 и линейное ранжирование с селекционным давлением $\phi = 1.6$.

a) Просимулируйте уравнение (8.40) несколько тысяч раз и запишите процент случаев, когда отбиралась каждая особь. Примените округление в уравнении (8.40), для того чтобы получить целочисленный индекс отобранной особи. Как результаты симулирования соотносятся с теоретическими вероятностями отбора?

b) Каково для этой задачи максимально возможное значение уравнения (8.40) (без округления)? Как это помогает объяснить несоответствие между симулированными и теоретическими результатами?


ЛАНЬ®


ЛАНЬ®

Часть III

Более поздние эволюционные алгоритмы

Глава 9

Симулированное закаливание

Мы предполагаем, что аналогия с термодинамикой может предложить новое понимание оптимизационных задач и эффективные алгоритмы их решения.

– В. Черны [Černý, 1985]

Симулированное закаливание (simulated annealing, SA) – это алгоритм оптимизации, основанный на охлаждающем и кристаллизующем поведении химических веществ. В литературе часто проводится различие между симулированным закаливанием и эволюционными алгоритмами, поскольку симулированное закаливание не включает популяцию кандидатных решений. Симулированное закаливание – это стохастический алгоритм с одной особью. Вместе с тем эволюционная стратегия (1 + 1) на самом деле является частным случаем алгоритма симулированного закаливания [Droste и соавт., 2002], поэтому мы можем разумно рассматривать симулированное закаливание как эволюционный алгоритм.

Симулированное закаливание было впервые представлено в его нынешнем виде Скоттом Киркпатриком (Scott Kirkpatrick), Чарльзом Гелаттом (Charles Gelatt) и Марио Векки (Mario Vecchi) в 1983 году для оптимального решения задач, связанных с компьютерным проектированием, таких как размещение компонентов схем и маршрутизация проводов [Kirkpatrick и соавт., 1983]. Симулированное закаливание было независимо получено Владом Черны в 1985 году, который использовал его для решения задачи коммивояжера [Černý, 1985]. Алгоритм оптимизации, очень похожий на симулированное закаливание, был разработан Мартином Пинкусом (Martin Pincus) в конце 1960-х годов [Pincus, 1968a], [Pincus, 1968b]. Симулированное закаливание иногда называют алгоритмом Метрополиса, потому что он тесно связан с исследования-

ми Николаса Метрополиса [Metropolis и соавт., 1953], чья разработка алгоритма разведывания свойств взаимодействующих частиц легла в основу алгоритма симулированного закаливания. Наконец, симулированное закаливание также иногда называют алгоритмом Метрополиса–Гастингса из-за работы У. Кейта Гастингса [Hastings, 1970], который обобщил результаты Метрополиса и его соавторов.

Краткий обзор главы

В разделе 9.1 дается краткое описание статистической механики, которая является основополагающим принципом симулированного закаливания. В разделе 9.2 представлен простой алгоритм симулированного закаливания. В разделе 9.3 обсуждаются различные режимы охлаждения, которое для симулированного закаливания является первичным регулировочным параметром и имеет самое сильное влияние на его работу. В разделе 9.4 кратко обсуждаются некоторые вопросы реализации, включая способы создания новых кандидатных решений в симулированном закаливании, время реинициализации температуры охлаждения и необходимость отслеживания лучшего кандидатного решения.

9.1. Закалка в природе

Кристаллические решетки – это завораживающие примеры оптимизационной способности природы. Кристаллическая решетка представляет собой расположение атомов или молекул в жидкости или твердом теле. Некоторые знакомые примеры, распространенные в повседневном опыте большинства людей, – это кристаллические структуры кварца, льда и соли. При высоких температурах кристаллические материалы не проявляют значительной структуры; высокие температуры придают материалам много энергии, которая способствует появлению большой вибрации и беспорядка. Однако, по мере того как температура уменьшается, кристаллические материалы приобретают более упорядоченное состояние. Конкретное состояние, которое они приобретают, не всегда одинаково. Нагретый и затем многократно охлажденный материал всякий раз будет приобретать разное состояние равновесия, но каждое состояние равновесия будет стремиться к тому, чтобы иметь низкую энергию. На рис. 9.1 сравниваются кристаллическая структура с высокой энтропией (высоким уровнем беспорядка) при высокой температуре и кристаллическая структура с низкой энтропией (высоким

уровнем порядка) при низкой температуре. Процесс нагрева и охлаждения материала для его перекристаллизации называется закаливанием[1].

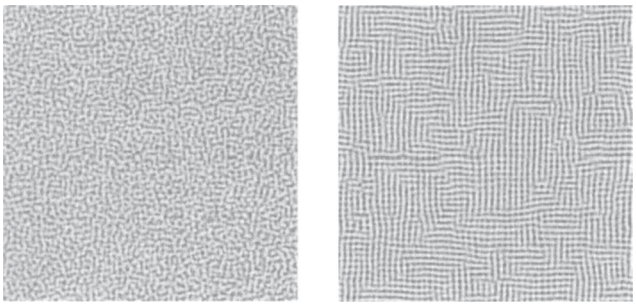


Рис. 9.1. Этот рисунок дает концептуальное представление о процессе закаливания. На рисунке слева показано неупорядоченное, высокоэнергетическое состояние кристаллической структуры при высокой температуре. На рисунке справа показано упорядоченное низкоэнергетическое состояние той же структуры после охлаждения (этот рисунок был скопирован с http://en.wikipedia.org/wiki/Simulated_annealing и распространяется в соответствии с положениями лицензии свободной документации GNU)

Симулированное закаливание основано на статистической механике, которая изучает поведение большого числа взаимодействующих частиц, таких как атомы в газе. Число атомов в любом материале составляет порядка 10^{23} на кубический сантиметр, поэтому когда мы исследуем свойства материала, то наблюдаем только те свойства, которые с высокой вероятностью произойдут. Мы также замечаем, что равновесно-энергетические конфигурации и похожие на них конфигурации наблюдаются очень часто, хотя эти конфигурации составляют лишь малую часть возможных конфигураций. Это происходит потому, что материалы, как правило, сходятся к состояниям с минимальной энергией; то есть природа является оптимизатором.

Предположим, что мы используем $E(s)$ для обозначения энергии определенной конфигурации s атомов в неком материале. Вероятность того, что система атомов находится в конфигурации s, выражается формулой

[1] Для справки: основными видами термической обработки в металлургии являются отжиг, нормализация и закалка. Отжиг состоит в нагреве металла, выдержке и последующем медленном охлаждении вместе с печью с соответствующей скоростью охлаждения. Отжиг приближает металл к равновесному состоянию. Нормализация – это частный вид отжига, отличающийся от отжига большей скоростью охлаждения. Закалка состоит в нагреве сплавов выше температур фазовых превращений и последующем быстром охлаждении, фиксирующем их высокотемпературное состояние. Назначение закалки – повысить твердость и прочность сталей, по сравнению с этими характеристиками после отжига, в 2–3 раза. См. любой учебник по материаловедению, либо http://www.comgun.ru/repair/539-vidy-i-rezhimy-termicheskoj-obrabotki-stalej.html. – *Прим. перев.*

$$P(s) = \frac{\exp\left[-E(s)/(kT)\right]}{\sum_w \exp\left[-E(w)/(kT)\right]}, \tag{9.1}$$

где k – это постоянная Больцмана, T – температура системы, находящейся в состоянии равновесия, и сумма в знаменателе берется по всем возможным конфигурациям w [Davis и Steenstrup, 1987]. Теперь предположим, что у нас есть система, которая находится в конфигурации q, и мы случайно отбираем конфигурацию r, являющуюся кандидатом конфигурации системы на следующем временном шаге. Если $E(r) < E(q)$, то мы принимаем r как конфигурацию на следующем временном шаге с вероятностью один:

$$P(r|q) = 1, \quad \text{если } E(r) < E(q). \tag{9.2}$$

То есть если наша кандидатная конфигурация r имеет энергию, которая меньше энергии s, мы на следующем временном шаге автоматически перейдем в r. Однако если $E(r) > E(q)$, то мы переходим в r на следующем временном шаге с вероятностью, пропорциональной относительной энергии q и r:

$$P(r|q) = \exp[(E(q) - E(r))/(kT)], \quad \text{если } E(r) \geq E(q). \tag{9.3}$$

То есть существует ненулевая вероятность P($r|q$), что система перейдет в конфигурацию с более высокой энергией. Если $E(r) > E(q)$, то уравнение (9.3) показывает, что вероятность P($r|q$), что система из состояния q перейдет в состояние r, меньше 1, но увеличивается вместе с увеличением T. Если использовать транзитные правила уравнений (9.2) и (9.3), тогда, по мере того как время стремится к бесконечности ($\rightarrow \infty$), вероятность того, что система находится в некоторой конфигурации s, сходится к больцмановскому распределению уравнения (9.1).

9.2. Простой алгоритм симулированного закаливания

Поскольку закаливание в природе приводит к низкоэнергетическим конфигурациям кристаллов, мы можем просимулировать его в алгоритме минимизации функций стоимости. Мы начинаем с кандидатного решения s для некой минимизационной задачи. Мы также начинаем с высокой «температуры», для того чтобы кандидатное решение с большой вероятностью изменилось в какую-то другую конфигурацию. Мы случайно генерируем альтернативное кандидатное решение r и измеряем его стоимость, аналогично энергии кристаллической структуры.

Если стоимость *r* меньше стоимости *s*, то мы соответствующим образом обновляем кандидатное решение, как указано в уравнении (9.2). Если стоимость *r* больше или равна стоимости *s*, то мы обновляем кандидатное решение с некоторой вероятностью, меньшей или равной единице, как указано в уравнении (9.3). Симулированное закаливание иногда называется больцмановским закаливанием из-за того, что в нем применяются уравнения (9.2) и (9.3). С течением времени (то есть по мере увеличения числа итераций) мы уменьшаем температуру. В результате приходим к тому, что кандидатное решение стремится локализоваться в низкостоимостном состоянии. Аналогии между закаливанием в природе и алгоритмом симулированного закаливания резюмируются следующим образом.

Закаливание в природе		**Симулированное закаливание**
атомарная конфигурация	↔	кандидатное решение
температура	↔	тенденция к разведыванию поискового пространства
охлаждение	↔	снижение тенденции к разведыванию
изменения в атомарных конфигурациях	↔	изменения в кандидатных решениях

Мы видим, что симулированное закаливание включает в себя ряд стандартных режимов работы эволюционного алгоритма. Несмотря на то что в этой главе мы использовали природное закаливание для обоснования симулированного закаливания, симулированное закаливание на самом деле может быть выведено без какого-либо обоснования со стороны природы [Michiels и соавт., 2007]. У обоих подходов есть свои преимущества. Обращение к аналогиям из природы может открыть новые возможности для исследований алгоритмов симулированного закаливания, но также может ограничить возможные расширения данного алгоритма. Простой алгоритм симулированного закаливания показан на рис. 9.2.

T = исходная температура > 0
$\alpha(T)$ = функция охлаждения: $\alpha(T) \in [0, T]$ для всех T
Инициализировать кандидатное решение x_0 для минимизационной задачи $f(x)$.
While not (критерий останова)
    Сгенерировать кандидатное решение x.
    If $f(x) < f(x_0)$
        $x_0 \leftarrow x$
    else
        $r \leftarrow U[0, 1]$

```
        If r < exp[(f(x0) – f(x))/T] then
            x0 ← x
        End if
    End if
    T ← α(T)
Next итерация
```


Рис.9.2. Базовый алгоритм симулированного закаливания для минимизации $f(x)$. Функция $U[0, 1]$ возвращает случайное число, равномерно распределенное на [0, 1]

На рис. 9.2 показано, что базовый алгоритм симулированного закаливания имеет общие черты с большинством других эволюционных алгоритмов. Во-первых, он прост и интуитивно привлекателен. Во-вторых, он основан на природном процессе оптимизации. В-третьих, имеет несколько регулировочных параметров, каждый из которых может оказать существенное влияние на его результативность.

- Исходная температура T обеспечивает верхнюю границу относительной важности разведывания, по сравнению с эксплуатацией. Если начальная температура слишком низкая, то алгоритм не будет эффективно разведывать поисковое пространство. Если начальная температура слишком высокая, то алгоритм будет слишком долго сходиться.
- Режим охлаждения $\alpha(T)$ управляет скоростью схождения. В начале алгоритма разведывание – высокое, а эксплуатация – низкая. В самом конце алгоритма верно обратное: эксплуатация – высокая и разведывание – низкое. Режим охлаждения управляет переходом от разведывания к эксплуатации. Если режим $\alpha(T)$ является слишком резким, то, как и кристаллическая структура, которая охлаждается слишком быстро, процесс закаливания будет сходиться к неупорядоченному (высокостоимостному) состоянию. Если режим $\alpha(T)$ слишком плавный, то процесс закаливания займет слишком много времени до схождения. Мы займемся обсуждением режимов охлаждения в разделе 9.3.
- Стратегия, используемая для создания кандидатного решения x на каждой итерации, может существенно повлиять на результативность симулированного закаливания. Случайная генерация x может сработать, но более интеллектуальный метод генерации x, который лучше, чем x_0, вероятно, даст более высокую результативность. Мы обсудим стратегии генерации кандидатов в разделе 9.4.1.

Упрощенный алгоритм симулированного закаливания может быть реализован путем замены критерия приемлемости на рис. 9.2 следующим образом:

Поменять «If $r < \exp[(f(x0) - f(x))/T]$» на «If $r < \exp[-c/T]$», (9.4)

где c называется *константой приемочной вероятности*. Она означает, что если стоимость решения x выше x_0, то вероятность замены x_0 на x не зависит от его стоимости. Константа приемочной вероятности c управляет противопоставлением разведывания и эксплуатации. Если c слишком велика, то алгоритм не будет разведывать поисковое пространство достаточно агрессивно. Если c слишком мала, то алгоритм будет разведывать слишком агрессивно, не эксплуатируя обнаруженные им ранее хорошие решения.

9.3. Режимы охлаждения

В этом разделе рассматриваются разные режимы охлаждения $\alpha(T)$, которые могут использоваться в алгоритме симулированного закаливания на рис. 9.2. Режим охлаждения может оказать значительное влияние на результативность симулированного закаливания. Если реализация симулированного закаливания не работает на некой задаче, то это может быть вызвано тем, что для данной задачи режим охлаждения не подходит. Некоторые часто применяемые режимы охлаждения включают линейное охлаждение, экспоненциальное охлаждение, обратное охлаждение, логарифмическое охлаждение и обратное линейное охлаждение, которые мы обсудим в следующих разделах. Мы также отметим, что оптимизационные задачи могут иметь разные шкалы вдоль разных размерностей, и поэтому в разделе 9.3.6 мы займемся обсуждением размерно-зависимого охлаждения.

9.3.1. *Линейное охлаждение*

Линейное охлаждение является простейшим типом охлаждения и следует режиму

$$\alpha(T) = T_0 - \eta k, \tag{9.5}$$

где T_0 – это исходная температура, k – число итераций симулированного закаливания и η – константа. Нам нужно гарантировать, чтобы $T > 0$ для всех k, поэтому мы должны выбрать η таким образом, чтобы температура при максимальном числе итераций была положительной. В качестве

альтернативы можно использовать следующую модифицированную форму линейного охлаждения:

$$\alpha(T) = \max(T_0 - \eta k, T_{\min}), \tag{9.6}$$

где $T_{\min}$ – задаваемая пользователем минимальная температура.

9.3.2. *Экспоненциальное охлаждение*

Экспоненциальное охлаждение следует режиму

$$\alpha(T) = \alpha T, \tag{9.7}$$

где обычно $\alpha \in (0.8,1)$. Более крупное значение α даст более медленный режим охлаждения. На рис. 9.3 показана нормализованная температура для режима экспоненциального охлаждения при различных значениях α. Мы видим, что α должно быть довольно близко к 1, иначе скорость охлаждения будет слишком резкой.

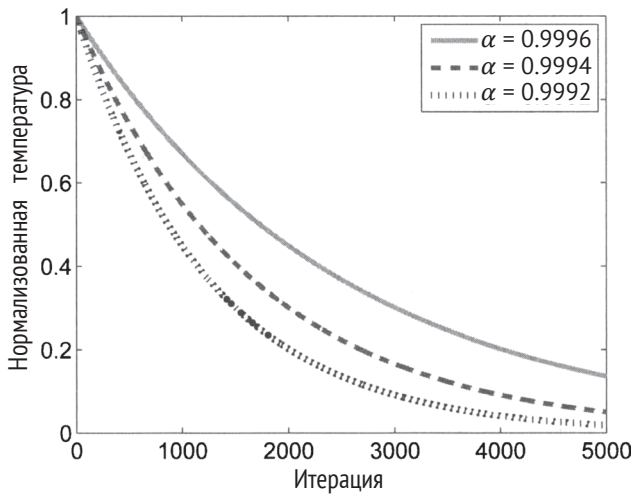


Рис. 9.3. Нормализованная температура как функция от α для режима экспоненциального охлаждения. В этом режиме охлаждения параметр α обычно очень близок к 1. Скорость охлаждения очень чувствительна к изменениям температуры

9.3.3. *Обратное охлаждение*

Обратное, или инвертированное, охлаждение следует графику

$$\alpha(T) = T/(1 + \beta T), \tag{9.8}$$

где β – это малая константа, обычно порядка 0.001. Меньшее значение β даст более медленный режим охлаждения. Данный режим охлаждения был впервые предложен в публикации [Lundy и Mees, 1986]. На рис. 9.4 показана нормализованная температура для режима обратного охлаждения при разных значениях β. Мы видим, что константа β должна быть совсем малой, иначе скорость охлаждения будет слишком резкой.

Сравнивая рис. 9.3 и 9.4, мы видим, что экспоненциальное охлаждение и обратное охлаждение можно сделать очень похожими друг на друга, подобрав соответствующие значения α и β.

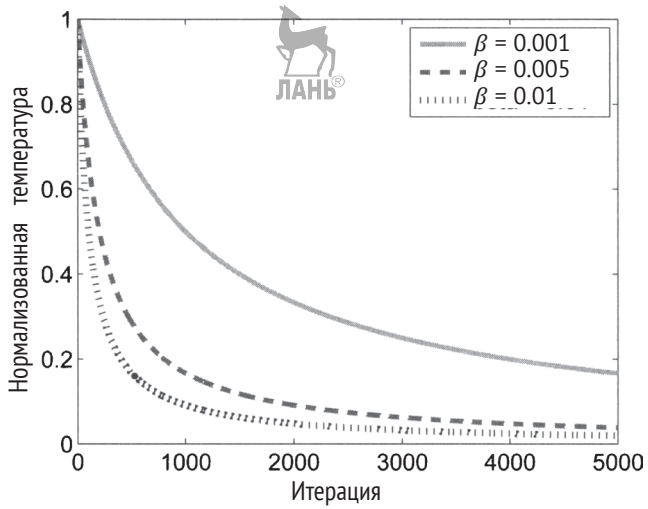


Рис. 9.4. Нормализованная температура как функция от β для режима обратного охлаждения. В этом режиме охлаждения параметр β обычно очень мал. Скорость охлаждения очень чувствительна к изменениям β

■ Пример 9.1

В данном примере мы оптимизируем 20-мерную функцию Экли, которая определена в приложении С.1.2, с помощью алгоритма симулированного закаливания рис. 9.2. Мы используем функцию обратного охлаждения, описанную в уравнении (9.8): $T_{k+1} = T_k/(1 + \beta T_k)$, где k – номер итерации, β – параметр режима охлаждения, T_k – температура на k-й итерации и $T_0 = 100$. Для генерации нового кандидатного решения на каждой итерации мы используем гауссово случайное число с центром в x_0:

$$x \leftarrow x_0 + N(0, T_k I), \tag{9.9}$$

где $N(0, T_k I)$ – гауссов случайный вектор со средним 0 и ковариацией $T_k I$ и I – единичная 20×20-матрица. Мы используем простую приемочную проверку из уравнения (9.4), где $c = 1$.

На рис. 9.5 показано лучшее решение, найденное как функция от номера итерации симулированного закаливания, усредненно по 20 симулированиям Монте-Карло, и для трех разных значений параметра β. Мы видим, что если параметр β слишком малый (0.0002), то охлаждение происходит очень медленно, и алгоритм симулированного закаливания слишком агрессивно скачет по поисковому пространству, не эксплуатируя хорошие решения, которые он уже получил. Если параметр β слишком большой (0.001), то охлаждение происходит очень быстро и алгоритм симулированного закаливания склонен застревать в локальных минимумах. Если параметр β подходит в самый раз (0.0005), то охлаждение происходит со скоростью, которая приводит к лучшему схождению. Однако мы отмечаем, что ближе к концу графика след $\beta = 0.0002$ быстро обгоняет след $\beta = 0.0005$. Это указывает на следующее. Несмотря на то что параметр $\beta = 0.0002$ слишком мал, для того чтобы давать хорошее схождение в пределах числа итераций, которые мы использовали, он в конечном итоге приведет к достаточному охлаждению, если число итераций продолжит увеличиваться, и в конечном итоге сойдется к хорошему результату.

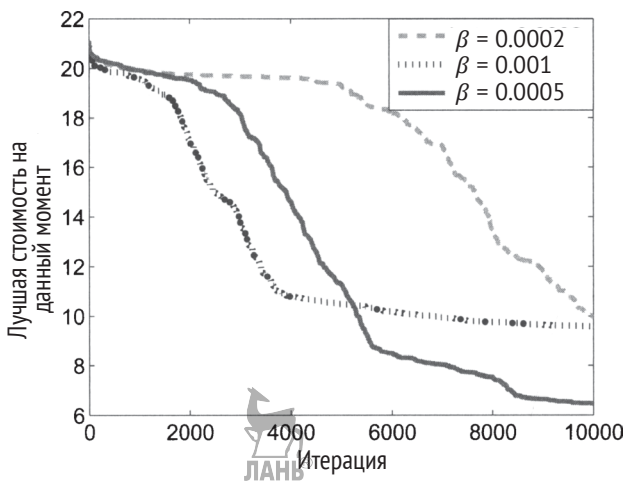


Рис. 9.5. Результаты примера 9.1 с симуляцией алгоритма симулированного закаливания для оптимизации 20-мерной функции Экли. Результаты усреднены по 20 симуляциям Монте-Карло. Параметр режима обратного охлаждения β оказывает значительное влияние на результативность симулированного охлаждения

■

Пример 9.1 показывает, что если охлаждение происходит слишком быстро или слишком медленно, то результативность симулированного закаливания, возможно, не будет хорошей. Такой же вывод можно сделать об исходной температуре. В примере 9.1 мы произвольно использовали $T_0 = 100$. К сожалению, хороших рекомендаций по выбору T_0 нет; все зависит исключительно от конкретной оптимизационной задачи.

9.3.4. *Логарифмическое охлаждение*

Логарифмическое охлаждение следует режиму

$$\alpha(T) = c / \ln k, \tag{9.10}$$

где c – это константа и k – число итераций симулированного закаливания. Впервые оно было предложено в публикации [Geman и Geman, 1984]. Иногда оно обобщается до

$$\alpha(T) = c / \ln(k + d), \tag{9.11}$$

где d – это константа, которая часто принимается равной 1 [Nourani и Andresen, 1998]. Как показано на рис. 9.6, логарифмическое охлаждение качественно отличается от экспоненциального и обратного охлаждения. Температура уменьшается очень быстро в течение первых нескольких итераций, а затем уменьшается чрезвычайно медленно. Это медленное уменьшение означает, что в режиме логарифмического охлаждения схождение симулированного закаливания обычно слабое. Поэтому режим логарифмического охлаждения не рекомендуется для практического применения.

Однако режим логарифмического охлаждения теоретически привлекателен и широко известен в сообществе исследователей симулированного закаливания, поскольку было доказано, что при определенных условиях он дает глобальный минимум [Geman и Geman, 1984]. В качестве простой демонстрации доказательства [Ingber, 1996] предположим, что имееся дискретная задача – такая, что размер поискового пространства конечен. Мы генерируем кандидата x из гауссова распределения так, что вероятность генерации x, при условии что x_k является текущим кандидатным решением на k-й итерации, равна

$$g_k \equiv P(x|x_k) = (2\pi\, T_k)^{D/2} \exp[-||x - x_k||_2^2/(2T_k)], \tag{9.12}$$

где D – это размер задачи. Другими словами, условная вероятность генерации x, при условии что x_k является текущим кандидатом, является гауссовой со средним x_k и ковариацией $T_k I$, где T_k – это температура на k-й итерации и I – единичная матрица. Чтобы посетить все кандидат-

ные решения в поисковом пространстве, достаточно показать, что по мере приближения числа итераций к бесконечности вероятность не посетить x приближается к нулю, то есть

$$\lim_{N\to\infty}\prod_{k=1}^{N}(1-g_k)=0. \tag{9.13}$$

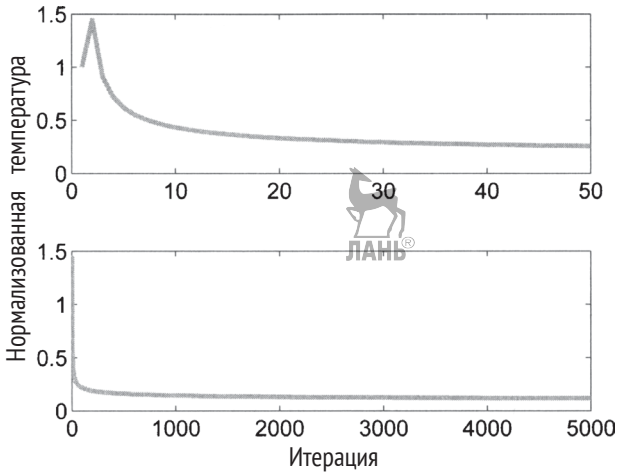


Рис. 9.6. Нормализованная температура для режима логарифмического охлаждения. На верхнем рисунке показана температура для первых 50 итераций; на нижнем – температура для первых 5000 итераций. В течение первых нескольких итераций температура уменьшается очень быстро, а затем уменьшается настолько медленно, что этот режим становится непрактичным для реализаций симулированного закаливания

Взятие натурального логарифма приведенного выше уравнения дает

$$\ln\left[\lim_{N\to\infty}\prod_{k=1}^{N}(1-g_k)\right]=\lim_{N\to\infty}\left[\ln\prod_{k=1}^{N}(1-g_k)\right]=-\infty. \tag{9.14}$$

Разложение в ряд Тейлора логарифма $g_1 = g_2 = \ldots = 0$ дает

$$\ln[(1-g_1)(1-g_2)\ldots]=\ln 1-g_1-g_2-\ldots. \tag{9.15}$$

Объединение двух предыдущих уравнений дает следующее достаточное условие для получения 100%-ной вероятности посещения x по мере приближения числа итераций к бесконечности:

$$\lim_{N\to\infty}\prod_{k=1}^{N}g_k=\infty. \tag{9.16}$$

Если g_k задано уравнением (9.12) и если $T_k = T_0/\ln k$, то левая часть приведенного выше уравнения становится

$$\lim_{N\to\infty}\sum_{k=1}^{N}(2\pi T_0/\ln k)^{D/2}\exp\left[-\|x-x_k\|_2^2/(2T_0/\ln k)\right] \geq \sum_{k=1}^{\infty}\exp(-\ln k) = \sum_{k=1}^{\infty}1/k = \infty, \tag{9.17}$$

где неравенство верно, если T_0 достаточно больше (см. задачу 9.5).

9.3.5. *Обратное линейное охлаждение*

Обратное линейное охлаждение следует режиму

$$\alpha(T) = T_0/k, \tag{9.18}$$

где T_0 – исходная температура, а k – номер итерации симулированного закаливания. Режим обратного линейного охлаждения проявляет быстрое охлаждение логарифмического графика в течение первых нескольких итераций, но при этом, как видно по рис. 9.7, он избегает ненулевых температур и медленного охлаждения более поздних итераций. Температура снижается очень быстро и быстро достигает нуля. Из этого следует, что обратное линейное охлаждение неэффективно для задач, требующих больших разведываний, но более подходит для задач, которые могут быть инициализированы кандидатным решением, о котором известно, что оно близко к оптимальному решению.

Обратное линейное охлаждение, как и логарифмическое, теоретически привлекательно и широко известно в сообществе исследователей симулированного закаливания, поскольку доказано, что при определенных условиях оно приводит к глобальному оптимуму [Szu и Hartley, 1987]. В качестве простой демонстрации, подобной той, которая использовалась в предыдущем разделе для логарифмического охлаждения [Ingber, 1996], предположим, что у нас есть дискретная задача – такая, что размер поискового пространства конечен. Мы генерируем кандидата x из распределения Коши так, что вероятность получения x, при условии что x_k является текущим кандидатным решением на k-й итерации, равна

$$g_k \equiv P(x \mid x_k) = \frac{T_k}{(\| x - x_k \|_2^2 + T_k^2)^{(D+1)/2}}, \tag{9.19}$$

где D – это размер задачи. Обратите внимание, что в предыдущем разделе для генерации кандидатных решений мы использовали распреде-

ление Гаусса, в то время как в этом разделе используем распределение Коши. На рис. 9.8 дается сравнение функции плотности вероятности (PDF) распределения Коши с функцией плотности вероятности распределения Гаусса. Мы видим, что плотность вероятности распределения Коши имеет гораздо более толстые хвосты, а это означает, что мы с большей вероятностью создадим кандидатные решения, которые находятся дальше от текущего кандидатного решения.

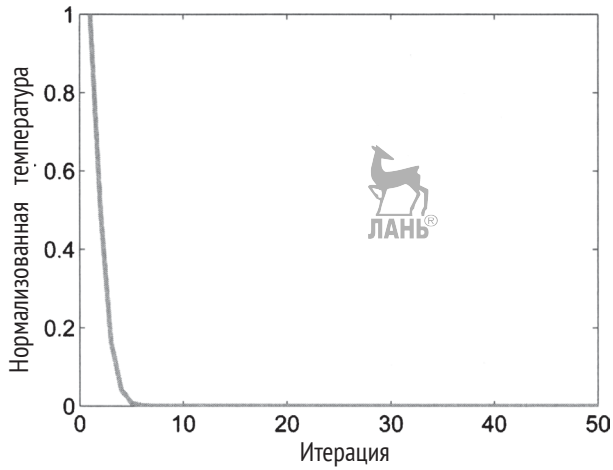


Рис. 9.7. Нормализованная температура для режима обратного линейного охлаждения. Температура достигает нуля очень быстро

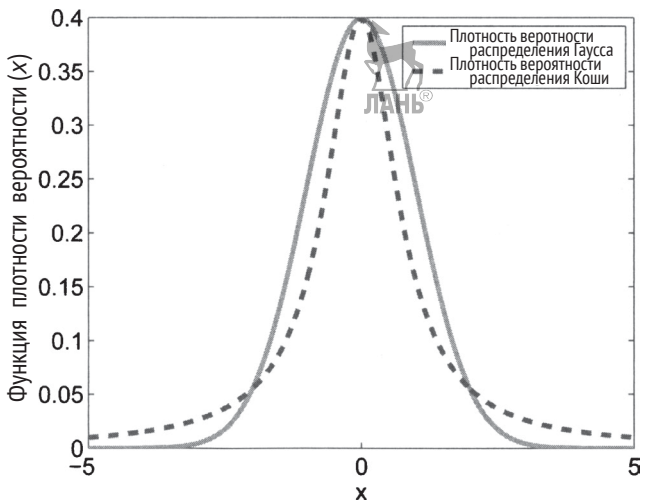


Рис. 9.8. Сравнение одномерных функций плотности вероятности распределения Коши и распределения Гаусса

Если g_k задано уравнением (9.19) и если $T_k = T_0/k$, то левая часть уравнения (9.16) становится

$$\lim_{N\to\infty}\sum_{k=1}^{N}\frac{T_0/k}{(\|x-x_k\|_2^2+T_0^2/k^2)^{(D+1)/2}} \geq \sum_{k=1}^{\infty}1/k=\infty, \tag{9.20}$$

где неравенство верно для соответствующих значений T_0 (см. задачу 9.6).

Теперь мы сравним результаты схождения, полученные с помощью режимов логарифмического и обратного линейного охлаждения. Напомним, что плотность вероятности распределения Коши имеет гораздо более толстые хвосты, чем гауссова плотность вероятности. Также напомним, что функция генерации кандидатного решения уравнения (9.19), в которой используется плотность вероятности распределения Коши, совмещена с режимом обратного линейного охлаждения рис. 9.7. Наконец, напомним, что функция генерации кандидатного решения уравнения (9.12), в которой используется гауссова плотность вероятности, использует режим логарифмического охлаждения рис. 9.6. Поскольку функция генерации распределения Коши имеет более толстые хвосты, чем функция генерации распределения Гаусса, она гарантированно сходится с гораздо более быстрым режимом охлаждения, чем тот, который используется в сочетании с функцией генерации распределения Гаусса.

9.3.6. *Размерно-зависимое охлаждение*

В реальных приложениях и даже в отдельных эталонных задачах ландшафт функции стоимости может выглядеть очень по-разному, если смотреть вдоль разных размерностей. Например, рассмотрим функцию

$$f(x) = 20 + e - 20\exp\left(-0.2\sum_{i=1}^{n}y_i^2/n\right) - \exp\left(\sum_{i=1}^{n}(\cos 2\pi y_i)/n\right)$$

$$y_i = \begin{cases} x_i & \text{для нечетного } i \\ x_i/4 & \text{для четного } i \end{cases} \tag{9.21}$$

Данная функция является шкалированной версией функции Экли, которая определена в приложении С.1.2. Тот факт, что x_i прошкалировано для четных значений i, означает, что функция «растягивается» вдоль этих размерностей. На рис. 9.9 показан двумерный график этой функции. Из-за шкалирования четных размерностей эта функция намного более сглажена вдоль размерности x_2, чем вдоль размерности x_1.

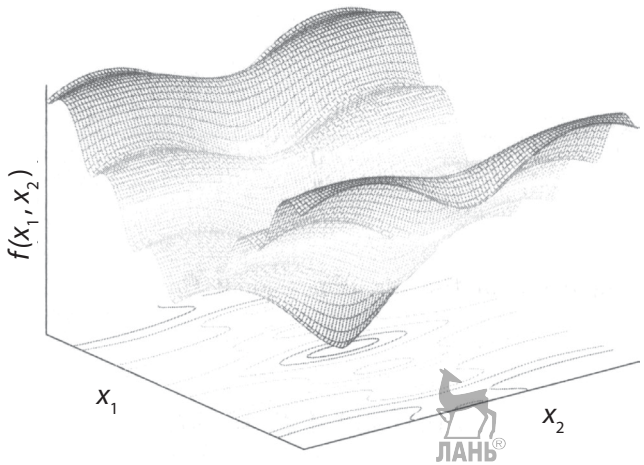


Рис. 9.9. Шкалированная версия двумерной функции Экли. Топология намного более гладкая вдоль направления x_2, что указывает на то, что алгоритм симулированного закаливания должен использовать более медленный режим охлаждения вдоль этой размерности

Для функций с разными топологиями в разных размерностях мы, возможно, захотим использовать разный режим охлаждения для разных размерностей. Для шкалированной функции Экли уравнения (9.21) мы бы использовали более медленное охлаждение вдоль четных размерностей и более быстрое охлаждение вдоль нечетных размерностей. Это позволит алгоритму симулированного закаливания постепенно сходиться к оптимальным значениям вдоль постепенно меняющихся размерностей функции. Быстрый режим охлаждения вдоль постепенно меняющихся размерностей не позволит алгоритму симулированного закаливания двигаться вниз по пологим склонам. Однако вдоль более динамических размерностей функции необходим более быстрый режим охлаждения. Алгоритм симулированного закаливания будет двигаться вниз по склону вдоль динамических размерностей даже с быстрой скоростью охлаждения, но медленная скорость охлаждения приведет к слишком большому числу скачков вокруг. Можно взглянуть на это и несколько по-иному. Мы можем сказать, что нам нужен более агрессивный поиск (высокие температуры) вдоль размерностей с низкой чувствительностью и менее агрессивный поиск (низкие температуры) вдоль размерностей с высокой чувствительностью.

Если мы используем режим обратного охлаждения уравнения (9.8), то приведенное выше обсуждение подразумевает меньшее значение β для четных размерностей и большее значение β для нечетных размер-

ностей. Из этого следует, что каждая размерность задачи будет иметь свою температуру. В результате приходим к модификации базового алгоритма симулированного закаливания на рис. 9.2 и получаем размерно-зависимый алгоритм симулированного закаливания на рис. 9.10.

T_i = исходная температура > 0, $i \in [1, n]$
$\alpha_i(T_i)$ = функция охлаждения для i-й размерности, $i \in [1, n]$: $\alpha(T_i) \in [0, T_i]$ для всех T_i
Инициализировать кандидатное решение x_0 для минимизационной задачи $f(x)$:
    $x_0 = [x_{01}, x_{02}, ..., x_{0n}]$
While not (критерий останова)
    Сгенерировать кандидатное решение $x_1 = [x_{11}, x_{12}, ..., x_{1n}]$
    If $f(x_i) < f(x_0)$
        $x_0 \leftarrow x_1$
    else
      For $i = 1, ..., n$
          $r \leftarrow U[0, 1]]$
          If $r < \exp[(f(x_0) - f(x_1))/T]$ **then**
            $x_{0i} \leftarrow x_{1i}$
          End if
      Next размерность i
    End if
    $T_i \leftarrow \alpha_i(T_i), i \in [1, n]$
Next итерация

Рис. 9.10. Алгоритм симулированного закаливания с размерно-зависимым охлаждением для минимизации n-размерной функции $f(x)$. Функция $U[0, 1]$ возвращает случайное число, равномерно распределенное на [0, 1]. Данный алгоритм является обобщением базового алгоритма симулированного закаливания на рис. 9.2; здесь мы позволили каждой размерности иметь свою собственную температуру и свой собственный режим охлаждения

■ Пример 9.2

В данном примере мы оптимизируем 20-мерную шкалированную функцию Экли, задаваемую уравнением (9.21), с помощью алгоритма размерно-зависимого симулированного закаливания на рис. 9.10. Мы применяем функцию обратного охлаждения, описанную в уравнении (9.8) для каждой размерности: $T_{k+1,i} = T_{ki}/(1 + \beta_i T_{ki})$, где i – это номер размерности, k – номер итерации симулированного закаливания, β_i – параметр режима охлаждения для i-й размерности, T_{ki} – температура на k-й итерации i-й размерности, $T_{ki} = 100$ для всех i. Для генерирования нового кандидатного решения на каждой итерации мы используем гауссово случайное число с центром в x_0:

$$x_{1i} \leftarrow x_{0i} + N(0, T_{ki}), \tag{9.22}$$

где $N(0, T_{ki})$ – это гауссово случайное число со средним 0 и дисперсией T_{ki}. Мы используем простую приемочную проверку из уравнения (9.4), где $c = 1$.

На рис. 9.11 показано лучшее решение, найденное как функция от номера итераций симулированного закаливания, усредненно по 20 симуляциям Монте-Карло, и для четырех разных комбинаций β. Мы видим, что если параметр β слишком низок (0.001), то охлаждение происходит очень медленно, и алгоритм симулированного закаливания скачет по поисковому пространству слишком агрессивно, не эксплуатируя хорошие решения, которые он уже получил. Если параметр β слишком высок (0.005), то охлаждение происходит очень быстро, и алгоритм симулированного закаливания имеет тенденцию застревать в локальных минимумах. Однако если параметр β большой для нечетных размерностей и малый для четных размерностей, то охлаждение происходит со скоростью, которая приводит к лучшему схождению. Такая комбинация дает быстрое охлаждение для высокодинамических нечетных размерностей и медленное охлаждение для четных размерностей.

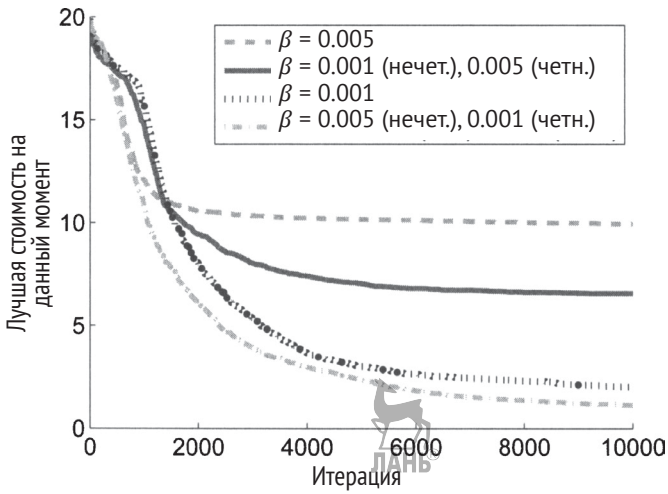


Рис. 9.11. Результаты примера 9.2 с симуляцией алгоритма размерно-зависимого симулированного закаливания, оптимизирующего 20-мерную шкалированную функцию Экли. Результаты усреднены по 20 симуляциям Монте-Карло. Параметры режима охлаждения $\{\beta_i\}$ могут регулироваться индивидуально для каждой размерности с целью получения лучших результатов

■

9.4. Вопросы реализации

В этом разделе рассматривается несколько вопросов реализации, в том числе такие как генерирование кандидатных решений, время реинициализации температуры охлаждения и зачем необходимо отслеживать лучшее кандидатное решение.

9.4.1. *Генерирование кандидатного решения*

Инструкция «Сгенерировать кандидатное решение» выглядит в алгоритмах симулированного закаливания на рис. 9.2 и 9.10 обманчиво простой. Существует много разных способов реализации этой инструкции, и выбор ее реализации может оказать большое влияние на результативность симулированного закаливания. Один из способов генерирования кандидатных решений – просто выбирать случайную точку в поисковом пространстве. Вместе с тем, после того как алгоритм симулированного закаливания начнет сходиться к хорошему решению, мы ожидаем, что текущее кандидатное решение x_0 будет намного лучше, чем большинство других точек в поисковом пространстве. Поэтому генерирование случайного кандидатного решения, вероятно, будет не очень эффективной мерой. Как правило, мы должны смещать генерирование кандидатных решений в сторону текущего кандидатного решения x_0. Именно это является причиной того, что для генерирования кандидатного решения в уравнениях (9.9) и (9.22) используется гауссова случайная величина с центром в x_0. Кроме того, дисперсия гауссовой случайной величины равна температуре, которая уменьшается со временем, и поэтому поиск имеет тенденцию сужаться по мере увеличения числа итераций алгоритма симулированного закаливания. Уравнение (9.19), распределение Коши, может быть использовано в качестве более агрессивного метода генерирования кандидатных решений, при этом по-прежнему оставаясь центрированным в x_0. Смещение генерирования кандидатных решений в сторону x_0 приводит к исключению не только довольно слабых, но и очень хороших кандидатов. Однако достаточно слабые точки в поисковом пространстве обычно встречаются чаще, чем очень хорошие кандидаты, поэтому смещение поиска в сторону x_0, как правило, эффективно.

9.4.2. *Реинициализация*

Как мы обсуждали ранее в этой главе, режим охлаждения является важным фактором в эффективности симулированного закаливания.

Если охлаждать температуру слишком быстро, то симулированное закаливание застрянет в локальном оптимуме, и результативность будет слабой. Вместе с тем мы обычно не знаем заранее, каким является подходящий режим охлаждения. Поэтому мы часто следим за улучшением алгоритма симулированного закаливания, и если мы не находим лучшего решения в течение L итераций, то реинициализируем температуру значением T_0, чтобы продлить процесс разведывания.

9.4.3. *Отслеживание лучшего кандидатного решения*

Напомним по рис. 9.2, что новое кандидатное решение x могло бы заменить текущее кандидатное решение x_0, даже если x хуже x_0. Этот риск оправдан необходимостью достаточного разведывания поискового пространства, но это, возможно, приведет к потере хорошего кандидатного решения. Поэтому в алгоритме симулированного закаливания обычно возникает потребность в организации архива, для того чтобы отслеживать лучшее кандидатное решение, полученное к настоящему моменту. Этот подход похож на элитарность из раздела 8.4; однако в том разделе мы фактически оставляли лучшие кандидатные решения в популяции. Мы не можем сделать это непосредственно в алгоритме симулированного закаливания, если не увеличим размер популяции сверх 1, что является возможностью, которую мы не обсуждали в этой главе. Однако независимо от размера популяции мы всегда можем вести архив, который будет содержать найденного к настоящему моменту лучшего кандидата. Поэтому даже если мы заменим хорошее кандидатное решение на слабого кандидата из-за разведывательного характера алгоритма симулированного закаливания, мы все равно будем отслеживать лучшего кандидата, найденного к настоящему моменту. Лучшее кандидатное решение в архиве никогда не будет заменено на худшего кандидата. И тогда мы сможем вернуть лучшего найденного кандидата после завершения алгоритма симулированного закаливания.

9.5. Заключение

Симулированное закаливание является одним из старейших эволюционных алгоритмов, который берет свое начало в 1983 году, однако мы обсуждали его в этой части книги, потому что он не всегда считается классическим эволюционным алгоритмом. Он не популяционно-ориентированный, но общеизвестно, что некоторые из классических эволюционных алгоритмов тоже не являются таковыми, и поэтому данный факт не выступает достаточной причиной для удаления его из катего-

рии эволюционных алгоритмов. Поскольку алгоритм симулированного закаливания основан на естественном процессе и является итеративным оптимизационным алгоритмом, мы обычно рассматриваем его как эволюционный алгоритм. Его зрелость и научные корни привели ко многим работам, книгам и приложениям. Читателям, которые хотят получить более полный и всеохватывающий материал по симулированному закаливанию, рекомендуется обратиться к книгам [van Laarhoven и Aarts, 2010], [Otten и van Ginneken, 1989] и [Aarts и Korst, 1989]. Учебно-практические главы имеются в книгах [Aarts и соавт., 2003] и [Henderson и соавт., 2003].

Как и все эволюционные алгоритмы, рассмотренные в этой книге, алгоритм симулированного закаливания может быть полезен для решения широкого круга оптимизационных задач, включая как задачи с непрерывной, так и задачи с дискретной областью. Современные направления исследований алгоритмов симулированного закаливания отражают текущие акценты в общих эволюционно-алгоритмических исследованиях: симулированное закаливание для многокритериальных задач [Bandyopadhyay и соавт., 2008], гибридизация симулированного закаливания с другими эволюционными алгоритмами [Cakir и соавт., 2011], распараллеливание [Zimmerman и Lynch, 2009] и ограниченная оптимизация [Singh и соавт. 2010].

В этой главе представлена предыстория и реализация алгоритма симулированного закаливания, но существуют и другие важные аспекты данного алгоритма, которые мы не успели обсудить. Например, марковская модель и некоторые теоретические доказательства сходимости представлены в публикации [Michiels и соавт. 2007], хотя по-прежнему остается ряд возможностей для дополнительного моделирования и теоретических результатов. Практикующие специалисты заинтересованы не только в схождении, но и в результативности на конечных временных интервалах, и этот вопрос обсуждается в публикациях [Henderson и соавт., 2003], [Vorwerk и соавт., 2009].

Задачи

Письменные упражнения

9.1. Каковы единицы измерения температуры в уравнении (9.1)?

9.2. Постройте качественно правильный график вероятности $P(r|q)$ из раздела 9.1 как функции от $\Delta E = E(r) - E(q)$.

9.3. Какое значение константы приемочной вероятности c даст вероятность принятия p, если новое кандидатное решение x имеет более высокую стоимость, чем текущее кандидатное решение x_0?

9.4. Предположим, вы хотите выполнить алгоритм симулированного закаливания для 10 000 итераций с линейным охлаждением. Какое значение p следует использовать, чтобы температура достигла 0 на последней итерации?

9.5. Что такое «достаточно большой» в доказательстве сходимости режима логарифмического охлаждения?

9.6. Каковы «подходящие значения T_0» в доказательстве сходимости режима обратного линейного охлаждения?

9.7. Напишите режимы линейного и обратного линейного охлаждения в виде $T_{k+1} = \alpha(k, T_k)$, где k – это номер итерации алгоритма симулированного закаливания.

9.8. В данной задаче сравнивается экспоненциальное и обратное охлаждение.

   а) Напишите режимы экспоненциального и обратного охлаждений в виде $T_k = f(k, T_0)$, где k – это номер итерации алгоритма симулированного закаливания и T_0 – исходная температура.

   б) Найдите выражение для α в режиме экспоненциального охлаждения, так чтобы температура после N итераций была такой же, как в режиме обратного охлаждения.

   с) При наличии $T_0 = 100$ какое значение α дает эквивалентную температуру после 10 000 итераций, когда:

      1) $\beta = 0.01$; 2) $\beta = 0.001$; 3) $\beta = 0.0001$?

Компьютерные упражнения

9.9. В данной задаче исследуется стратегия реинициализации, рассмотренная в разделе 9.4.2. Реализуйте симулированное закаливание для минимизации 20-мерной функции Экли. Используйте следующие параметры:

- обратное охлаждение с $\beta = 0.001$;
- исходная температура = 100;
- допустимое число итераций = 10 000;
- приемочная проверка с уравнением (9.4), где $c = 1$;

- генерирование кандитатного решения с использованием $x \leftarrow x_0 + r$, где r – это нормально распределенное случайное число с нулевым средним и дисперсией T^2.

Отследите лучшее к настоящему моменту решение x_k^* как функцию от номера итерации k и постройте график среднего значения x_k^* по 20 симуляциям Монте-Карло. Сравните графики для следующих ниже стратегий реинициализации:

- реинициализировать T всякий раз, когда x хуже x_0 в течение 10 итераций подряд;
- реинициализировать T всякий раз, когда x хуже x_0 в течение 100 итераций подряд;
- реинициализировать T всякий раз, когда x хуже x_0 в течение 1000 итераций подряд;
- ни разу не реинициализировать T.

Какой вывод вы делаете из своих результатов о значении реинициализации T?

9.10. В данной задаче рассматриваются методы, используемые для генерации кандидатных решений. Реализуйте алгоритм симулированного закаливания для минимизации 20-мерной функции Экли. Примените те же самые параметры, что и в задаче 9.9. Отследите лучшее к настоящему моменту решение x_k^* как функцию от номера итерации k и постройте график среднего значения x_k^* по 20 симуляциям Монте-Карло. Сравните графики для следующих ниже стратегий создания кандитатных решений:

- $x \leftarrow x_0 + r_1$, где r_1 – это нормально распределенное случайное число с нулевым средним и дисперсией T^2;
- $x \leftarrow r_2$, где r_2 – это случайное число, равномерно распределенное по поисковой области.

Какой вывод вы делаете из своих результатов о важности генерирования кандидатных решений?

Глава 10

Оптимизация на основе муравьиной кучи

Пойди к муравью, ленивец, посмотри на действия его и стань мудрым!

– Притчи 6:6

Муравьи – простые существа, но они могут делать многое, работая сообща. Цитата в начале этой главы представляет муравьев как парадигму упорного труда, но они также могут быть изображены как воплощение бескорыстного сотрудничества. Один муравей ничего не может предложить. Одинокий муравей, возможно, будет бесцельно бродить по кругу до тех пор, пока не умрет от истощения [Delsuc, 2003]. В мозгу среднестатистического муравья только 10 000 нейронов, чего, похоже, совсем недостаточно, чтобы достичь многого. Но муравьи объединяются в муравьиные кучи, которые могут насчитывать миллионы. Один миллион членов муравьиной кучи насчитывает в общей сложности 10 млрд нейронов, что начинает вступать в соперничество с числом нейронов у среднестатистического человека. Муравьи, по-видимому, действуют как единое целое и поэтому иногда называются суперорганизмом [Hölldobler и Wilson, 2008]. Муравьиная суперкуча, обнаруженная на японском острове Хоккайдо, по сообщениям, в 1979 году содержала более 300 млн муравьев, живущих в 45 000 взаимосвязанных муравьиных гнезд [Hölldobler и Wilson, 1990, стр. 1]. Муравьи процветают почти в любой среде на Земле и, по оценкам, составляют более 15 % от массы всех наземных животных [Schultz, 2000]. Мирмекологи (то есть те, кто изучает муравьев) говорят нам, что число видов муравьев составляет

примерно 8800 и они имеют глобальную популяцию, насчитывающую около одного квадриллиона, а это означает, что на каждого человека на Земле приходится где-то 150 000 муравьев. Почему получилось так, что муравьи, эти крошечные существа, остаются такими успешными и так долго в столь разных средах? Ученые связывают их приспособляемость и доминирование с их социальной организацией. Возможно, не случайно, что самое доминирующее млекопитающее на Земле, человек, тоже является явным лидером в социальной организации.

Муравьи общаются главным образом с помощью феромонов, то есть химических веществ, которые они выделяют. Когда муравьи путешествуют по тропинке к источнику пищи и приносят пищу обратно в свою кучу, они оставляют след феромона. Другие муравьи улавливают запах феромонов своими антеннами, следуют по этой тропинке и приносят еще больше пищи в кучу. В этом процессе муравьи продолжают наносить феромон, который другие муравьи продолжают улавливать, и тропинка к источнику пищи усиливается. С течением времени кратчайший путь к пище, таким образом, становится более привлекательным, поскольку он усиливается положительной обратной связью.

Иногда источник пищи истощается, или препятствие преграждает перемещение к источнику пищи. Когда муравьи путешествуют по тропинке и не находят пищи, они начинают бродить вокруг до тех пор, пока не найдут пищу. Если они не возвращаются в свою кучу, используя первоначальный путь, то они не наносят никакого дополнительного феромона на этом пути. По прошествии времени феромон на исходном пути испаряется, меньше число муравьев следует исходному пути, больше муравьев следует по новой тропинке к новому источнику пищи, и муравьи открывают новый оптимальный путь. Этот общий процесс показан на рис. 10.1.

Муравьи могут не только находить оптимальный путь к источнику пищи, но и выполнять многие другие впечатляющие задачи, работая сообща. Они могут строить сложные сети туннелей под землей либо, в случае муравьев-ткачей, на деревьях. Их колонии имеют специализированные помещения для хранения, спаривания и ухода за личинками.

Они могут сажать сады для выращивания собственного источника пищи [Schultz, 1999], могут образовывать цепи для пересечения брешей над землей или над водой (см. рис. 10.2), могут образовывать плоты, для того чтобы пережить наводнение или путешествовать по воде.

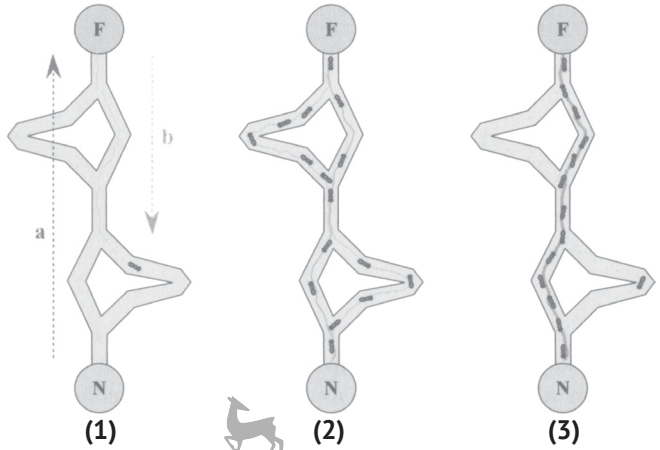


Рис. 10.1. Муравьи, наносящие феромон и следующие за феромоном: (1) первый муравей путешествует в направлении, указанном *a*, находит источник пищи *F* и возвращается в муравейник *N* в направлении, обозначенном *b*, прокладывая феромонный след на своем пути; (2) муравьи следуют по одному из четырех возможных путей из *N* в *F*, но усиление феромонами делает кратчайший путь привлекательнее; (3) муравьи, как правило, следуют по пути с наибольшим количеством феромонов, продолжая укреплять его целесообразность, в то время как феромон на более длинных путях испаряется (этот рисунок был создан Иоганном Дрео (Johann Dréo), скопирован с http://en.wikipedia.Org/wiki/File:Aco_branches.svg и распространяется в соответствии с положениями лицензии свободной документации GNU)

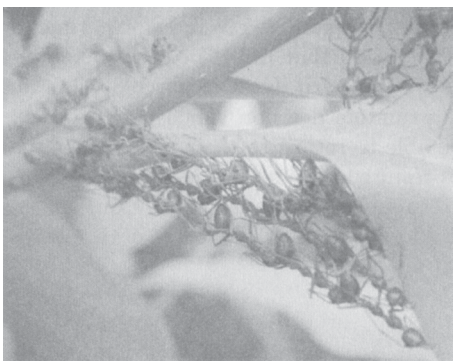


Рис. 10.2. Муравьи образуют мост между листьями. Муравьи используют мосты не только для транспортировки, но и для стягивания листьев при строительстве гнезд. Многие другие подобные фотографии можно найти в работе [Hölldobler и Wilson, 1994] (эта фотография была сделана Шоном Хойландом (Sean Hoyland), скопирована с http://en.wikipedia.Org/wiki/File:SSL11903p.jpg и распространяется в соответствии с положениями лицензии свободной документации GNU)

Краткий обзор главы

В этой главе мы обсудим оптимизацию на основе муравьиной кучи (ant colony optimization, ACO), которая представляет собой алгоритм, обусловленный феромонным поведением муравьев. Большинство исследователей алгоритма оптимизации на основе муравьиной кучи подчеркивает, что это не эволюционный алгоритм, так как кандидатные решения непосредственно не обмениваются информацией о решениях друг с другом. Мы включили оптимизацию на основе муравьиной кучи в эту книгу не потому, что хотим поучаствовать в дебатах о том, относится данный алгоритм к эволюционным или нет, а просто потому, что алгоритм оптимизации на основе муравьиной кучи – это интересный и эффективный биологически обусловленный популяционно-ориентированный алгоритм оптимизации.

Алгоритм оптимизации на основе муравьиной кучи был описан Мартином Дориго (Martin Dorigo) в его докторской диссертации и впервые опубликован в 1991 году [Colorni и соавт., 1991]. В разделе 10.1 мы обсуждаем отложение феромонов биологическими муравьями, их испарение и математические модели, описывающие эти процессы. В разделе 10.2 анализируем муравьиную систему, которая была предложена в середине 1990-х годов и которая была первым алгоритмом оптимизации на основе муравьиной кучи. Первоначально предлагалось, что данный алгоритм будет находить оптимальные пути, но был быстро модифицирован для решения оптимизационных задач с непрерывными областями, и это то, что мы обсуждаем в разделе 10.3. В разделе 10.4 мы обсудим некоторые популярные модификации, которые были внесены в основной алгоритм муравьиной системы, включая минимаксную муравьиную систему (MMAS) и систему муравьиной кучи (ACS). В разделе 10.5 дается краткий обзор исследований по оптимизации на основе муравьиной кучи в области теории и моделирования.

10.1. Модели феромона

Предположим, что мы наблюдаем за муравьиным гнездом и источником пищи и видим, что муравьи имеют два возможных маршрута для получения пищи. Один маршрут длинный, а другой короткий, как показано на рис. 10.3. Госс (Goss) и его коллеги провели ряд экспериментов такого типа с аргентинским муравьем и обнаружили, что в 95 % своих экспериментов более 80 % муравьиного трафика было на более коротком пути [Goss и соавт., 1989]. Когда муравьи достигали развилки пути,

они принимали случайное решение, какой путь выбрать. Муравьи, выбравшие более короткий путь, могли возвращаться в свое гнездо раньше, чем муравьи, выбравшие более длинный путь. Это привело к тому, что за единицу времени больше муравьев принимало более короткий путь. Это, в свою очередь, привело к нанесению большего количества феромонов на более коротком пути. Наконец, большее количество феромонов на более коротком пути побуждало муравьев идти по этому пути.

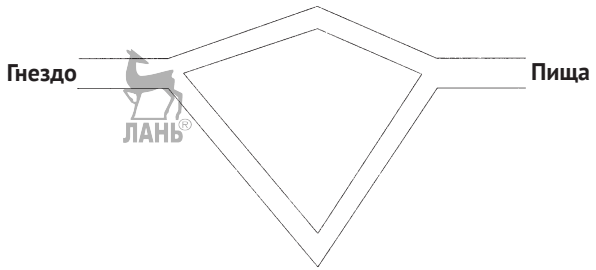


Рис. 10.3. Экспериментальная конструкция для изучения того, как муравьи находят кратчайшее расстояние до пищи. В 95 % экспериментов более 80 % трафика муравьев было на более коротком пути. Взято из публикации [Goss и соавт., 1989]

Мы видим, что муравьиные путешествия – это феномен с положительной обратной связью, по крайней мере до определенного момента. Всегда были некоторые муравьи, которые выбирали более длинный путь, потому что их выбор частично регулируется случайными процессами. Однако в целом чем больше муравьев выбирают более короткий путь, тем больше феромонов получает короткий путь, и чем больше феромонов получает короткий путь, тем больше муравьев его выбирает.

Этот феномен положительной обратной связи также характерен для эволюционных алгоритмов. Например, в генетических алгоритмах особи в первом поколении с полезными генетическими признаками будут с большей вероятностью отобраны для рекомбинации. Из этого следует, что второе поколение, скорее всего, будет обладать этими генетическими признаками. Более широкое распространение этих полезных признаков во втором поколении повышает вероятность того, что они будут переданы третьему поколению. Данный феномен положительной обратной связи обнаруживается не только в оптимизации на основе муравьиной кучи и генетическом алгоритме, но во всех эволюционных алгоритмах.

Феромоны не только наносятся, но они и испаряются. Госс провел еще один эксперимент, в котором муравьиной куче был доступен только один путь. Муравьи бегали взад и вперед между своим гнездом и источником пищи по этому пути, потому что у них не было других вариантов. Через некоторое время Госс добавил короткий путь к пище, как показано на рис. 10.4. У муравьев теперь был выбор, но все феромоны были на длинном пути. Однако когда муравьи достигали развилки пути, они не шли по насыщенному феромонами пути автоматически. Они с большей вероятностью идут по пути с большим количеством феромонов, но в их поведении также существует случайный элемент. Поэтому некоторые муравьи пошли по новоявленному короткому пути.

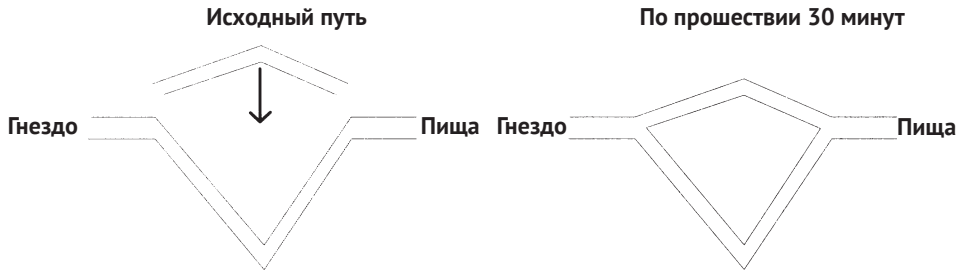


Рис. 10.4. Экспериментальная конструкция для изучения реакции муравьев при добавлении более короткого пути к пище. В 20 % экспериментов более 50 % муравьиного трафика сходилось к более короткому пути. Взято из публикации [Goss и соавт., 1989]

Пройдя этот короткий путь, они отложили на нем феромоны, что сделало его более привлекательным для последующих муравьев-путешественников. Примерно в 20 % экспериментов большинство муравьев в конечном итоге шло по короткому пути, хотя у него не было никаких исходных отложений феромона. Это свидетельствует о том, что феромоны испаряются. Однако в данном эксперименте они не испарялись достаточно быстро для того, чтобы муравьи чаще шли по более короткому пути, чем по более длинному.

С учетом этих и других подобных экспериментов Денебург и его коллеги предложили математическую модель нанесения и испарения феромонов [Deneubourg и соавт., 1990]. При наличии двух путей на выбор вероятность того, что муравей выберет путь 1, определяется формулой

$$p_1 = \frac{(m_1 + k)^h}{(m_1 + k)^h + (m_2 + k)^h}, \tag{10.1}$$

где m_i – это число муравьев, которые ранее выбрали путь i, а h и k – экспериментально определенные параметры. Эта исходная модель не учитывает испарения феромонов. Типичные значения для k и h составляют

$$k \approx 20, \quad h \approx 2. \tag{10.2}$$

На рис. 10.5 показаны результаты симуляции уравнения (10.1), которое применяется к двум путям одинаковой длины. Верхний график показывает, что поведение первых 100 муравьев не предсказуемо. Поведение муравьев в основном случайное, и есть 50%-ная вероятность, что муравей выберет путь 1 либо 2. Нижний график показывает, что, по мере того как один путь начинает получать большинство отложений феромонов, он становится более привлекательным, что приводит к феномену положительной обратной связи, который мы обсуждали ранее в этом разделе. В итоге 100 % трафика будет проходить только по одному из двух путей.

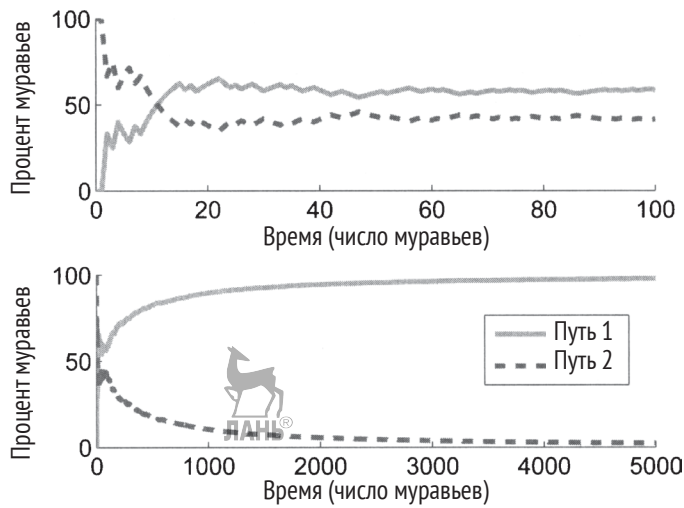


Рис. 10.5. Результаты симулирования уравнения (10.1). Первоначально муравьи имеют около 50 % шансов выбрать любой из этих путей. Через некоторое время один путь начинает преобладать по количеству феромонов, что приводит к положительной обратной связи, и 100 % муравьиного трафика формируется на одном из двух путей

Мы видим, что муравьи способны находить оптимальные решения задач, с которыми они сталкиваются в своей повседневной жизни. Это мотивирует нас симулировать искусственных муравьев для поиска оптимальных решений инженерных задач.

10.2. Муравьиная система

Муравьиная система (ant system) была первым алгоритмом оптимизации на основе муравьиной кучи, который был опубликован в изданиях [Colorni и соавт., 1991], [Dorigo и соавт., 1996]. Ее можно проиллюстрировать на задаче коммивояжера (см. раздел 2.5 и главу 18). Каждый муравей в симуляции оптимизиции на основе муравьиной кучи путешествует из одного города в другой, и симуляция наносит феромон на путях муравьев, после того как они завершают свои путешествия. Феромоны не только наносятся, но они и испаряются. Вероятность того, что муравей будет путешествовать из своего нынешнего города в какой-то другой город, пропорциональна количеству феромонов между городами. Предполагается, что муравьи также имеют некоторые знания о задаче, которая помогает им принимать решения во время своих путешествий. Они знают расстояние от своего текущего города до других городов, и они с большей вероятностью отправятся в близлежащий город, чем в отдаленный, так как цель алгоритма – найти кратчайший путь. Алгоритм муравьиной системы показан на рис. 10.6.

n = число городов
α, β = относительная важность феромонов относительно эвристической информации
Q = константа выделения феромона
ρ = скорость испарения $\in$ (0, 1)
$\tau_{ij} = \tau_0$ (исходный феромон между городами i и j) для $i \in [1, n]$ и $j \in [1, n]$
d_{ij} = расстояние между городами i и j для $i \in [1, n]$ и $j \in [1, n]$
While not (критерий останова)
  For $q = 1$ **to** $n - 1$
    For each муравей $k \in [1, N]$
      Инициализировать стартовый город c_{k1} каждого муравья $k \in [1, N]$.
      Инициализировать множество городов, посещенных муравьем k:
        $C_k \leftarrow \{c_{k1}\}$ для $k \in [1, N]$
      For each город $j \in [1, n]$, $j \notin C_k$
        Вероятность $p_{ij}^{(k)} \leftarrow \left(\tau_{ij}^{\alpha} / d_{ij}^{\beta}\right) / \left(\sum_{m=1, m \notin C_k}^{n} \tau_{im}^{\alpha} / d_{im}^{\beta}\right)$.
      Next j
      Пусть муравей k идет в город j с вероятностью $p_{ij}^{(k)}$.
      Использовать $c_{k,q+1}$ для обозначения города, отобранного в предыдущей строке.
      $C_k \leftarrow C_k \cup \{c_{k,q+1}\}$

Next муравей
Next q
$L_k \leftarrow$ длина общего пути, сконструированного муравьем k, для $k \in [1, N]$
For each город $i \in [1, n]$ и город $j \in [1, n]$
For each муравей $k \in [1, N]$
If муравей k прошел из города i в город j
$\Delta\tau_{ij}^{(k)} \leftarrow Q/L_k$
else
$\Delta\tau_{ij}^{(k)} \leftarrow 0$
End if
Next муравей
$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \sum_{k=1}^{N} \Delta\tau_{ij}^{(k)}$
Next пара городов
Next поколение

Рис. 10.6. Простая муравьиная система (AS) для решения задачи коммивояжера. Каждое поколение некоторое количество феромонов между городами i и j испаряется, но вместе с тем феромон также увеличивается из-за муравьев, которые путешествуют между этими двумя городами

На рис. 10.6 показано, что вероятность путешествия каждого муравья из города i в город j пропорциональна количеству феромонов на пути между этими городами и обратно пропорциональна расстоянию между этими городами. Соотношение α/β определяет относительную важность информации о феромонах относительно информации о расстоянии во время принятия решения, в какой город путешествовать. Когда муравей путешествует из города i в город j, количество феромонов на этом пути увеличивается пропорционально качеству решения муравья (то есть обратно пропорционально общему расстоянию путешествия муравья).

Рисунок 10.6 представляет собой довольно полный алгоритм, но при этом все еще остаются некоторые детали реализации, которые оставлены программисту. Например, верно ли, что $\tau_{ij} = \tau_{ji}$? В биологической муравьиной системе количество феромона между вершинами i и j такое же, как и между вершинами j и i, но при симулировании муравьиной системы это не обязательно так. Легко предположить, что путешествие из вершины i в j может привести к хорошему решению, в то время как путешествие из вершины j в i может привести к слабому решению. Это приведет к $\tau_{ij} \neq \tau_{ji}$, что соответствует асимметричной задаче коммивояжера (см. рис. 10.7).

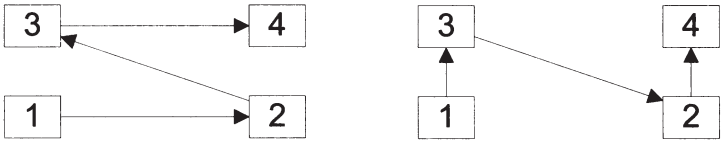


Рис. 10.7. В данном примере мы исходим из того, что маршрут начинается в вершине 1. Маршрут слева намного хуже (то есть он имеет большее расстояние), чем маршрут справа, но оба маршрута имеют пути между вершинами 2 и 3. Поскольку маршрут слева длинный, а маршрут справа короткий, мы рассчитываем, что для эффективного алгоритма оптимизации на основе муравьиной кучи τ_{23} будет меньше, чем τ_{32}; то есть он должен быть менее привлекательным для перехода из вершины 2 в 3, чем для перехода из вершины 3 в 2

Другие детали реализации, которые могут быть добавлены в рис. 10.6, включают в себя инициализацию, элитарность и мутацию. Во-первых, результативность оптимизации на основе муравьиной кучи, как и результативность эволюционного алгоритма, может сильно зависеть от правильной инициализации (см. раздел 8.1). Для задачи коммивояжера мы можем инициализировать определенных особей, используя простой эвристический алгоритм. Например, в первом поколении мы можем заставить одного из муравьев детерминированно посещать ближайший город в каждой точке принятия решения. Мы обсуждаем инициализацию задачи коммивояжера более подробно в разделе 18.2. Во-вторых, в оптимизации на основе муравьиной куче элитарность может применяться, как и в любом эволюционном алгоритме (см. раздел 8.4). Элитарность может быть встроена путем отслеживания лучших нескольких муравьев каждого поколения и заставления их повторять один и тот же маршрут в следующем поколении. Это гарантирует, что лучший маршрут не будет потерян из поколения в поколение. Алгоритм муравьиной системы с элитарностью иногда называют элитарной муравьиной системой [Dorigo и соавт., 1996], [Blum, 2005a]. В-третьих, в оптимизации на основе муравьиной кучи может применяться мутация так же, как и в любом эволюционном алгоритме (см. раздел 8.9). Мутация может быть встроена путем случайного изменения маршрутов с некоторой мутационной вероятностью. Исследователи предложили несколько механизмов мутации маршрутов в задаче коммивояжера, которые мы обсудим в разделе 18.4.

Рисунок 10.6 показывает, что для муравьиной системы следует отрегулировать несколько параметров. Эти параметры включают:

- число муравьев N, то есть размер популяции;

- α и β, то есть относительную важность количества феромонов и эвристической информации (эвристическую чувствительность);
- Q – константу выделения феромона;
- ρ – скорость испарения;
- τ_0, то есть исходное количество феромонов между каждым городом.

Несколькими исследователями были изучены эффекты этих параметров. В качестве типичного примера значений данных параметров рекомендуются [Dorigo и соавт., 1996] следующие:

- $N = n$ (то есть число муравьев = число городов);
- $\alpha = 1$ и $\beta = 5$;
- $Q = 100$, хотя его эффект незначителен;
- $\rho \in [0.5, 0.99]$;
- $\tau_0 \approx 10^{-6}$.

■ Пример 10.1

Данный пример применяет алгоритм муравьиной системы рис. 10.6 к задаче коммивояжера Berlin52, которая состоит из 52 местоположений в Берлине, Германия [Reinelt, 2008]. Задача Berlin52 – это симметричная задача коммивояжера, то есть нам дано множество вершин и расстояний между каждой парой вершин, и наша цель состоит в том, чтобы найти круговой путь минимальной общей длины при посещении каждой вершины ровно один раз. В симметричной задаче коммивояжера, такой как задача Berlin52, расстояние от вершины i до j совпадает с расстоянием от вершины j до i. Мы используем следующие ниже параметры муравьиной системы:

- $N = 53$;[1]
- $\alpha = 1$ и $\beta = 5$;
- $Q = 20$;
- $\rho = 0{,}9$;
- $\tau_0 = 10^{-6}$;
- в общем случае $\tau_{ij} \neq \tau_{ji}$;[2]
- случайная инициализация;
- два элитарных муравья в каждом поколении;
- отсутствие мутации.

[1] Стандартный размер популяции для алгоритма оптимизации на основе муравьиной кучи варьируется от одной исследовательской работы к другой. Во многих работах по алгоритму оптимизации на основе муравьиной кучи и в задаче коммивояжера используется $N = n$, тогда как в других используется $N = n + 1$.

[2] Для симметричных задач коммивояжера обычно рекомендуется $\tau_{ij} = \tau_{ji}$ (но это не обязательно).

На рис. 10.8 показан лучший маршрут исходной популяции с общим расстоянием 24 780. Мы видим, что лучший исходный маршрут выглядит довольно слабым. На рис. 10.9 показано схождение муравьиной системы, по мере того как она ищет лучший маршрут. Мы видим, что муравьиная система сходится очень быстро, а элитарность гарантирует, что общее расстояние лучшего маршрута никогда не будет увеличиваться из поколения в поколение. На рис. 10.10 показан лучший маршрут, найденный муравьиной системой после 10 поколений, с общим расстоянием 7796. Мы видим, что алгоритм оптимизации на основе муравьиной кучи нашел гораздо более оптимальный маршрут, чем лучший маршрут исходной популяции, и общее расстояние сократилось на 69 %.

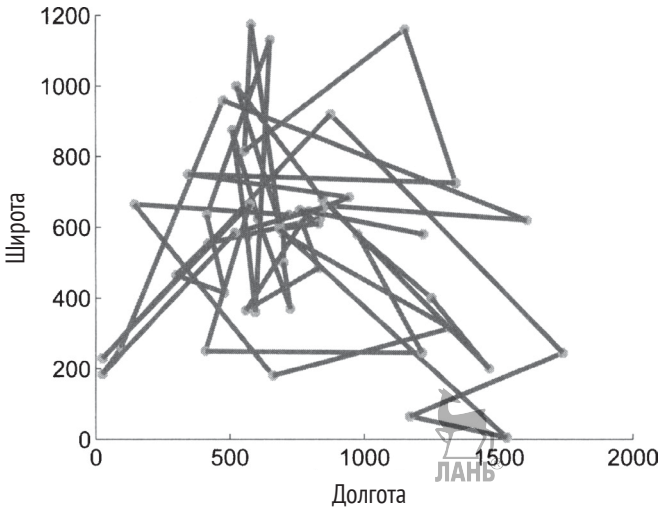


Рис. 10.8. Лучший исходный маршрут из 53 случайных маршрутов для примера 10.1 с общим расстоянием 24 780

Наконец, на рис. 10.11 показан глобально оптимальный маршрут, который, как было доказано, является оптимальным и который имеет общее расстояние 7542. Сравнивая рис. 10.10 и 10.11, мы видим, что алгоритм оптимизации на основе муравьиной кучи нашел маршрут, похожий на оптимальный маршрут, и он только на 3 % хуже, чем оптимальный маршрут. Каждая симуляция оптимизации на основе муравьиной кучи будет находить другое решение, так как оптимизация на основе муравьиной кучи является стохастическим алгоритмом. Но учитывая то, что существует $51! = 1.6 \times 10^{66}$ потенциальных решений этой задачи, алгоритм оптимизации на основе муравьиной кучи справ-

ляется довольно хорошо, находя решение, которое только на 3 % хуже оптимального.

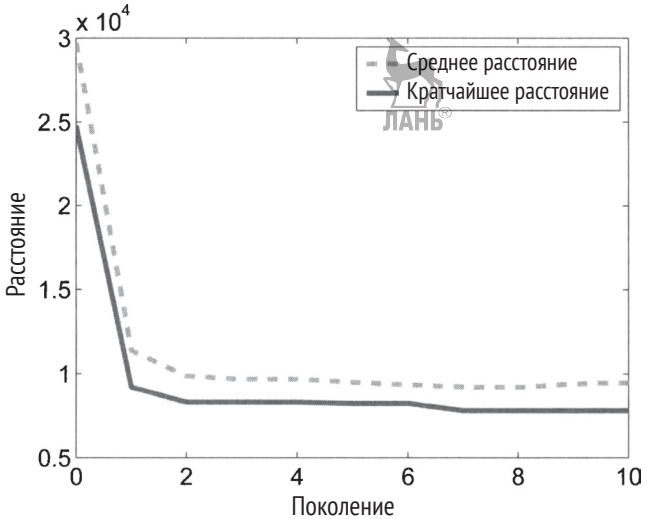


Рис. 10.9. Схождение муравьиной системы в примере 10.1

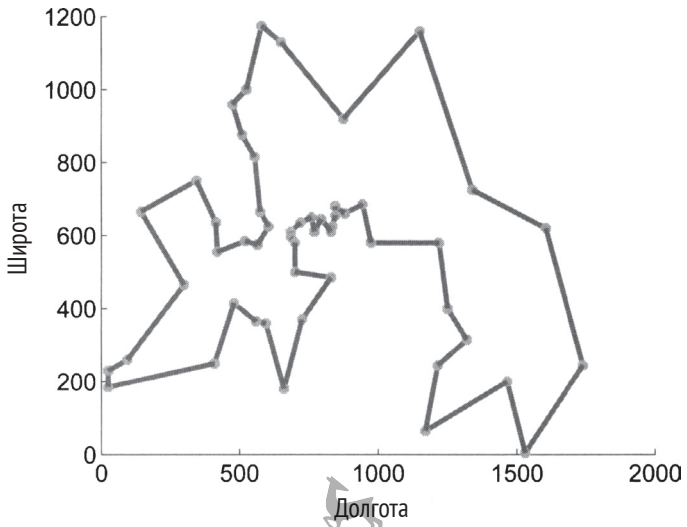


Рис. 10.10. Лучший маршрут найден после 10 поколений муравьиной системы для примера 10.1 с общим расстоянием 7796. Это на 69 % лучше, чем лучший исходный маршрут, показанный на рис. 10.8, и на 3 % хуже, чем глобально оптимальный маршрут, показанный на рис. 10.11

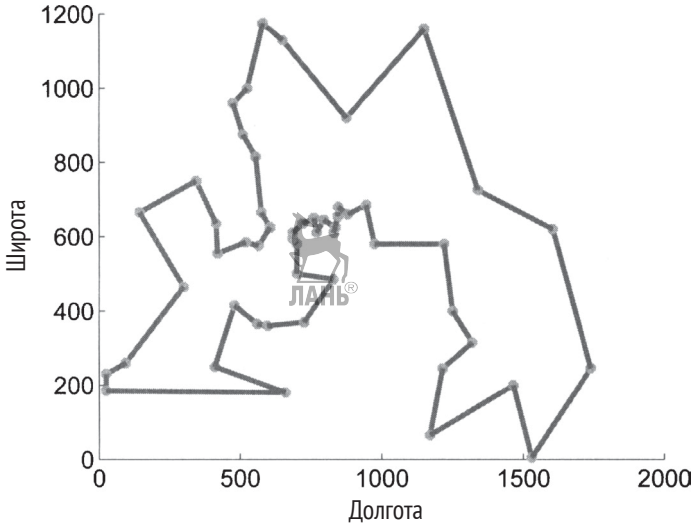


Рис.10.11. Глобально оптимальный маршрут для примера 10.1 с общим расстоянием 7542

■

10.3. Непрерывная оптимизация

Алгоритм оптимизации на основе муравьиной кучи был первоначально разработан для задач, подобных задаче коммивояжера, но с тех пор был модифицирован для оптимизационных задач с непрерывными областями [Socha и Dorigo, 2008], [Tsutsui, 2004], [de Franca et al, 2008]. Один из простых подходов к применению дискретного оптимизационного алгоритма, аналогичного алгоритму оптимизации на основе муравьиной кучи, к задаче с непрерывной областью заключается в разделении каждой размерности i поискового пространства на дискретизированные интервалы. То есть мы пытаемся минимизировать n-мерную задачу $f(x)$, где $x = [x_1, \ldots, x_n]$, и

$$x_i \in [x_{i,\min},\ x_{i,\max}]$$
$$x_{i,\min} = b_{i1} < b_{i2} < \ldots < b_{i,B_i} = x_{i,\max} \tag{10.3}$$

где $B_i - 1$ – это число дискретных интервалов, на которые мы делим i-ю область. Каждое поколение мы обновляем феромон этого интервала, как в алгоритме стандартной муравьиной системы, если i-я область кандидатного решения находится между b_{ij} и $b_{i,j+1}$:

$$\textbf{if } x_i \in [b_{ij}, b_{i,j+1}] \textbf{ them } \tau_{ij} = \tau_{ij} + Q/f(x), \tag{10.4}$$

где Q – это стандартная константа выделения феромона в муравьиной системе, и мы исходим из того, что $f(x) > 0$ для всех x. Уравнение (10.4) аналогично утверждению $\Delta\tau_{ij}^{(k)} \leftarrow Q/L_k$ на рис. 10.6. Мы используем количество феромонов для вероятностного построения новых решений в начале каждого поколения. Если интервал $[b_{ij}, b_{i,j+1}]$ имеет много феромонов, то существует большая вероятность того, что кандидатное решение будет построено таким образом, что его i-я размерность будет находиться в этом интервале. Один из способов сделать это – принять i-ю размерность кандидатного решения равной случайному числу $r \in [b_{ij}, b_{i,j+1}]$.

На рис. 10.12 схематично представлен алгоритм непрерывной муравьиной системы. На рис. 10.12 предполагается, что функция стоимости, обозначенная как L_k, положительна для всех k. Если это свойство для данной задачи не удовлетворяется, то стоимостные значения популяции должны быть сдвинуты так, чтобы она была удовлетворена. На рис. 10.12 не показана опция элитарности, но мы можем (и должны) легко включить элитарность, как описано для эволюционных алгоритмов в разделе 8.4. Непрерывная муравьиная система также может быть скомбинирована с локальным поиском, с тем чтобы, после того как муравей помещен в дискретный интервал, локальный поиск использовался для отыскания оптимального решения в этом интервале.

Использование дискретизированных интервалов для каждой размерности задачи является простым способом расширить муравьиную систему на непрерывные задачи. Более строгая реализация непрерывной муравьиной системы может использовать ядра для построения непрерывной аппроксимации дискретной функции плотности вероятности, которая представлена количествами феромонов [Simon, 2006, глава 15], [Blum, 2005a].

n = число размерностей

Разделить i-у размерность на $B_i - 1$ интервала, как показано в уравнении (10.3), $i \in [1, n]$

α = важность количеств феромона

Q = константа выделения феромона

ρ = скорость испарения $\in (0, 1)$

$\tau_{ij_i} = \tau_0$ (исходный феромон) для $i \in [1, n]$ и $j_i \in [1, B_i - 1]$

Случайно инициализировать популяцию муравьев (кандидатные решения) a_k, $k \in [1, N]$

While not (критерий останова)
  For each муравей a_k, $k \in [1, N]$
    For each размерность $i \in [1, n]$
      For each дискретизированный интервал $[b_{ij}, b_{ij+1}]$, $j \in [1, B_i - 1]$
        Вероятность $p_{ij}^{(k)} \leftarrow \tau_{ij}^{\alpha} / \sum_{m=1}^{B_i-1} \tau_{im}^{\alpha}$
      Next дискретизированный интервал
      $a_k(x_i) \leftarrow U[b_{ij}, b_{ij+1}]$ с вероятностью $p_{ij}^{(k)}$
    Next размерность
  Next муравей
  $L_k \leftarrow$ стоимость решения, сконструированного муравьем a_k, $k \in [1, N]$
  For each размерность $i \in [1, n]$
    For each дискретизированный интервал $[b_{ij}, b_{ij+1}]$, $j \in [1, B_i - 1]$
      For each муравей a_k, $k \in [1, N]$
        If $a_k(x_i) \in [b_{ij}, b_{ij+1}]$
          $\Delta\tau_{ij}^{(k)} \leftarrow Q/L_k$
        else
          $\Delta\tau_{ij}^{(k)} \leftarrow 0$
        End if
      Next муравей
      $\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \sum_{k=1}^{N} \Delta\tau_{ij}^{(k)}$
    Next дискретизированный интервал
  Next размерность
Next поколение

Рис. 10.12. Простая муравьиная система (AS) для решения задачи минимизации непрерывной области. $a_k(x_i)$ – это i-й элемент k-го кандидатного решения. Каждое поколение феромон в каждом интервале испаряется, но феромон также увеличивается в количестве, пропорциональном числу муравьев, которые строят кандидатное решение в этом интервале. $U[b_{ij}, b_{ij+1}]$ – это случайное число, равномерно распределенно между b_{ij+1} и b_{ij+1}

■ Пример 10.2

В данном примере мы оптимизируем 20-мерную функцию Экли (см. приложение С.1.2). Мы применяем алгоритм рис. 10.12 со следующими параметрами:

- $N = 50$;
- $\alpha = 1$;
- $Q = 20$;
- $\rho = 0.9$;
- $\tau_0 = 10^{-6}$;

- два элитарных решения каждое поколение;
- скорость мутации = 1 % на размерность на особь в поколении;
- число интервалов $B_i = 40$ или 80 для $i \in [1, n]$.

На рис. 10.13 показано лучшее решение для каждого поколения, усредненно по 20 симуляциям Монте-Карло для интервала 40 и для интервала 80 на размерность. Мы видим, что схождение оказывается лучше для большего числа интервалов на размерность, однако время вычисления увеличивается по мере увеличения числа интервалов. Для этого есть две причины, которые видны на рис. 10.12. Первая причина – циклы «for each дискретизированный интервал». Вторая причина в том, что сложнее принять решение о том, в каком интервале разместить $a_k(x_j)$. Однако следует отметить, что для большинства реальных оптимизационных задач вычисление функции стоимости является основным вычислительным соображением, поэтому дополнительные вычисления, необходимые для дискретизированных интервалов областей, не могут представлять серьезную проблему (см. главу 21).

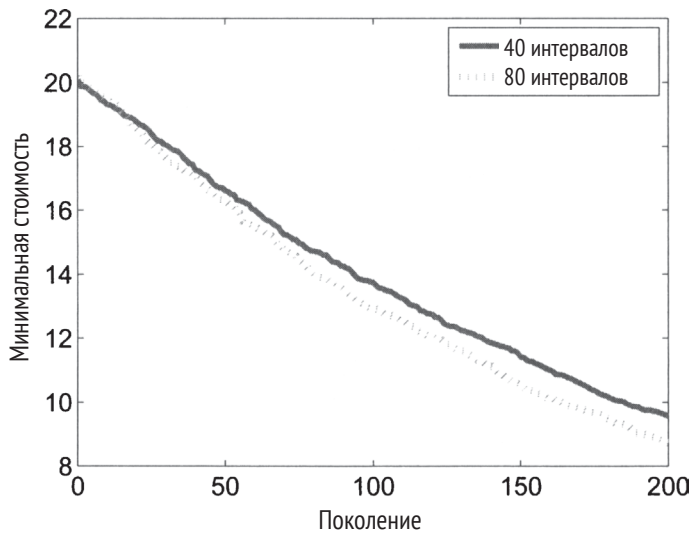


Рис. 10.13. Пример 10.2: схождение непрерывной муравьиной системы применительно к 20-мерной функции Экли. Графики показывают лучшее решение в каждом поколении, усредненно по 20 симуляциям Монте-Карло. Мы получаем более высокую результативность, если распределяем феромон на большее число интервалов на размерность

■

10.4. Другие муравьиные системы

В алгоритм стандартной муравьиной системы, который описан в предыдущих разделах, был внесен ряд модификаций. В этом разделе описываются две основные модификации: минимаксная муравьиная система в разделе 10.4.1 и система муравьиной кучи в разделе 10.4.2.

10.4.1. *Минимаксная муравьиная система*

Минимаксная муравьиная система (max-min ant system, MMAS) – это простая модификация алгоритма стандартной муравьиной системы [Dorigo и соавт., 2006], [Stützle и Hoos, 2000]. Она характеризуется двумя основными особенностями. Во-первых, каждое поколение феромон увеличивается только лучшим муравьем. Это приводит к сокращению разведывания и увеличению эксплуатации лучшего из известных решений. Во-вторых, количество феромонов ограничено сверху и снизу. Это имеет противоположный эффект, то есть это увеличивает разведывание, потому что даже худшие маршруты сохраняют ненулевое количество феромона, и даже лучшие маршруты не могут получить достаточно феромона на то, чтобы они стали полностью доминировать над решениями муравьев.

Первое различие между алгоритмом стандартной муравьиной системы и минимаксной муравьиной системы можно увидеть в следующих уравнениях, которые заменены на рис. 10.6 и 10.12:

Стандартная муравьиная система: $\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \sum_{k=1}^{N} \Delta\tau_{ij}^{(k)}$

Минимаксная муравьиная система: $\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \Delta\tau_{ij}^{(\text{лучший})}$ (10.5)

где (лучший) – это индекс лучшего кандидатного решения. В задаче коммивояжера рис. 10.6 $\Delta\tau_{ij}^{(\text{лучший})}$ задается как

$$\Delta\tau_{ij}^{(\text{лучший})} = \begin{cases} Q/L_{\text{лучший}} & \text{если город } i \rightarrow \text{город } j \text{ принадлежит лучшему маршруту} \\ 0 & \text{в противном случае} \end{cases} \quad (10.6)$$

а в непрерывной муравьиной системе на рис. 10.12 $\Delta\tau_{ij}^{(\text{лучший})}$ задается как

$$\Delta\tau_{ij}^{(\text{лучший})} = \begin{cases} Q/L_{\text{лучший}} & \text{если } i\text{-я размерность лучшей особи} \\ 0 & \text{в противном случае} \end{cases} \quad (10.7)$$

Второе различие между алгоритмами стандартной муравьиной системы и минимаксной муравьиной системы реализуется следующими простыми уравнениями после обновления:

$$\begin{aligned} \tau_{ij} &\leftarrow \max(\tau_{ij}, \tau_{\min}) \\ \tau_{ij} &\leftarrow \min(\tau_{ij}, \tau_{\max}) \end{aligned} \tag{10.8}$$

где $\tau_{\min}$ и $\tau_{\max}$ настроены для конкретной оптимизируемой задачи.

При некотором воображении мы можем увидеть, что минимаксная муравьиная система может быть обобщена несколькими различными способами. Например, вместо того чтобы позволять наносить феромон только лучшему муравью, мы можем позволить наносить феромон лучшим M муравьям, где M – это регулировочный параметр. Либо мы можем позволить m-му лучшему муравью с вероятностью p_m наносить феромон, где p_m уменьшается с увеличением стоимости. Исходя из того, что мы хотим больше разведывания в начале процесса оптимизации и больше эксплуатации в конце процесса, мы можем увеличивать $(\tau_{\max} - \tau_{\min})$ по мере увеличения числа поколений. С применением некоторого воображения и экспериментов мы, несомненно, можем также найти другие расширения минимаксной муравьиной системы, которые улучшат ее результативность на различных видах задач.

■ Пример 10.3

В данном примере мы повторим минимизацию функции Экли с N = 20 размерностями, как в примере 10.2. Мы используем следующие ниже параметры:

- $N = 40$;
- $\alpha = 1$;
- $Q = 20$;
- $\rho = 0.9$;
- $\tau_0 = 10^{-6}$;
- два элитарных решения каждое поколение;
- скорость мутации = 1 % на размерность на особь в поколении;
- число интервалов $B_i = 20$ для $i \in [1, n]$;
- $\tau_{\min} = 0$, $\tau_{\max} = \infty$.

Мы позволяем только M муравьям наносить феромон:

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \Delta\tau_{ij}^{(\text{лучший}_m)} \tag{10.9}$$

для $m \in [1, M]$, где (лучший$_m$) – это индекс m-й лучшей особи каждое поколение. То есть только лучшие M муравьев наносят феромон на области, которые они разведали. Кроме этого изменения, данный алгоритм, используемый нами в этом примере, совпадает с алгоритмом муравьиной системы в примере 10.2. На рис. 10.14 показано лучшее решение для каждого поколения, усредненно по 20 симуляциям Монте-Карло, для $M = 4$ и $M = 40$. Мы видим, что схождение намного лучше, когда разрешено наносить феромоны меньшему числу муравьев. В интуитивном плане это имеет смысл. Мы не хотим, чтобы слабые особи усиливали свои решения.

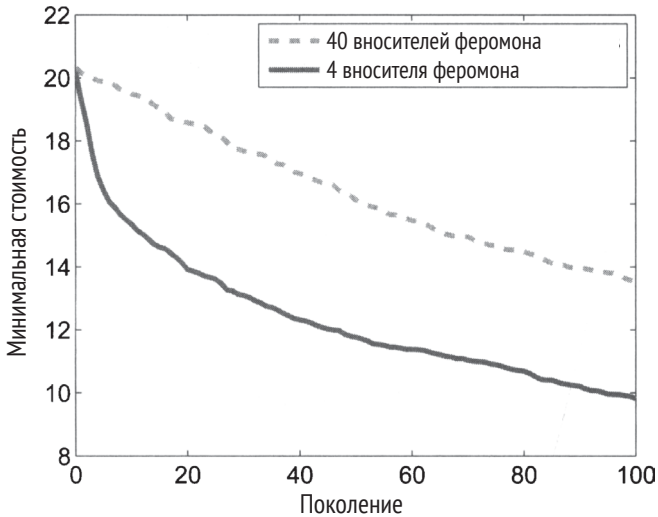


Рис. 10.14. Пример 10.3: схождение непрерывной муравьиной системы применительно к 20-мерной функции Экли. Графики показывают лучшее решение в каждом поколении, усредненно по 20 симуляциям Монте-Карло. Мы получаем более высокую результативность, если позволяем только лучшим муравьям наносить феромон на своих решениях

■

10.4.2. *Система муравьиной кучи*

Система муравьиной кучи (ant colony system, ACS) представляет собой расширение муравьиной системы [Dorigo и Gambardella, 1997a], [Dorigo и Gambardella, 1997b], [Dorigo и соавт. 2006]. Несмотря на их общие корни, муравьиная система и система муравьиной кучи довольно разные по своему поведению и результативности. Система муравьиной кучи

характеризуется двумя основными расширениями муравьиной системы. Во-первых, при построении решения локальное обновление феромонов реализуется каждым муравьем. Когда муравей путешествует из города i в город j, феромон вдоль этого пути обновляется следующим образом:

$$\tau_{ij} \leftarrow (1 - \phi)\tau_{ij} + \phi\tau_0, \tag{10.10}$$

где $\phi \in [0, 1]$ – это константа локального распада феромонов, а τ_0 – исходное количество феромонов. Если $\phi = 0$, то τ_{ij} не изменяется, и мы возвращаемся к исходной муравьиной системе. Уравнение (10.10) указывает, что феромон между городами i и j распадается, когда муравьи путешествуют по этому пути. Этот подход не является точным с биологической точки зрения[3], но он дестимулирует других муравьев следовать по тому же пути и, следовательно, способствует разведыванию и многообразию. После того как все муравьи построили кандидатное решение, мы реализуем одно из стандартных правил глобального обновления феромонов из уравнения (10.5).

Второе расширение, которое система муравьиной кучи вносит в муравьиную систему, является использованием псевдослучайного пропорционального правила для построения кандидатного решения. Обозначим через $(a_k \rightarrow j)$ событие, что k-й муравей при построении своего кандидатного решения идет в город j. Обозначим через $\Pr(a_k \rightarrow j)$ вероятность того, что $(a_k \rightarrow j)$. Разница между построением кандидатного решения стандартной муравьиной системой и построением кандидатного решения системой муравьиной кучи заключается в следующем:

Муравьиная система: $\Pr(a_k \rightarrow j) = p_{ij}^{(k)}$

$$\text{Система муравьиной кучи:} \quad \Pr(a_k \rightarrow j) = \begin{cases} \left.\begin{array}{ll} 1 & \text{если } j = \arg\max_l p_{il}^{(k)} \\ 0 & \text{в противном случае} \end{array}\right\} & \text{если } r < q_0 \\ p_{ij}^{(k)} & \text{если } r \geq q_0 \end{cases} \tag{10.11}$$

где r – случайное число, взятое из равномерного распределения на $[0, 1]$, а $q_0 \in [0, 1]$ – регулировочный параметр. В стандартной муравьиной системе вероятности выводятся, используя количество феромона,

[3] Расширения к алгоритмам оптимизации на основе муравьиной кучи и эволюционным алгоритмам часто отклоняются от биологических основ этих алгоритмов, но наша цель в первую очередь заключается в разработке эффективных оптимизационных алгоритмов, а не в точной модели биологической жизни. Биологические корни оптимизационных алгоритмов на основе муравьиной кучи и эволюционных алгоритмов главным образом служат в качестве вдохновляющего фактора.

и муравей k решает, в какой город ему идти, основываясь на этих вероятностях (см. рис. 10.6 и 10.12). Однако в системе муравьиной кучи существует q_0 вероятность того, что муравей k отправится в город, имеющий наибольшую вероятность (то есть с наибольшим количеством феромонов, ведущих в него из текущего города, обозначаемом функцией arg max в уравнении (10.12)); и есть вероятность $(1 - q_0)$, что для принятия решения, в какой город ему идти, муравей k применит правило стандартной муравьиной системы. Это смещает муравьев в сторону разведывания весьма перспективных вариантов в построении их решения. Такой подход концептуально эквивалентен увеличению вероятности высокоферомонных путей, что эквивалентно увеличению на рис. 10.6 и 10.12.

Вероятности системы муравьиной кучи уравнения (10.11) при $r > q_0$ точны только приближенно. Для большей точности их следует нормализовать так, чтобы они в сумме составляли 1 (см. задачу 10.7).

■ Пример 10.4

В данном примере мы исследуем использование константы локального распада феромонов ϕ в системе муравьиной кучи. Как и в предыдущих примерах в этой главе, мы минимизируем функцию Экли с $N = 20$ размерностями. Мы используем следующие параметры:

- $N = 40$;
- $\alpha = 1$;
- $Q = 20$;
- $\rho = 0.9$;
- $\tau_0 = 10^{-6}$;
- два элитарных решения каждое поколение;
- скорость мутации = 1 % на размерность на особь в поколении;
- число интервалов $B_i = 20$ для $i \in [1, n]$;
- $\tau_{\min} = 0$, $\tau_{\max} = \infty$;
- лучшие 4 муравья наносят феромон каждое поколение;
- константа разведывания $q_0 = 0$.

На рис. 10.15 показано лучшее решение для каждого поколения, усредненно по 20 симуляциям Монте-Карло, для $\phi = 0$, 0.001 и 0.01. Мы видим, что результативность заметно лучше с ненулевым значением для константы локального распада феромонов. Положительное значение ϕ стимулирует больший объем разведывания, что приводит к более быстрому схождению. Однако если значение ϕ слишком велико, то

другие муравьи дестимулируются разведывать ранее использованные пути, и в результате результативность ухудшается. Для того чтобы сделать более определенные выводы, мы должны провести тесты на статистическую значимость результатов рис. 10.15 (см. приложения В.2.4 и В.2.5).

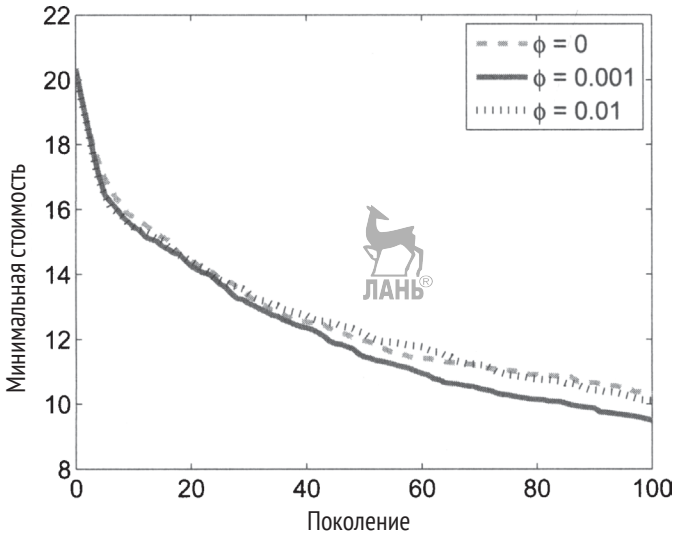


Рис. 10.15. Пример 10.4: результативность системы муравьиной колонии (ACS) на 20-мерной функции Экли. Следы показывают лучшее решение в каждом поколении, усредненно по 20 симуляциям Монте-Карло для разных значений константы локального распада феромонов. Мы получаем более высокую результативность при $\phi > 0$, но если значение ϕ слишком большое, то результативность страдает

■

■ Пример 10.5

В данном примере мы исследуем использование константы разведывания q_0 в системе муравьиной кучи. Как и в предыдущих примерах в этой главе, мы минимизируем функции Экли с $n = 20$ размерностями. Мы используем те же параметры системы муравьиной кучи, что и в примере 10.4, за исключением того, что задаем значение константы локального распада феромонов $\phi = 0$ и проверяем различные значения q_0. На рис. 10.16 показано лучшее решение для каждого поколения, усредненно по 100 симуляциям Монте-Карло, для $q_0 = 0$, 0.001 и 0.01.

Мы видим, что результативность немного лучше с ненулевым значением константы разведывания. Положительное значение q_0 обеспечивает большее смещение муравьев в сторону использования более благоприятных признаков решения. Однако если q_0 слишком велико, то система мравьиной кучи не выполняет достаточного разведывания, и результативность становится хуже. Для того чтобы сделать более определенные выводы, мы должны провести тесты на статистическую значимость результатов рис. 10.16 (см. приложения B.2.4 и B.2.5). Также примите во внимание, что эти результаты сильно зависят от конкретной решаемой задачи и от других параметров, перечисленных в предыдущем примере. В большинстве реализаций алгоритма системы муравьиной кучи используются высокие значения q_0, такие как $q_0 = 0.9$ [Dorigo и Gambardella, 1997b].

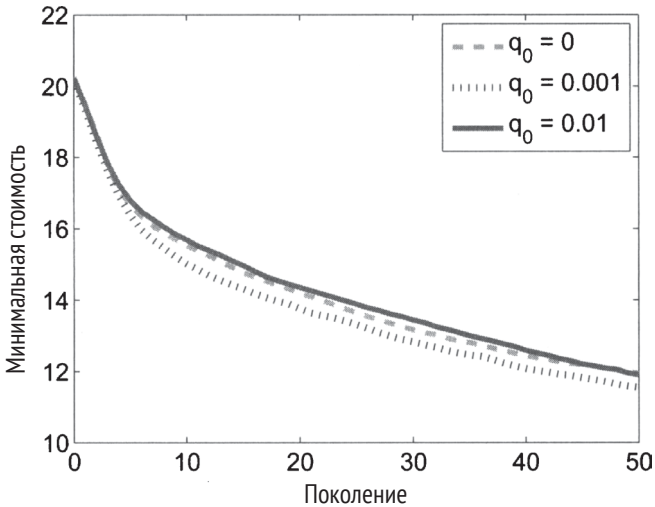


Рис. 10.16. Результативность в примере 10.5 системы муравьиной кучи на 20-мерной функции Экли. Следы показывают лучшее решение в каждом поколении, усредненно по 100 симуляциям Монте-Карло для разных значений константы разведывания. Мы получаем более высокую результативность с $q_0 > 0$, но если значение q_0 слишком велико, то результативность страдает

■

10.4.3. *Еще больше муравьиных систем*

Пространство книги не позволяет нам вдаваться в подробности в отношении других муравьиных систем, но есть несколько заметных

вариаций, о которых мы кратко здесь упомянем. В элитарной муравьиной системе лучшее решение наносит феромон всякий раз, когда другие муравьи наносят феромон [Dorigo и Stützle, 2004, глава 3]. Следовательно, расчет $\Delta\tau$ на рис. 10.12 изменяется следующим образом:

$$\begin{aligned}&\text{Стандартная муравьиная система: } \Delta\tau_{ij}^{(k)} \leftarrow \delta_{ij}^{(k)} Q/L_k \\ &\text{Элитарная муравьиная система: } \Delta\tau_{ij}^{(k)} \leftarrow \delta_{ij}^{(k)} Q/L_k + \delta_{ij}^{(\text{лучший})} Q/L_{\text{лучший}}\end{aligned} \tag{10.12}$$

где $\delta_{ij}^{(k)} = 1$, если i-я размерность k-го кандидатного решения лежит в j-м дискретизированном интервале, а (лучший) – это индекс лучшей особи в популяции. Мы видим, что в элитарной муравьиной системе всякий раз, когда муравей наносит феромон, лучший муравей делает то же самое.

Система Ant-Q является гибридом муравьиной системы и Q-самообучения [Gambardella и Dorigo, 1995]. В ранговых муравьиных системах количество отложенных феромонов зависит не только от качества решения муравья, но и от его ранга по отношению к другим муравьям [Dorigo и Stützle, 2004, глава 3]. Аппроксимированный недетерминированный древовидный поиск (approximated non-deterministic tree search, ANTS) задает определенные механизмы для определения степени привлекательности хода и способа обновления феромонов [Maniezzo и соавт., 2004]. Муравьиная система «лучший–худший» (best-worst AS) наносит дополнительные феромоны на лучшее решение, применяет дополнительное испарение на худшее решение, а также использует мутацию для стимулирования разведывания [Cordon и соавт., 2000]. Гиперкубовый алгоритм оптимизации на основе муравьиной кучи (hypercube ACO algorithm) ограничивает количество феромонов интервалом [0, 1] с целью регуляризации поведения оптимизационного алгоритма на задачах с разными целевыми критериями и облегчения теоретического разведывания [Blum и Дориго, 2004]. Популяционно-ориентированная оптимизация на основе муравьиной кучи поддерживает популяцию феромоновых историй, вместо того чтобы хранить всю информацию в едином феромоновом словаре; в данном алгоритме эта популяция используется для модификации алгоритма обновления [Guntsch и Middendorf, 2002]. Лучевой алгоритм оптимизации на основе муравьиной кучи (Beam ACO) является гибридом алгоритма оптимизации на основе муравьиной кучи и лучевого поиска, то есть популярного алгоритма древовидного поиска [Blum, 2005b].

10.5. Теоретические результаты

С тех пор как экспериментальные результаты впервые стали показывать, что оптимизация на основе муравьиной кучи работает, исследователи принялись разрабатывать теорию оптимизации на основе муравьиной кучи, с тем чтобы объяснить, когда, почему и как она работает. Первые доказательства сходимости оптимизации на основе муравьиной кучи были приведены в работе [Gutjahr, 2000]. С того времени были опубликованы разные доказательства сходимости для разных типов алгоритмов оптимизации на основе муравьиной кучи [Dorigo и Stützle, 2004]. Большинство этих доказательств утверждает что-то вроде: «при наличии достаточного времени алгоритм оптимизации на основе муравьиной кучи в конечном итоге найдет лучшее решение комбинаторно-оптимизационной задачи». Такие результаты схождения интересны математически, но имеют ограниченный практический интерес. До тех пор, пока феромон вдоль каждой ветви поддерживается в нижней и верхней границах, как и в минимаксной муравьиной системе, всегда существует ненулевая вероятность того, что каждый муравей разведает каждую возможную ветвь пространства решений, так что при наличии достаточного времени каждая ветвь будет разведана. Это означает, что в конечном итоге будет найдено оптимальное решение. Конечно, любой алгоритм стохастического поиска с ненулевой вероятностью поиска каждого возможного кандидатного решения в конечном итоге сойдется. Даже простейший случайный поиск в конечном итоге сойдется [Bäck, 1996].

Более интересные теоретические результаты лежат в области времени до схождения [Gutjahr, 2008], [Neumann и Witt, 2009], вероятности схождения в течение заданного времени, масштабируемости вместе с размером задачи и описательных математических моделей, таких как марковские модели или системно-динамические модели (см. главу 4 относительно математических моделей для генетических алгоритмов). Обратите внимание, что теоретические результаты для комбинаторных задач сильно отличаются от теоретических результатов для задач с непрерывной областью. Кроме того, если оптимизацию на основе муравьиной кучи можно было бы продемонстрировать как эквивалентную другим алгоритмам оптимизации, для которых существуют более интересные доказательства сходимости, то эти доказательства сходимости могли бы быть адаптированы к оптимизации на основе муравьиной кучи с целью укрепления ее теоретических основ.

Уже было показано, что при определенных условиях оптимизация на основе муравьиной кучи эквивалентна алгоритмам стохастического градиентного восхождения (stochastic gradient ascent, SGA) и перекрестной энтропии (cross entropy, CE) [Meuleau02 и Dorigo, 2002], [Zlochin и соавт., 2004], [Dorigo и Stützle, 2004]. Алгоритмы стохастического градиентного восхождения и перекрестной энтропии являются модельно-ориентированными оптимизационными алгоритмами, которые строят решения на основе параметризованного распределения вероятностей на поисковом пространстве. Оценивание кандидатных решений используется для модификации распределения вероятностей таким образом, чтобы оно было смещено в сторону лучших кандидатных решений.

10.6. Заключение

Некоторые исследователи оптимизации на основе муравьиной кучи подчеркивают, что данная оптимизация является не алгоритмом, а метаэвристикой из-за ее многочисленных вариаций. Вместе с тем мы можем сказать то же самое о любом из алгоритмов, обсуждаемых в этой книге (эволюционные алгоритмы, эволюционное программирование, эволюционные системы, генетическое программирование и т. д.). Все они имеют ряд вариаций, и поэтому все они являются метаэвристиками. Разница между алгоритмом и метаалгоритмом является степенной, и поэтому эта разница не является черно-белой. Большинство исследователей оптимизации на основе муравьиной кучи подчеркивает, что алгоритм оптимизации на основе муравьиной кучи не является эволюционным алгоритмом, потому что особи не обмениваются информацией друг с другом в традиционном смысле эволюционных алгоритмов. Как мы увидели в этой главе, несмотря на то что параметры построения решения данного алгоритма эволюционируют с течением времени, надо признать, что особи данного алгоритма непосредственно не обмениваются информацией друг с другом.

Подавляющая часть нашего обсуждения в этой главе была сосредоточена на следе феромонов. Однако муравьи наносят другие феромоны для целей, отличных от разметки путей. Типичная муравьиная куча использует до 20 различных феромонов [Hölldobler и Wilson, 1990, глава 7]. Например, муравьи выпускают феромоны сигнала тревоги, когда они задавлены. Это может стимулировать других муравьев к агрессивной борьбе с хищником, раздавившим их коллегу [Sobotnfk и соавт. 2008]. Эти типы феромонов могут быть просимулированы в оптимизационном алгоритме на основе муравьиной кучи за счет того, что

слабое решение распространяет информацию, которая не дает другим особям повторять его слабую стратегию. Это подобно отрицательному подкреплению в оптимизации на основе роя частиц, обсуждаемой в разделе 11.6.

Муравьи-самки наносят эпидеиктичные[4] феромоны, когда они откладывают яйца, для того чтобы просигнализировать самкам одного вида откладывать свои яйца в другом месте [Gómez и соавт., 2005], животные наносят территориальные феромоны для обозначения своей территории [Horne и Jaeger, 1988]. Территориальные феромоны присутствуют в моче кошек и собак, которые они наносят на границах заявленной ими территории. Животные выпускают феромоны секса, для того чтобы сообщить об их готовности к спариванию [Wyatt, 2003]. Муравьи выпускают феромоны рекрутирования, для того чтобы привлечь других муравьев в какое-то место, где требуется работа [Hölldobler и Wilson, 1990]. Эти типы феромонов могут быть просимулированы в алгоритме оптимизации на основе муравьиной кучи путем распространения информации о ранее разведанной территории в поисковом пространстве, для того чтобы не дать другим особям проводить поиск в участках, которые уже были разведаны, или стимулировать других особей разведывать перспективные участки поисковой области. Муравьи также могут выпускать феромоны для конкретных задач [Greene и Gordon, 2007]. Это может быть просимулировано в алгоритме оптимизации на основе муравьиной кучи для многокритериальной оптимизации с разными особями, преследующими оптимизацию разных подзадач. Как мы видим, для биологически обусловленных расширений алгоритма оптимизации на основе муравьиной кучи существует целый ряд возможностей.

Дополнительную информацию для чтения об алгоритме оптимизации на основе муравьиной кучи можно найти в книгах [Bonabeau и соавт., 1999], [Dorigo и Stützle, 2004], [Solnon, 2010]; главах из книги [Maniezzo и соавт., 2004], [Dorigo и Stützle, 2010] и учебных статьях [Blum, 2005a], [Blum, 2007]. Другие направления дальнейших исследований в области оптимизации на основе муравьиной кучи аналогичны приоритетам исследований для других оптимизационных алгоритмов [Dorigo и соавт., 2006], включая ответы на вопросы, как оптимизация на основе муравьиной кучи может применяться к задачам динамической оптимизации, для которых поисковое пространство изменяется со временем, и как данная оптимизация может применяться к стохастическим

[4] Эпидеиктичный (epideictic) – подгоняющий скорость увеличения численности популяции к наличию соответствующих ресурсов в окружающей среде. – *Прим. перев.*

оптимизационным задачам с вычислением шумных функций приспособленности (см. главу 21), как данная оптимизация может применяться к многокритериальным оптимизационным задачам (см. главу 20), как оптимизацию на основе муравьиной кучи можно гибридизировать с другими эволюционными алгоритмами.

Задачи

Письменные упражнения

10.1. Приведите пример в реальной прикладной задаче, когда стоимость перемещения из вершины A в вершину B будет отличаться от стоимости перемещения из вершины B в вершину A.

10.2. Пусть t есть общее число муравьев, такое что $m_1 \approx p_1 t$ и $m_2 \approx p_2 t$ в уравнении (10.1).

а) Каковы равновесные соотношения p_1/p_2?

б) Какие из равновесных соотношений устойчивы, а какие неустойчивы?

10.3. Предположим, что в муравьиной системе на рис. 10.6 $\beta = 1$. Если два сегмента пути имеют одинаковое количество феромонов, а сегмент 1 вдвое короче сегмента 2, насколько более вероятно, что муравей будет путешествовать по сегменту 1, чем сегменту 2? Что делать, если $\beta = 2$? Что делать, если $\beta = 3$?

10.4. Муравьиная система на рис. 10.6 устанавливает феромонное отложение k-го муравья равным $\Delta\tau_{ij}^{(k)} \leftarrow \delta_{ij}^{(k)} Q/L_k$, где $\delta_{ij}^{(k)} = 1$, если k-й муравей пришел из города i в город j, и $\delta_{ij}^{(k)} = 0$ в противном случае. Предположим, что вместо этого мы устанавливаем его равным $\delta_{ij}^{(k)} \epsilon\tau_{ij}$, где ϵ – это регулировочный параметр.

а) Какой диапазон ϵ стабилизирует уравнение обновления феромона?

б) Каково равновесное значение τ_{ij} в этом случае? Является ли это значение желательным равновесным значением?

10.5. В стандартной непрерывной области, как показано на рис. 10.12, феромонное отложение m-го муравья равно $\Delta\tau_{ij}^{(m)} = Q/L_m$. Предположим, что вместо этого мы позволим m-му муравью нанести

феромон с вероятностью p_m, где p_m уменьшается с увеличением стоимости, как упоминалось в конце раздела 10.4.1:

$$p_m \leftarrow \frac{1}{L_m} \sum_{r=1}^{N} L_r$$

$r \leftarrow U[0, 1]$ – то есть r – это случайное число, равномерно распределенное на [0, 1]

If $r < p_m$ **then**

$\Delta\tau_{ij}^{(m)} \leftarrow Q_1/L_m$

else

$\Delta\tau_{ij}^{(m)} \leftarrow 0$

End if

Какое значение мы должны использовать для Q_1 в приведенном выше алгоритме, для того чтобы среднее количество феромонов, нанесенных m-м муравьем, было равно тому, которое было нанесено стандартной муравьиной системе на рис. 10.12?

10.6. Как вычислительное усилие в муравьиной системе с непрерывной областью на рис. 10.12 увеличивается с увеличением размера популяции? Как оно увеличивается с размерностью задачи? Как оно увеличивается с числом дискретизированных интервалов в каждой размерности?

10.7. Вероятности системы муравьиной кучи

а) Предположим, что у вас есть система муравьиной кучи с четырьмя городами и $q_0 = 1/2$. Предположим, что k-й муравей находится в городе 1 и что

$p_{11}^{(k)} = 0$
$p_{12}^{(k)} = 1/4$
$p_{13}^{(k)} = 1/4$
$p_{14}^{(k)} = 1/2$

Какова, согласно уравнению (10.11), вероятность того, что k-й муравей проследует в каждый из четырех городов? Эти вероятности равны 1?

б) Нормализуйте вероятности системы муравьиной кучи $\Pr(a_k \to j)$ уравнения (10.11) так, чтобы сумма от $j = 1$ до n равнялась 1.

c) Используйте свой ответ на вопрос части (b) для вычисления новых вероятностей для сценария, описанного в части (a). Новая вероятность составляет в сумме 1?

10.8. Предложите способ реализации ранговой муравьиной системы, подобной той, которая упомянута в разделе 10.4.3.

10.9. Предложите способ реализации муравьиной системы «лучший–худший», подобной той, которая упомянута в разделе 10.4.3.

Компьютерные упражнения

10.10. В этой задаче исследуется влияние β, то есть эвристической чувствительности муравьиной системы, на результативность муравьиной системы. Просимулируйте муравьиную систему примера 10.1 двадцать раз, записывая лучшую стоимость среди всех муравьев в каждом поколении. Постройте график среднего значения 20 симуляций Монте-Карло как функции от числа поколений. Выполните это для $\beta = 0.1$, 1 и 10. Обсудите свои результаты.

10.11. Повторите пример 10.3 с $M = 40$. Выполните 20 симуляций Монте-Карло для каждого из следующих значений $\tau_{\min}$: 0, 0.001, 0.01 и 0.1. Постройте график результатов. Прокомментируйте влияние $\tau_{\min}$ на результативность муравьиной системы.

10.12. Повторите пример 10.3 с $M = 40$. Выполните 20 симуляций Монте-Карло для каждого из следующих значений $\tau_{\max}$: 1, 10, 100 и ∞. Постройте график результатов. Прокомментируйте влияние $\tau_{\max}$ на результативность муравьиной системы.

Глава 11

Оптимизация на основе роя частиц

Алгоритм на основе роя частиц имитирует социальное поведение человека.

– Джеймс Кеннеди и Рассел Эберхарт [Kennedy и Eberhart, 2001]

Мы наблюдаем коллективный разум во многих природных системах. Например, муравьи демонстрируют чрезвычайно высокий уровень коллективного интеллекта, как мы обсуждали в начале главы 10. В таких системах интеллект не принадлежит отдельным особям, а распределяется между группой, состоящей из многих особей. Это можно увидеть в стаях животных, когда они убегают от хищников, ищут пищу, ищут пути для быстрого перемещения и т. п.

Группы животных часто могут избегать хищников эффективнее в группе, чем в одиночку. Например, льву, возможно, легко распознать одну зебру из-за ее контраста с окружающим ландшафтом, но группа зебр смешивается вместе, и их труднее распознать как отдельных особей [Stone, 2009]. Группа животных также, возможно, будет выглядеть больше, или звучать громче, или выглядеть более угрожающей иными способами, чем одиночное животное. Наконец, хищнику может быть трудно сосредоточиться на одном животном, если оно входит в большую группу. Эти явления называются эффектом замешательства хищника [Milinski и Heller, 1978]. В публикации [Heinrich, 2002] дается интересное описание того, как антилопы используют эффект замешательства хищника.

Еще один способ, которым группы защищаются от хищников, описывается гипотезой «многих глаз» [Lima, 1995]. Когда большая группа до-

бывает пищу или находится на водопое, случайные эффекты диктуют, что всегда будет несколько животных, которые следят за хищниками. Это сотрудничество не только обеспечивает большую защиту от хищников, но и дает каждой особи больше времени для кормления и питья.

Наконец, группы защищаются от хищников за счет эффекта разбавления контакта [Krause и Ruxton, 2002]. Он может принимать различные формы. Во-первых, отдельные животные могут искать укрытия и защиту у группы в качестве особого типа эгоистичного поведения, для того чтобы уменьшить вероятность их привязанности [Hamilton, 1971]. Во-вторых, когда хищник бродит по своей территории, он с меньшей вероятностью может столкнуться с одной группой, чем с одной из многих особей, рассеянных по всей территории [Turner и Pitcher, 1986].

Животные также имеют больше успеха в поиске пищи, когда они находятся в группах, чем когда они одни. Возможно, на первый взгляд это покажется неправильным. В конце концов, когда особь находится в группе, она не может незаметно подойти к своей добыче; и когда она ловит свою добычу, она должна делиться пищей с другими членами группы. Тем не менее успех групп в кормежке связан с гипотезой «многих глаз» в избегании хищников. При наличии большого числа ищущих пищу глаз группа имеет непропорционально большие шансы на успех, чем одно животное, которое ищет пищу само по себе [Pitcher и Parrish, 1993]. Кроме того, группа увеличивает свои шансы на успех, если она может окружить свою добычу.

Животные также могут двигаться быстрее в группах, чем в одиночку. Это замечено у велосипедистов, которые едут, выстроившись в линию, и подменяют друг друга. Из-за сопротивления ветра идущие следом велосипедисты могут тратить на целых 40 % меньше энергии, чем ведущий велосипедист [Burke, 2003]. Такой же эффект, хотя и в меньшей степени, можно наблюдать в конькобежном спорте, беге, плавании и других видах спорта. В животном мире подмену можно увидеть в стаях гусей, когда они находятся в полете [McNab, 2002], стаях уток, когда они гребут по воде [Fish, 1995], и косяках рыб, когда они движутся [Noren et al, 2008].

Оптимизация на основе роя частиц (particle swarm optimization, PSO) опирается на наблюдение, что группы особей работают вместе, для того чтобы улучшить не только их коллективную результативность в какой-то задаче, но и результативность каждой особи. Принципы оптимизации на основе роя частиц четко наблюдаются не только в животном, но и в человеческом поведении. По мере того как мы пытаемся

улучшить нашу результативность в какой-то задаче, мы корректируем наш подход на основе некоторых основных идей.

- Инерция. Мы склонны придерживаться старых способов, которые оказались успешными в прошлом. «Я всегда так делал и буду продолжать так делать».
- Влияние общества. Мы слышим о других добившихся успеха людях и стараемся подражать их подходам. Мы можем прочитать об успехе других людей в книгах, интернете или газете. «Если это сработало для них, то, возможно, это сработает и для меня».
- Влияние соседей. Мы больше всего учимся у тех, кто лично нам близок. На нас больше влияют друзья, чем общество. В наших беседах с другими людьми мы делимся историями успеха и неудачи, и в результате этих разговоров мы меняем наше поведение. Инвестиционный совет от нашего соседа-миллионера или двоюродного брата будет иметь на нас более сильное влияние, чем отдаленные истории миллиардеров, которые мы читаем в интернете.

Краткий обзор главы

В разделе 11.1 приводится общий обзор оптимизации на основе роя частиц (particle swarm optimization, PSO) и некоторые простые его примеры. В разделе 11.2 рассматриваются способы ограничения скорости частиц в оптимизации на основе роя частиц, которое требуется для хорошей оптимизации. В разделе 11.3 обсуждаются коэффициенты инерционного взвешивания и инерционного сужения, двух признаков оптимизации на основе роя частиц, косвенно ограничивающих скорости частиц. В разделе 11.4 анализируется глобальный алгоритм оптимизации на основе роя частиц, являющийся обобщением данной оптимизации, в котором в каждом поколении для обновления скорости каждой особи используется лучшая особь. В разделе 11.5 обсуждается алгоритм оптимизации на основе полноинформированного роя частиц, в котором в каждом поколении скорость каждой особи вносит вклад в скорость каждой другой особи. Раздел 11.6 подходит к теме самообучения с использованием оптимизации на основе роя частиц с другого направления – если мы можем учиться на успехах других, то мы также можем учиться на их ошибках.

11.1. Базовый алгоритм оптимизации на основе роя частиц

Предположим, что у нас есть минимизационная задача, определенная на непрерывной области из d размерностей. У нас также есть популяция из N кандидатных решений, обозначенных как $\{x_i\}$, $i \in [1, N]$. Более того, предположим, что каждая особь x_i движется с некоторой скоростью v_i по поисковому пространству. Это движение по поисковому пространству является сутью оптимизации на основе роя частиц, и в этом заключается фундаментальное различие между оптимизацией на основе роя частиц и другими эволюционными алгоритмами. Большинство других эволюционных алгоритмов более статично, чем оптимизация на основе роя частиц, потому что они моделируют кандидатные решения и их эволюцию из поколения в поколение, но не моделируют динамику движения кандидатных решений по поисковому пространству.

Когда особь оптимизации на основе роя частиц движется по поисковому пространству, она имеет некоторую инерцию, и поэтому она склонна поддерживать свою скорость. Однако ее скорость может изменяться вследствие двух разных факторов:

- во-первых, она помнит свое лучшее положение в прошлом, и она хотела бы изменить свою скорость, для того чтобы вернуться в это положение. Это похоже на склонность человека помнить старые добрые времена и пытаться вернуть прошлое. В оптимизации на основе роя частиц особь перемещается по поисковому пространству, и ее положение в поисковом пространстве изменяется из поколения в поколение. Однако особь помнит свою результативность из прошлых поколений, и она помнит положение в поисковом пространстве, в котором получала свою лучшую результативность в прошлом;
- во-вторых, особь знает лучшее положение своих соседей в текущем поколении. Для этого требуется определить размер окрестности и нужно, чтобы все соседи сообщали друг другу о своей результативности в оптимизационной задаче.

Эти два эффекта случайно влияют на скорость особи и похожи на наши собственные социальные взаимодействия. Иногда мы чувствуем себя упрямее, чем в другие разы, и поэтому на нас не сильно влияют наши соседи. Иногда мы испытываем больше ностальгии, чем в другие разы, и поэтому на нас сильнее влияют наши прошлые успехи. На

рис. 11.1 мы схематично формулируем базовый алгоритм оптимизации на основе роя частиц (PSO).

Инициализировать случайную популяцию особей $\{x_i\}$, $i \in [1, N]$.
Инициализировать n-элементный скоростной вектор каждой особи v_i, $i \in [1, N]$.
Инициализировать лучшее к настоящему моменту положение каждой особи:
$b_i \leftarrow x_i$, $i \in [1, N]$
Задать размер окрестности $\sigma < N$.
Задать значения максимального влияния $\phi_{1,\max}$ и $\phi_{2,\max}$.
Задать максимальную скорость $v_{\max}$.
While not (критерий останова)
  For each особь x_i, $i \in [1, N]$
    $H_i \leftarrow \{\sigma$ ближайших соседей для $x_i\}$
    $h_i \leftarrow \arg\min_x \{f(x) : x \in H_i\}$
    Сгенерировать случайный вектор ϕ_1, где $\phi_1(k) \sim U[0, \phi_{1,\max}]$ для $k \in [1, n]$.
    Сгенерировать случайный вектор ϕ_2, где $\phi_2(k) \sim U[0, \phi_{2,\max}]$ для $k \in [1, n]$.
    $v_i \leftarrow v_i + \phi_1 \circ (b_i - x_i) + \phi_2 \circ (h_i - x_i)$
    If $|v_i| > v_{\max}$ **then**
      $v_i \leftarrow v_i v_{\max} / |v_i|$
    End if
    $x_i \leftarrow x_i + v_i$
    $b_i \leftarrow \arg\min_x \{f(x_i) : f(b_i)\}$
  Next особь
Next поколение

Рис. 11.1. Базовый алгоритм оптимизации на основе роя частиц (PSO) для минимизации n-размерной функции $f(x)$, где x_i – это i-е кандидатное решение и v_i – вектор его скорости. Запись $a \circ b$ означает поэлементное умножение векторов a и b

Рисунок 11.1 показывает, что в алгоритме оптимизации на основе роя частиц есть несколько регулировочных параметров.

- Мы не только должны инициализировать популяцию, как и в любом другом эволюционном алгоритме, но и должны инициализировать скоростные векторы популяции. Существует несколько способов инициализации скоростей. Например, они могут быть инициализированы случайными значениями, либо они могут быть инициализированы нулями [Helwig и Wanka, 2008].
- Мы должны определить размер окрестности σ алгоритма. Обратите внимание, что термин «размер окрестности» неоднозначен.

Иногда он означает, что каждая особь имеет σ близких соседей, а иногда означает, что поскольку в окрестности в общей сложности имеется σ особей, каждая особь имеет (σ – 1) близких соседей. В одной из первоначальных работ по оптимизации на основе роя частиц показано, что меньшие окрестности (только две особи) обеспечивают лучшее глобальное поведение и избегают локальных минимумов, в то время как большие окрестности обеспечивают более быстрое схождение [Eberhart и Kennedy, 1995].

- Мы должны выбрать максимальные скорости самообучения $\phi_{1,\max}$ и $\phi_{2,\max}$. Параметры ϕ_1, который называется скоростью познавательного самообучения, и ϕ_2, который называется скоростью социального самообучения, представляют собой несколько случайных чисел, распределенных соответственно в $[0, \phi_{1,\max}]$ и $[0, \phi_{2,\max}]$. Мы обсудим их далее в разделе 11.3.3, но пока просто отметим эмпирическое правило, что $\phi_{1,\max}$ и $\phi_{2,\max}$ часто принимаются равными примерно 2.05.
- Мы должны выбрать максимальную скорость $v_{\max}$. Эмпирические данные свидетельствуют о том, что каждый элемент $v_{\max}$ должен быть ограничен соответствующим динамическим диапазоном поискового пространства [Eberhart и Shi, 2000]. Это выглядит интуитивно понятным; если бы $v_{\max}$ была больше динамического диапазона поискового пространства, то частица может легко покинуть поисковое пространство в одном поколении. Другие результаты показывают, что $v_{\max}$ составляет от 10 % до 20 % диапазона поискового пространства [Eberhart и Shi, 2001]. В некоторых задачах мы не имеем в виду диапазон поискового пространства; то есть мы заранее не имеем представления о том, где находится тот оптимальный вариант, который мы ищем. В этом случае для лучшей результативности мы все равно должны применять конечную максимальную скорость $v_{\max}$ [Carlisle и Dozier, 2001].
- Мы можем упростить обновление скорости на рис. 11.1 следующим образом:

$$v_i \leftarrow v_i + \phi_1(b_i - x_i) + \phi_2(h_i - x_i), \tag{11.1}$$

где ϕ_1 и ϕ_2 – это скаляры, а не векторы, при этом $\phi_1 \sim U[0, \phi_{1,\max}]$ и $\phi_2 \sim U[0, \phi_{2,\max}]$. В этом варианте, именуемом линейной оптимизацией на основе роя частиц [Paquet и Engelbrecht, 2003], каждый элемент скоростного вектора v_i обновляется с теми же значения-

ми ϕ_1 и ϕ_2. Однако считается, что линейная оптимизация на основе роя частиц в целом обеспечивает худшую результативность, чем стандартный алгоритм рис. 11.1.

- Как и в большинстве других эволюционных алгоритмов, результативность оптимизации на основе роя частиц часто повышается за счет элитарности. Мы не показали элитарность на рис. 11.1, но можем легко реализовать ее, как описано в разделе 8.4.
- Обновление уравнения $x_i \leftarrow x_i + v_i$ на рис. 11.1 может привести к выходу x_i за пределы поисковой области. Мы обычно реализуем некоторый тип операции ограничения, с тем чтобы сохранить x_i в поисковой области. Например, после уравнения обновления мы можем включить следующие два уравнения:

$$x_i \leftarrow \min(x_i, x_{\max})$$
$$x_i \leftarrow \max(x_i, x_{\min}), \qquad (11.2)$$

где $[x_{\min}, x_{\max}]$ определяет границы поисковой области.

Топологии роя частиц

На рис. 11.1 показано, что каждая частица находится под влиянием своих ближайших соседей. Расположение соседей, влияющих на частицу, называется *роевой* топологией. Поскольку окрестности каждой частицы на рис. 11.1 изменяются каждое поколение, такое расположение называется динамической топологией. Так как окрестности являются локальными (то есть не включают весь рой целиком), оно также называется *локально лучшей* топологией (lbest topology).

Для определения окрестности каждой частицы можно применить ряд других методов [Akat и Gazi, 2008]. Например, мы можем определить окрестности в начале алгоритма, с тем чтобы окрестности были статичными и не менялись из поколения в поколение. Либо если процесс оптимизации застаивается, то мы можем в это время случайно переопределить окрестности [Clerc и Poli, 2006]. В крайнем случае, мы можем иметь единую окрестность, которая охватывает весь рой целиком, то есть значение H_i на рис. 11.1 будет равно всему рою целиком для всех i, и h_i не зависит от i и равно лучшей частице среди всей популяции. Это называется *совокупной* топологией (all topology) или *глобально лучшей* топологией (gbest topology). Именно с этой топологией оптимизация на основе роя частиц первоначально и разрабатывалась, и она по-прежнему широко применяется. Еще одной распространен-

ной топологией является *кольцевая* топология (ring topology), в которой каждая частица соединена с двумя другими частицами. *Кластерная* топология (cluster topology) – это топология, в которой каждая частица полносвязна внутри своего кластера, в то время как несколько частиц в каждом кластере также связано с дополнительной частицей в другом кластере. *Колесная* топология (wheel topology) – это топология, в которой фокальная частица соединена со всеми другими частицами, в то время как все остальные частицы соединены только с фокальной частицей. *Квадратная* топология (square topology), так называемая топология фон Неймана, – это топология, в которой каждая частица связана с четырьмя соседями. На рис. 11.2 показаны некоторые из этих топологий. Результативность оптимизации на основе роя частиц может сильно варьироваться в зависимости от топологии, и исследователи экспериментировали со многими другими топологиями, кроме тех немногих, которые мы здесь упоминаем [Mendes и соавт., 2004], [del Valle и соавт., 2008].

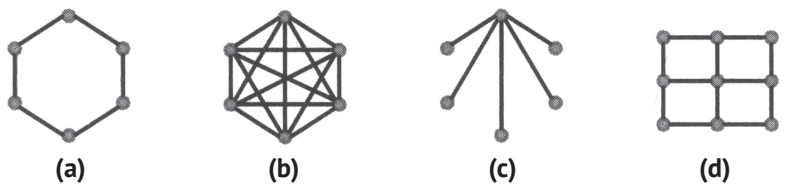


Рис. 11.2. Несколько топологий оптимизации на основе роя частиц: (a) представляет собой *кольцевую* топологию; (b) представляет *совокупную* топологию; (c) представляет собой *колесную* топологию; (d) представляет собой *квадратную* топологию. Квадратная топология циклически переносится сверху вниз и слева направо, поэтому с каждой частицей, связанной с четырьмя соседями, она образует тороид. Каждая из этих топологий может быть статической или динамической

11.2. Ограничение скорости

Во многих приложениях оптимизации на основе роя частиц было обнаружено, что если v_{max} не используется, частицы алгоритма оптимизации на основе роя частиц дико скачут по поисковому пространству [Eberhart и Kennedy, 1995]. Для того чтобы понять, почему это происходит, рассмотрим базовый алгоритм оптимизации на основе роя частиц на рис. 11.1, но с упрощением, что $\phi_2 = 0$. Обновления положения и скорости в этом случае равны

$$\begin{aligned} v_i(t+1) &= v_i(t) + \phi_1(b_i - x_i) \\ x_i(t+1) &= x_i(t) + v_i(t+1), \end{aligned} \tag{11.3}$$

где t – это номер поколения, а ϕ_1 – скорость познавательного самообучения. Это можно записать в виде:

$$\begin{bmatrix} x_i(t+1) \\ v_i(t+1) \end{bmatrix} = \begin{bmatrix} 1-\phi_1 & 1 \\ -\phi_1 & 1 \end{bmatrix} \begin{bmatrix} x_i(t) \\ v_i(t) \end{bmatrix} + \begin{bmatrix} \phi_1 \\ \phi_1 \end{bmatrix} b_i. \tag{11.4}$$

Собственные значения матрицы в правой части приведенного выше уравнения равны

$$\lambda = \frac{2 - \phi_1 \pm \sqrt{\phi_1^2 - 4\phi_1}}{2}, \tag{11.5}$$

и эти собственные значения определяют стабильность системы[1]. Если $\phi_1 \in [0, 4]$, то оба собственных значения имеют величину 1, а это означает, что система минимально стабильная и что $x_i(t)$ и $v_i(t)$ могут стать неограниченными, по мере того как $t \to \infty$, в зависимости от исходных условий. Если $\phi_1 > 4$, то одно из собственных значений больше 1 по величине, а значит, система нестабильна и $x_i(t)$ и $v_i(t)$ увеличиваются без ограничения практически для любых исходных условий. Этот простой пример показывает, почему для ограничения величины v_i, как показано на рис. 11.1, имеет важное значение использовать v_{max}.

Однако этот анализ исходит из того, что b_i не является функцией от x_i. Кроме того, реальные реализации оптимизации на основе роя частиц сложнее, чем уравнение (11.3), поэтому наш анализ может быть неверным для общих алгоритмов оптимизации на основе роя частиц. Если $\phi_2 > 0$ или если используемый инерционный вес меньше 1, как мы позже обсудим в уравнении (11.9), то для получения хорошей результативности может не потребоваться ограничивать скорость [Carlisle и Dozier, 2001], [Clerc и Kennedy, 2002].

Если мы хотим ограничить скорость, то можем ограничить ее несколькими способами. Один из способов – проверить величину v_i, и если она больше скалярного v_{max}, то прошкалировать компоненты v_i так, чтобы $|v_i| = v_{max}$:

$$\text{If } |v_i| > v_{max} \text{ then } v_i \leftarrow \frac{v_i v_{max}}{|v_i|}, \tag{11.6}$$

[1] Относительно обсуждения вопроса стабильности обратитесь к любой книге по линейным системам либо работе [Simon, 2006, глава 1].

как показано на рис. 11.1. Еще один способ – ограничить величину каждого компонента v_i. Напомним, что каждая особь в популяции имеет n размерностей, поэтому $v_i = [v_i(1) \ldots v_i(n)]$[2]. При таком подходе указывается максимальная скорость каждой размерности, поэтому $v_{max}(j)$ определена для $j \in [1, n]$. Ограничение скорости i-й частицы затем выполняется следующим образом:

$$v_i(j) \leftarrow \begin{cases} v_i(j) & \text{если } |v_i(j)| \le v_{max}(j) \\ v_{max}(j)\,\text{sign}\,(v_i(j)) & \text{если } |v_i(j)| > v_{max}(j) \end{cases} \quad \text{для } j \in [1, n] \qquad (11.7)$$

Ограничение скорости можно рассматривать как контроль над разведочно-эксплуатационным балансом оптимизации на основе роя частиц. Большее значение v_{max} допускает больше изменений в каждой особи из поколения в поколение, что подчеркивает разведывание. Малое значение v_{max} ограничивает изменения в особях, что подчеркивает эксплуатацию.

11.3. Коэффициенты инерционного взвешивания и сужения

Для того чтобы избежать ограничения скорости, мы можем изменить уравнение обновления скорости на рис. 11.1, предотвратив увеличение скорости без ограничения. В этом разделе мы сначала обсудим использование инерционного взвешивания в разделе 11.3.1. Затем обсудим эквивалентный, но более часто используемый коэффициент инерционного сужения 11.3.2. Наконец, в разделе 11.3.3 мы представим несколько условий стабильности алгоритма оптимизации на основе роя частиц.

11.3.1. *Инерционное взвешивание*

В приложениях оптимизации на основе роя частиц часто используется инерционный вес. Как видно из уравнения обновления скорости на рис. 11.1, частица стремится поддерживать свою скорость из поколения в поколение, хотя из-за скоростей самообучения допускаются некоторые изменения скорости:

$$v_i(k) \leftarrow v_i(k) + \phi_1(b_i - x_i(k)) + \phi_2(k)(h_i(k) - x_i(k)) \quad \text{для } k \in [1, n], \qquad (11.8)$$

[2] Обратите внимание, что j в члене $v_i(j)$ обозначает конкретный элемент вектора v_i, в то время как $v_i(t)$ в уравнении (11.3) обозначает значение v_i в t-м поколении. Это обозначение не является последовательным, но его смысл должен быть ясен из контекста.

где n – это размерность задачи. Однако опытным путем было установлено, что уменьшение инерции в процессе оптимизации может обеспечить более высокую результативность. Уравнение (11.8), следовательно, меняется на следующее уравнение:

$$v_i(k) \leftarrow wv_i(k) + \phi_1(k)(b_i(k) - x_i(k)) + \phi_2(h_i(k) - x_i(k)), \qquad (11.9)$$

где w – это инерционный вес, который часто уменьшается примерно с 0.9 в первом поколении до 0.4 в последнем поколении [Eberhart и Shi, 2000]. Это помогает замедлять скорость каждой частицы по мере увеличения числа поколений, что улучшает схождение.

В публикации [Clerc и Poli, 2006] рекомендуются параметры оптимизации на основе роя частиц для обновления скорости формы уравнения (11.9). В этой статье размер популяции принят равным 30, размер окрестности принят равным 4 и окрестности фиксированы до тех пор, пока процесс оптимизации на основе роя частиц не начнет застаиваться, и в это время окрестности случайно реинициализируются. Обновление скорости осуществляется, как показано в уравнении (11.9) со следующими рекомендуемыми параметрами [Clerc и Poli, 2006, уравнение 19]:

$$\begin{aligned} w &= 0.72 \\ \phi_1(k) &\sim U[0, 1.108] \quad \text{для } k \in [1, n] \\ \phi_2(k) &\sim U[0, 1.108] \quad \text{для } k \in [1, n] \end{aligned} \qquad (11.10)$$

В публикации [Clerc и Poli, 2006] также предлагаются другие вариации оптимизации на основе роя частиц для повышения результативности. Стабильность оптимизации на основе роя частиц с обновлением уравнения скорости (11.9) обсуждается в публикации [Poli, 2008].

Более обширный набор рекомендуемых параметров оптимизации на основе роя частиц можно найти для обновлений скорости формы уравнения (11.9) в публикации [Pedersen, 2010] и табл. 11.1. Эти рекомендуемые параметры применяются, когда окрестностью для каждой частицы является весь рой целиком, так что $h_i()$ (для всех i) в уравнении (11.9) равен лучшей особи в популяции.

11.3.2. *Коэффициент сужения*

Вместо уравнения (11.9) инерционное взвешивание часто реализуется вместе с коэффициентом сужения. Данная реализация выполняет то же самое, что и инерционное взвешивание, и включает в себя запись уравнения обновления скорости как

$$v_i \leftarrow K\left[v_i + \phi_1(b_i - x_i) + \phi_2(h_i - x_i)\right], \tag{11.11}$$

где K называется коэффициентом сужения [Clerc, 1999], [Eberhart и Shi, 2000], [Clerc и Kennedy, 2002]. Для упрощения нашего анализа мы использовали линейное обновление скорости в уравнении (11.11). Уравнение (11.11) эквивалентно линейной форме уравнения (11.9), если $K = w$ и если ϕ_1 и ϕ_2 в уравнении (11.11) заменены соответственно на ϕ_1/K и ϕ_2/K. Для анализа этого подхода мы используем t, который обозначает номер поколения, и записываем уравнение (11.11) следующим образом:

$$\begin{aligned} v_i(t+1) &= K\left[v_i(t) + (\phi_1 + \phi_2)\left(\frac{\phi_1 b_i(t) + \phi_2 h_i(t)}{\phi_1 + \phi_2} - x_i(t)\right)\right] \\ &= K\left[v_i(t) + \phi_T\left(p_i(t) - x_i(t)\right)\right], \end{aligned} \tag{11.12}$$

где ϕ_T и $p_i(t)$ определяются вышеприведенным уравнением. Теперь определим

$$y_i(t) = p_i(t) - x_i(t). \tag{11.13}$$

Таблица 11.1. Рекомендуемые параметры оптимизации на основе роя частиц (PSO) для разных размерностей задачи и оцениваний имеющихся функций приспособленности [Pedersen, 2010]. N – это размер популяции, а w, ϕ_1 и ϕ_2 – рекомендуемые параметры для уравнения (11.9), когда окрестности каждой частицы состоят из всего роя целиком. Из таблицы видно, что некоторые конфигурации задач имеют более одного рекомендуемого множества параметров, так как несколько множеств параметров дает практически одинаковую результативность на эталонах

Размерность задачи	Вычисления функций	N	w	ϕ_1	ϕ_2
2	400	25	0.3925	2.5586	1.3358
		29	–0.4349	–0.6504	2.2073
2	4,000	156	0.4091	2.1304	1.0575
		237	–0.2887	0.4862	2.5067
5	1,000	63	–0.3593	–0.7238	2.0289
		47	–0.1832	0.5287	3.1913
5	10,000	223	–0.3699	–0.1207	3.3657
		203	0.5069	2.5524	1.0056
10	2,000	63	0.6571	1.6319	0.6239
		204	–0.2134	–0.3344	2.3259

Размерность задачи	Вычисления функций	N	w	ϕ_1	ϕ_2
10	20,000	53	−0.3488	−0.2746	4.8976
20	40,000	69	−0.4438	−0.2699	3.3950
20	400,000	149	−0.3236	−0.1136	3.9789
		60	−0.4736	−0.9700	3.7904
		256	−0.3499	−0.0513	4.9087
30	600,000	95	−0.6031	−0.6485	2.6475
50	100,000	106	−0.2256	−0.1564	3.8876
100	200,000	161	−0.2089	−0.0787	3.7637

Принимая, что $p_i(t)$ постоянна во времени, мы можем объединить уравнения (11.12) и (11.13), чтобы получить

$$\begin{aligned} v_i(t+1) &= Kv_i(t) + K\phi_T y_i(t) \\ y_i(t+1) &= p_i - x_i(t+1) \\ &= p_i - x_i(t) - v_i(t+1) \\ &= y_i(t) - Kv_i(t) - K\phi_T y_i(t) \\ &= -Kv_i(t) + (1 - K\phi_T)y_i(t). \end{aligned} \tag{11.14}$$

Эти уравнения для $v_i(t+1)$ и $y_i(t+1)$ можно объединить, в результате чего мы получим

$$\begin{bmatrix} v_i(t+1) \\ y_i(t+1) \end{bmatrix} = \begin{bmatrix} K & K\phi_T \\ -K & 1-K\phi_T \end{bmatrix} \begin{bmatrix} v_i(t) \\ y_i(t) \end{bmatrix}. \tag{11.15}$$

Матрица в правой части приведенного выше уравнения, которая управляет стабильностью системы, имеет собственные значения

$$\begin{aligned} \lambda &= \frac{1}{2}\left[1 - K(\phi_T - 1) \pm \sqrt{1 + K^2(\phi_T - 1)^2 - 2K(\phi_T + 1)}\right] \\ &= \frac{1}{2}\left[1 - K(\phi_T - 1) \pm \sqrt{\Delta}\right], \end{aligned} \tag{11.16}$$

где дискриминант Δ определяется вышеприведенным уравнением. Мы обозначаем собственные значения как λ_1 и λ_2:

$$\begin{aligned} \lambda_1 &= \frac{1}{2}\left[1 - K(\phi_T - 1) + \sqrt{\Delta}\right]; \\ \lambda_2 &= \frac{1}{2}\left[1 - K(\phi_T - 1) - \sqrt{\Delta}\right]. \end{aligned} \tag{11.17}$$

Динамическая система уравнения (11.15) стабильна, если $|\lambda_1| < 1$ и $|\lambda_2| < 1$. Этот анализ предполагает, что ϕ_T постоянен. В обновлении скорости уравнения (11.11) члены ϕ_i случайны, но в этом анализе мы делаем упрощающее допущение, что каждый параметр ϕ_i постоянен. См. задачу 11.8 относительно обсуждения того, следует ли в оптимизации на основе роя частиц использовать константу K либо варьирующийся со временем K. В следующем разделе мы изучим поведение собственных значений λ_1 и λ_2, которое определяет стабильность алгоритма оптимизации на основе роя частиц как функции от коэффициента сужения K.

11.3.3. *Стабильность оптимизации на основе роя частиц*

Мы можем использовать уравнение (11.16), для того чтобы сделать следующее наблюдение.

Наблюдение 11.1. При $K = 0$ мы получаем $\Delta = 1$, $\lambda_1 = 1$ и $\lambda_2 = 0$.

Далее мы рассмотрим значения λ_1 и λ_2 по мере увеличения K от 0. Уравнение (11.16) показывает, что

$$\lim_{|K|\to\infty} \Delta = \infty$$

$$\Delta = 0 \quad \text{для} \quad K = \frac{\phi_T + 1 \pm 2\sqrt{\phi_T}}{(\phi_T - 1)^2} = \{K_1, K_2\}$$

$$\Delta < 0 \quad \text{для} \quad K \in (K_1, K_2) \tag{11.18}$$

где K_1 и K_2 определяются вышеприведенным уравнением. Предположим, что $\phi_T > 0$, благодаря чему $\sqrt{\phi_T}$ – вещественное. На рис. 11.3 показан график Δ как функции от K. Это приводит нас к следующему далее наблюдению.

Наблюдение 11.2. λ_1 и λ_2 вещественны для $K < K_1$.

Когда $K = K_1$, мы видим, что $\Delta = 0$, а значит, $\lambda_1 = \lambda_2$. На самом деле из уравнения (11.16) легко сделать следующее далее наблюдение.

Наблюдение 11.3. При $K = K_1$ мы получаем $\lambda_1 = \lambda_2 = (1 - \sqrt{\phi_T})/(1 - \phi_T)$, которое находится между 0 и 1 для всех $\phi_T \neq 1$.

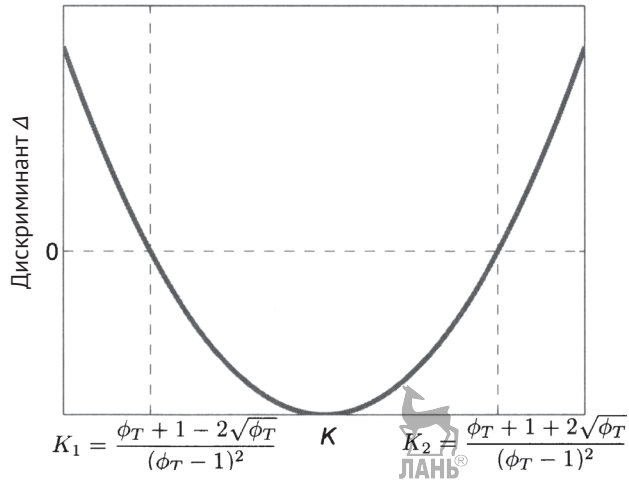


Рис. 11.3. Дискриминант Δ уравнения (11.16) как функция от коэффициента сужения K. $\Delta > 0$ для $K < K_1$ и $K > K_2$, а значит, λ_1 и λ_2 – вещественные. $\Delta < 0$ для $K \in (K_1, K_2)$, а значит, λ_1 и λ_2 – комплексные

Теперь рассмотрим поведение λ_1, по мере того как K увеличивается от 0 до K_1. Взяв производную от λ_1 по K, мы видим, что

$$\frac{d\lambda_1}{dK} = \frac{1-\phi_T}{2} - \frac{\phi_T - K(\phi_T - 1)^2 + 1}{\sqrt{K^2(\phi_T - 1)^2 - 2K(\phi_T + 1) + 1}}. \tag{11.19}$$

Мы можем применить элементарные алгебраические преобразования, для того чтобы показать, что если $\phi_T > 1$, то эта производная отрицательна для $K < K_1$. Аналогично мы можем показать, что производная λ_2 по K положительна для $K < K_1$. Это подводит нас к следующему далее наблюдению.

Наблюдение 11.4. Если $\phi_T > 1$, то λ_1 и λ_2 находятся в диапазоне от 0 до 1 для $K \in (0, K_1)$.

Теперь рассмотрим поведение λ_1 и λ_2, по мере того как K увеличивается от K_1. По рис. 11.3 мы видим, что когда $K \in (K_1, K_2)$, λ_1 и λ_2 являются комплексными с той же величиной, которая может быть выведена как

$$|\lambda| = \frac{1}{2}\sqrt{\left[1 - K(\phi_T - 1)\right]^2 - 2K(\phi_T + 1) - 1 - K^2(\phi_T - 1)^2}. \tag{11.20}$$

После некоторых алгебраических преобразований эта формула сводится к $|\lambda| = \sqrt{K}$. Производная этого выражения положительна для всех $K > 0$. Это подводит нас к следующему далее наблюдению.

Наблюдение 11.5. При $K \in (K_1, K_2)$ λ_1 и λ_2 являются комплексными и имеют одинаковую величину, которая монотонно увеличивается вместе с K.

Теперь рассмотрим значения λ_1 и λ_2 при $K = K_2$. По рис. 11.3 известно, что λ_1 и λ_2 являются вещественными и равны при $K = K_2$. На самом деле из уравнения (11.16) легко увидеть, что при $K = K_2$, $\lambda_1 = \lambda_2 = (1 + \sqrt{\phi_T})/(1 - \phi_T)$, находящееся между 0 и –1 для всех $\phi_T > 4$, которые мы сформулируем следующим образом.

Наблюдение 11.6. При $K = K_2$ мы получаем $\lambda_1 = \lambda_2 = (1 + \sqrt{\phi_T})/(1 - \phi_T)$, находящееся между 0 и –1, если $\phi_T > 4$.

Теперь рассмотрим значения λ_1 и λ_2 при $K > K_2$. λ_1 и λ_2 вещественны для этого диапазона K. Уравнение (11.19) дает производную от λ_1 по K, когда λ_1 вещественно. Мы можем выполнить некоторые элементарные алгебраические преобразования уравнения (11.19), для того чтобы показать, что если $\phi_T > 1$ и $K > K_2$, то производная от λ_1 положительна и производная от λ_2 отрицательна. Объединив это рассуждение с наблюдением 11.6, мы видим, что λ_1 остается меньше 1 по величине для всех значений $K > K_2$. Однако λ_2 приближается к $-\infty$, по мере того как $K \to \infty$, как видно из уравнения (11.17). Это дает нам следующее далее наблюдение.

Наблюдение 11.7. При $K > K_2$ λ_1 является вещественным и отрицательным и меньше 1 по величине, и λ_2 является вещественным и отрицательным и приближается к $-\infty$, по мере того как $K \to \infty$.

Предел λ_1 как $K \to \infty$ можно вывести из уравнения (11.16):

$$\begin{aligned}
\lambda_1 &= \frac{1}{2}\left[1 - K(\phi_T - 1) + \sqrt{1 + K^2(\phi_T - 1)^2 - 2K(\phi_T + 1)}\right] \\
&= \frac{1/K - (\phi_T - 1) + \sqrt{1/K^2 + (\phi_T - 1)^2 - 2(\phi_T + 1)/K}}{2} \\
&= \frac{N(K)}{D(K)}, \qquad (11.21)
\end{aligned}$$

где числитель $N(K)$ и знаменатель $D(K)$ определяются вышеприведенным уравнением. Предел $N(K)$ и $D(K)$ равен 0, по мере того как $K \to \infty$, поэтому для вычисления предела мы можем применить правило Лопиталя следующим образом[3]:

$$\begin{aligned}
\frac{dN(K)}{dK} &= -K^{-2} + \frac{-2K^{-3} + 2(\phi_T + 1)K^{-2}}{2\sqrt{K^{-2} + (\phi_T - 1)^2 - 2(\phi_T + 1)K^{-1}}} \\
\frac{dD(K)}{dK} &= -2K^{-2} \\
\frac{dN(K)/dK}{dD(K)/dK} &= \frac{1}{2} + \frac{K^{-1} - \phi_T - 1}{2\sqrt{K^{-2} + (\phi_T - 1)^2 - 2(\phi_T + 1)K^{-1}}} \\
\lim_{K\to\infty} \lambda_1 &= \lim_{K\to\infty} \frac{dN(K)/dK}{dD(K)/dK} \\
&= \frac{1}{2} - \frac{\phi_T + 1}{2(\phi_T - 1)} \\
&= \frac{1}{1 - \phi_T}
\end{aligned} \tag{11.22}$$

который меньше единицы по величине, если $\phi_T > 2$. Это приводит нас к следующему далее наблюдению, которое представляет собой расширение наблюдений 11.6 и 11.7.

Наблюдение 11.8. По мере того как K увеличивается от K_2 до ∞, λ_1 монотонно возрастает от $(1 + \sqrt{\phi_T}) / (1 - \phi_T)$ до $1 / (1 - \phi_T)$ и λ_2 монотонно уменьшается от $(1 + \sqrt{\phi_T})/(1 - \phi_T)$ до $-\infty$.

Поскольку λ_2 уменьшается от $(1 + \sqrt{\phi_T}) / (1 - \phi_T)$, что больше -1, до $-\infty$, λ_2 должно быть равно -1 при некотором значении K, которое мы обозначаем как K_3. Поэтому из уравнения (11.17) мы имеем

$$-1 = \frac{1}{2}\left[1 - K_3(\phi_T - 1) - \sqrt{1 + K^2(\phi_T - 1)^2 - 2K(\phi_T + 1)}\right]. \tag{11.23}$$

Решение этого уравнения для K_3 дает $K_3 = 2 / (\phi_T - 2)$. Объединение его со всеми вышеперечисленными наблюдениями дает нам следующую теорему.

[3] Спасибо Стиву Шатмары (Steve Szatmary) за выведение $\lim_{K\to\infty}\lambda_1$.

Теорема 11.1. Если b_i и h_i постоянны в обновлении скорости уравнения (11.11) и если $\phi_T = \phi_1 + \phi_2 > 4$, то оптимизация на основе роя частиц стабильна для

$$K < \frac{2}{\phi_T - 2}. \tag{11.24}$$

На рис. 11.4 показано, как при увеличении K от 0 до ∞ изменяются собственные значения уравнения (11.15) в комплексной плоскости. На рис. 11.5 показано, как их величины изменяются вместе с K.

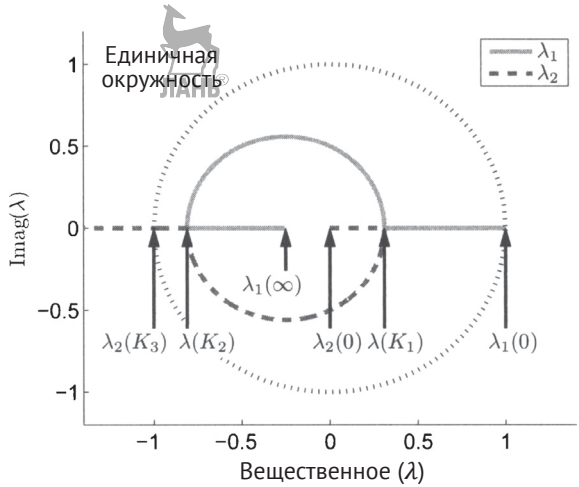


Рис. 11.4. Собственные значения уравнения (11.15), по мере того как коэффициент сужения K варьируется от 0 до ∞ проиллюстрированы для случая $\phi_T = 5$. При $K = 0$, $\lambda_1 = 1$ и $\lambda_2 = 0$. При $K = K_1$, $\lambda_1 = \lambda_2 > 0$. Для $K \in (K_1, K_2)$, λ_1 и λ_2 являются комплексными. При $K = K_2$, $\lambda_1 = \lambda_2 < 0$. При $K = K_3$, $\lambda_2 = -1$. По мере того как $K \to \infty$, $\lambda_1 \to 1/(1 - \phi_T)$ и $\lambda_2 \to -\infty$

Мы можем написать K для стабильного алгоритма оптимизации на основе роя частиц в виде

$$K = \frac{2\alpha}{\phi_T - 2}, \quad \text{где } \phi_T = \phi_{1,\max} + \phi_{2,\max}, \tag{11.25}$$

где $\alpha \in (0,1)$, так что α указывает, насколько близок коэффициент сужения K к его теоретически максимальному значению до того, как алгоритм оптимизации на основе роя частиц станет нестабильным. Более высокое значение α допускает больше разведывания, в то время как более низкое значение α подчеркивает больше эксплуатации.

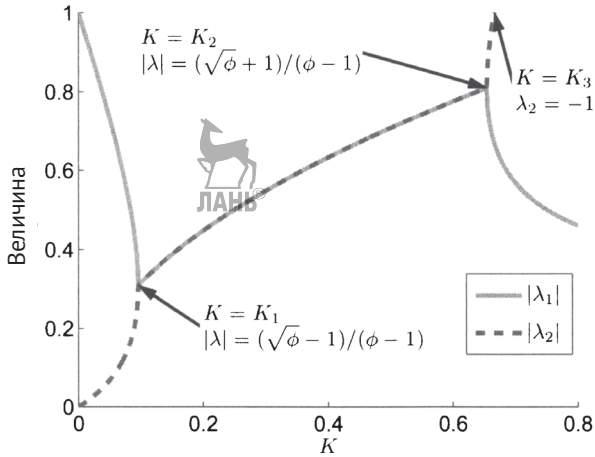


Рис. 11.5. Величины собственных значений уравнения (11.15) по мере увеличения коэффициента сужения *K* от 0, проиллюстрированные для случая $\phi_T = 5$. Этот график показывает величины собственных значений, которые проиллюстрированы на рис. 11.4

Алгоритмы оптимизации на основе роя частиц нередко представлены в учебниках и научно-исследовательских публикациях [Carlisle и Dozier, 2001], [Clerc и Kennedy, 2002], [Eberhart и Shi, 2000], [Poli и соавт. 2007] со следующей ниже рекомендацией:

Общая рекоммендация: $\phi_T > 4$

$$K > \frac{2}{\phi_T - 2 + \sqrt{\phi_T(\phi_T - 4)}}. \qquad (11.26)$$

Она эквивалентна теореме 11.1, по мере того как $\phi_T \to 4$, но теорема 11.1 является более общей для $\phi_T > 4$. Уравнение (11.26) не дает каких-либо указаний на верхний предел для ϕ_T либо для распределения ϕ_T между $\phi_{1,\max}$ и $\phi_{2,\max}$. Часто рекомендуется принимать ϕ_T равным чуть больше 4 и размещать ϕ_T примерно одинаково между $\phi_{1,\max}$ и $\phi_{2,\max}$. Например, $\phi_{1,\max} = \phi_{2,\max} = 2.05$. Однако эмпирические результаты показывают, что существует несколько оптимизационных задач, для которых можно получить более высокую результативность оптимизации на основе роя частиц для значений ϕ_T, которые намного больше, чем 4.1, и для значений $\phi_{1,\max}$ и $\phi_{2,\max}$, которые находятся далеко друг от друга [Carlisle и Dozier, 2001]. Также обратите внимание, что наш анализ использует всего один конкретный подход, но иные подходы с другими допущениями приводят к разным условиям стабильности [Clerc и Poli, 2006].

11.4. Глобальные обновления скорости

Один из способов, которым мы можем обобщить обновление скорости уравнения (11.11), состоит в том, чтобы написать

$$v_i \leftarrow K\left[v_i + \phi_1(b_i - x_i) + \phi_2(h_i - x_i) + \phi_3(g - x_i)\right], \tag{11.27}$$

где g – это лучшая особь, найденная с начала первого поколения. Анализ предыдущего раздела справедлив для уравнения (11.27), если мы определим $\phi_T = \phi_{1,\max} + \phi_{2,\max} + \phi_{3,\max}$ и если мы примем, что $b_i + h_i + g$ постоянно во времени. Новый член $\phi_3(g - x_i)$ добавляет член в уравнение обновления скорости, который стремится привести каждую частицу к лучшей особи, найденной к настоящему моменту с начала первого поколения. Это концептуально похоже на племенной эволюционный алгоритм (stud EA), в котором для каждой операции рекомбинации используется лучшая особь каждом поколении (раздел 8.7.7). Разница лишь в том, что g в уравнении (11.27) является лучшей особью, найденной с начала первого поколения, в то время как «племенной жеребец» (stud) в разделе 8.7.7 является лучшей особью в текущем поколении. Это сходство и различие может мотивировать на использование более g-подобной операции в племенном эволюционном алгоритме или на использование более stud-подобной операции в алгоритме глобальной оптимизации на основе роя частиц.

■ Пример 11.1

В данном примере мы используем оптимизацию на основе роя частиц с общим обновлением скорости уравнения (11.27) для оптимизации 20-мерной функции Экли. Мы применяем популяцию размером 50, параметр элитарности 2 и размер окрестности $\sigma = 4$. Мы используем номинальные значения

$$\begin{aligned} \phi_{1,\max} &= \phi_{2,\max} = \phi_{3,\max} = 2.1 \\ \phi_T &= \phi_{1,\max} + \phi_{2,\max} + \phi_{3,\max} \\ K &= \frac{2\alpha}{\phi_T - 2}, \quad \alpha = 0.9 \end{aligned} \tag{11.28}$$

Обратите внимание, что мы можем альтернативно решить для ϕ_T с точки зрения K:

$$\phi_T = \frac{2(\alpha + K)}{K}. \tag{11.29}$$

Рисунки 11.6–11.9 показывают среднюю результативность оптимизации на основе роя частиц при разных значениях $\phi_{1,\max}$, $\phi_{2,\max}$, $\phi_{3,\max}$ и α, когда остальные параметры равны их номинальным значениям. Мы видим, что номинальные значения уравнения (11.28) действительно приближенно оптимальны для 20-мерной функции Экли.

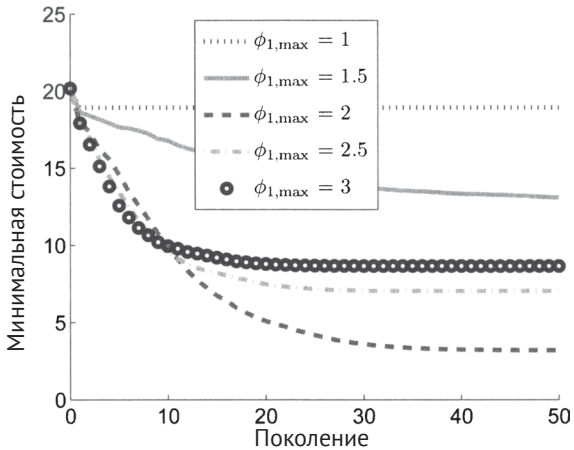


Рис.11.6. Пример 11.1: результативность оптимизации на основе роя частиц на 20-мерной функции Экли для разных значений $\phi_{1,\max}$, усредненно по 20 симуляциям Монте-Карло. $\phi_{1,\max} = 2$ является приближенно оптимальным для этой эталонной функции

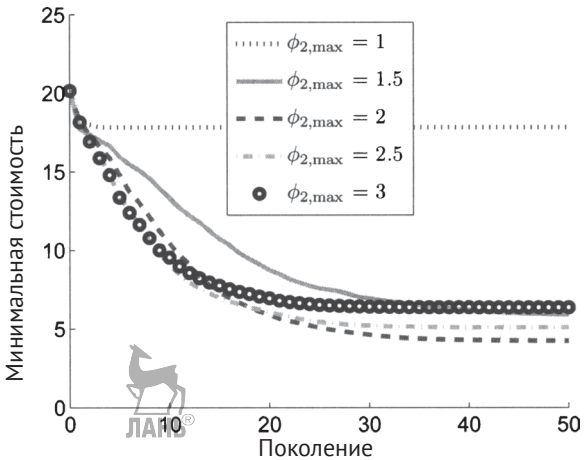


Рис. 11.7. Пример 11.1: результативность оптимизации на основе роя частиц на 20-мерной функции Экли для разных значений $\phi_{2,\max}$, усредненно по 20 симуляциям Монте-Карло. $\phi_{2,\max} = 2$ является приближенно оптимальным для этой эталонной функции

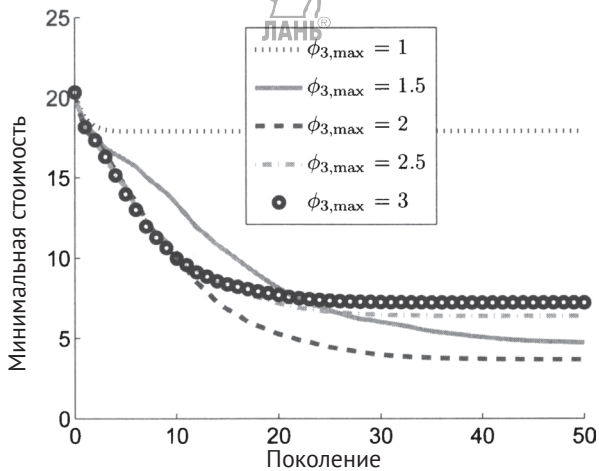


Рис. 11.8. Пример 11.1: результативность оптимизации на основе роя частиц на 20-мерной функции Экли для разных значений $\phi_{3,\max}$, усредненно по 20 симуляциям Монте-Карло. $\phi_{3,\max} = 2$ является приближенно оптимальным для этой эталонной функции

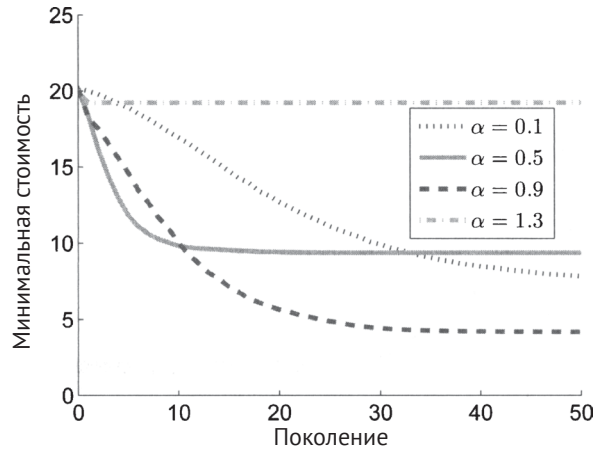


Рис. 11.9. Пример 11.1: результативность оптимизации на основе роя частиц на 20-мерной функции Экли для разных значений коэффициента сужения $K = \alpha K_{\max}$, усредненно по 20 симуляциям Монте-Карло. $K = 0.9K_{\max}$ является приближенно оптимальным для этой эталонной функции

Рисунки 11.6–11.8 показывают, что когда значения $\phi_{\max}$ слишком малы, частицы блуждают неориентированным образом. Когда они слишком велики, частицы чрезмерно ограничены и не могут эффективно разведывать поисковое пространство.

Рисунок 11.9 показывает, что когда параметр α (и, следовательно, K) слишком мал, частицы застаиваются из-за малых скоростей. Когда α (и, следовательно, K) слишком велик, частицы скачут слишком агрессивно по поисковому пространству.

■

11.5. Полноинформированный рой частиц

Уравнения (11.12) и (11.27) показывают, что наша самая общая (к этому моменту) форма обновления скорости имеет вид:

$$\begin{aligned} v_i(t+1) &= K\left[v_i(t) + \phi_T(p_i(t) - x_i(t))\right] \\ \phi_T &= \phi_{1,\max} + \phi_{2,\max} + \phi_{3,\max} \\ p_i(t) &= \frac{\phi_1 b_i(t) + \phi_2 h_i(t) + \phi_3 g(t)}{\phi_1 + \phi_2 + \phi_3} \end{aligned} \tag{11.30}$$

Мы видим, что три положения частиц вносят вклад в обновление скорости: лучшее к настоящему моменту положение текущей особи $b_j(t)$, лучшее к настоящему моменту текущее положение окрестности $h_j(t)$ и лучшее к настоящему моменту положение популяции $g(t)$. Это приводит к идее сделать обновление скорости более общим. Почему бы не позволить каждой особи в популяции вносить свой вклад в обновление скорости? Обобщение уравнения (11.30) можно записать в виде:

$$\begin{aligned} v_i(t+1) &= K\left[v_i(t) + \phi_T(p_i(t) - x_i(t))\right] \\ \phi_T &= \frac{1}{N}\sum_{j=1}^{N} \phi_{j,\max} \\ p_i(t) &= \frac{\sum_{j=1}^{N} w_{ij}\phi_j b_j(t)}{\sum_{j=1}^{N} w_{ij}\phi_j} \end{aligned} \tag{11.31}$$

где $b_j(t)$ – это лучшее решение, найденное к настоящему моменту j-й частицей:

$$b_j(t) = \arg\min_x f(x) : x \in \{x_j(0), \ldots, x_j(t)\}. \tag{11.32}$$

Обратите внимание на множитель $1/N$ в определении ϕ_T в уравнении (11.31), который представляет собой ситуативный (ad-hoc) подход к поддержанию разумного баланса между вкладом $v_i(t)$ и $(p_i(t) - x_i(t))$ в новую скорость $v_i(t+1)$. Параметры ϕ_j в уравнении (11.31) являются

факторами случайного воздействия, взятыми из равномерного распределения $U[0, \phi_{j,\max}]$. Как показано в примере 11.1, мы часто используем

$$\begin{aligned} \phi_{j,\max} &\approx 2 \\ K &= 2\alpha/(3\phi_T - 2) \end{aligned} \tag{11.33}$$

где $\alpha \in (0, 1)$. Множитель 3 в значении K компенсирует тот факт, что в уравнении (11.27) ϕ_T является суммой трех членов $\phi_{j,\max}$, а в уравнении (11.31) является средним значением членов $\phi_{j,\max}$. Веса w_{ij} в уравнении (11.31) являются детерминированными факторами, описывающими влияние j-й частицы на скорость i-й частицы. Иногда мы используем w_{ij} = постоянный для всех j. В других случаях мы хотим, чтобы w_{ij} был крупнее для значений j, соответствующих более оптимальным частицам x_j, а также крупнее для значений j, соответствующих частицам x_j, которые ближе к x_i. Например, если нашей задачей является минимизационная задача, то мы могли бы использовать что-то вроде

$$w_{ij} = \left[\max_k f(x_k) - f(x_j)\right] + \left[\max_k |x_i - x_k| - |x_i - x_j|\right], \tag{11.34}$$

где $|\cdot|$ – это мера расстояния. Возможно, нам также придется соответствующим образом взвесить вклад стоимости и приспособленности, для того чтобы они оба вносили равные порядки величины в w_{ij}. Например,

$$S_i = \frac{\max_k f(x_k) - \min_k f(x_k)}{\max_k |x_i - x_k|}$$

$$w_{ij} = \left[\max_k f(x_k) - f(x_j)\right] + S_i\left[\max_k |x_i - x_k| - |x_i - x_j|\right] \tag{11.35}$$

S_i – это коэффициент шкалирования, который делает двух членов, которые вносят вклад в w_{ij}, приблизительно равными. Поскольку уравнение (11.31) позволяет каждой частице влиять на любую другую частицу, оно называется полноинформированным роем частиц (fully informed particle swarm, FIPS) [Mendes и соавт., 2004]. Эта идея напоминает глобальную равномерную рекомбинацию в эволюционных алгоритмах (раздел 8.8.6).

■ Пример 11.2

В данном примере мы используем полноинформированный рой частиц уравнения (11.31) с весами уравнения (11.35) для оптимизации 20-мерной функции Экли. Мы применяем популяцию размером 40 и параметр элитарности 2. Используем номинальные значения

$$\phi_{j,\max} = \phi_{\max} = 2, \quad \text{для } j \in [1, 20]$$
$$K = 2\alpha/(3\phi_{\max} - 2), \quad \alpha = 0.9 \tag{11.36}$$

На рис. 11.10 и 11.11 показана средняя результативность оптимизации на основе роя частиц при разных значениях $\phi_{\max}$ и α, когда другой параметр равен его номинальному значению.

На рис. 11.10 показано, что когда $\phi_{\max}$ мал, рой сходится очень быстро, но он сходится к слабому решению. По мере увеличения параметра $\phi_{\max}$ начальное схождение замедляется, но окончательное сходящееся решение становится лучше. Это может мотивировать нас на использование адаптивного $\phi_{\max}$, который изначально мал, а затем постепенно увеличивается с течением времени. Рисунок 11.11 показывает, что для малых значений схождение очень медленное. Схождение быстрее всего для $\alpha = 0.9$, но окончательное решение лучше для $\alpha = 0.5$.

Эти результаты весьма специфичны. Они применяются для конкретной эталонной функции с конкретной размерностью, конкретным параметром элитарности и конкретной формой для взвешивающих параметров w_{ij} (уравнение (11.35)). Для того чтобы увидеть, можно ли обобщить выводы из этого примера на более широкий круг задач, требуются дополнительные эксперименты.

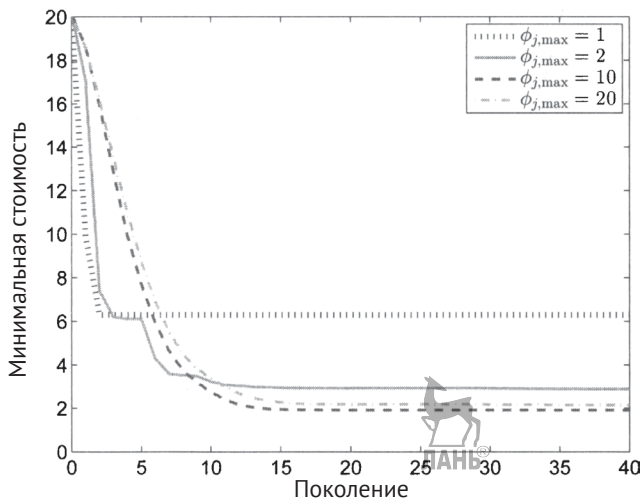


Рис. 11.10. Пример 11.2: результативность полноинформированного роя частиц на 20-мерной функции Экли для разных значений $\phi_{\max}$, усредненно по 20 симуляциям Монте-Карло. $\phi_{\max} = 1$ дает лучшую краткосрочную результативность, тогда как более крупные значения $\phi_{\max}$ дают лучшую долгосрочную результативность

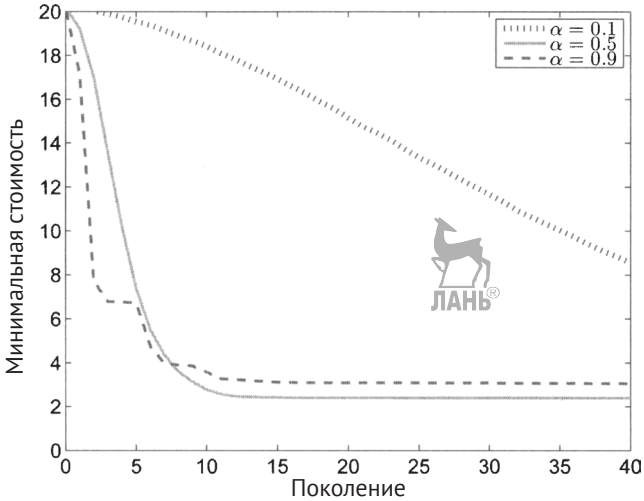


Рис. 11.11. Пример 11.2: результативность полноинформированного роя частиц на 20-мерной функции Экли для разных значений α, усредненно по 20 симуляциям Монте-Карло. $\alpha = 0.9$ дает лучшее краткосрочное поведение, тогда как $\alpha = 0.5$ дает лучшее долгосрочное поведение

■

Иногда оптимизация на основе полноинформированного роя частиц записывается иначе, чем уравнение (11.31). Например, уравнение (11.31) можно заменить следующим [Poli и соавт. 2007]:

$$v_i(t+1) = K\left[v_i(t) + \frac{1}{n_i}\sum_{j=1}^{n_i}\phi_i(b_{i,j}(t) - x_i(t))\right], \qquad (11.37)$$

где n_i – это размер окрестности i-й частицы, ϕ_j берется из равномерного распределения $U[0, \phi_{\max}]$, а $b_{ij}(t)$ – лучшее решение, найденное к настоящему моменту j-м соседом i-й частицы. В этой формулировке каждая частица имеет определенную фиксированную окрестность и лучшее решение $b_{ij}(t)$ каждого соседа имеет (в среднем) равновзвешенный вклад в обновление скорости i-й частицы. Обратите внимание, что при определенных условиях уравнение (11.37) эквивалентно уравнению (11.11). В некоторых публикациях было обнаружено, что оптимизация на основе полноинформированного роя частиц показывает слабую результативность, потому что частицы испытывают на себе слишком много конфликтующих притяжений или потому, что поисковое пространство каждой частицы уменьшается с увеличением размера окрестности [de Oca и Stützle, 2008].

11.6. Самообучение на ошибках

Оптимизация на основе роя частиц основана на идее, что биологические организмы склонны повторять стратегии, которые доказывали свою успешность в прошлом. Сюда входят полезные стратегии, которые они использовали сами, а также полезные стратегии, которые они наблюдали у других. Основным уравнением для скорости обновления, как мы видим в уравнении (11.27), является

$$v_i \leftarrow K\left[v_i + \phi_1(b_i - x_i) + \phi_2(h_i - x_i) + \phi_3(g - x_i)\right], \tag{11.38}$$

где x_i и v_i – это положение и скорость i-й частицы, b_i – предыдущее лучшее положение i-й частицы, h_i – текущее лучшее положение i-й окрестности, g – предыдущее лучшее положение всего роя в целом и K, $\phi_{1,\max}$, $\phi_{2,\max}$ и $\phi_{3,\max}$ – положительные регулировочные параметры.

Однако биологические организмы не только учатся на успехах, но и учатся на ошибках. Мы склонны избегать стратегий, которые оказались вредными в прошлом. Сюда входят пагубные стратегии, которые мы использовали сами, а также пагубные стратегии, которые мы наблюдали у других. Естественным продолжением оптимизации на основе роя частиц является встраивание этого избегания негативного поведения в базовый алгоритм оптимизации на основе роя частиц. Этот алгоритм получил название «нового» алгоритма в работах [Yang и Simon, 2005], [Selvakumar и Thanushkodi, 2007], но термин «новый» является шаблонным и неописательным, поэтому в этом разделе мы именуем его алгоритмом оптимизации на основе роя частиц с «отрицательным подкреплением» (negative reinforcement PSO, NPSO).

В алгоритме оптимизации на основе роя частиц с отрицательным подкреплением каждая частица изменяет свою скорость не только в направлении лучшего положения от себя и своих соседей, но и прочь от направления худшего положения от себя и своих соседей. Поэтому уравнение (11.38) модифицируется, принимая следующий вид:

$$\begin{aligned} v_i \leftarrow K\big[&v_i + \phi_1(b_i - x_i) + \phi_2(h_i - x_i) + \phi_3(g - x_i) \\ &- \phi_4(\overline{b}_i - x_i) + \phi_5(\overline{h}_i - x_i) + \phi_6(\overline{g} - x_i)\big] \end{aligned} \tag{11.39}$$

где $\overline{b}_i$ – это предыдущее худшее положение i-й частицы, $\overline{h}_i$ – текущее худшее положение i-й окрестности, $\overline{g}$ – предыдущее худшее положение всего роя в целом; каждый ϕ_j берется из равномерного распределения на $(0, \phi_{j,\max})$, и каждый $\phi_{j,\max}$ является положительным регулировочным параметром.

Мы должны найти баланс между регулировкой скорости в сторону полезных решений, которые приходят из алгоритма стандартной оптимизации на основе роя частиц, и регулировкой скорости прочь от вредных решений, которые мы добавили в алгоритм оптимизации на основе роя частиц с отрицательным подкреплением. Это именно тот баланс, который мы все пытаемся найти в нашей повседневной жизни. В какой мере мы фокусируемся на успехе и пытаемся подражать ему, по сравнению с тем, в какой мере мы фокусируемся на неудаче и пытаемся ее избегать? Большинство из нас согласно с тем, что позитивное подкрепление эффективнее, чем негативное, но многие из нас также согласны с тем, что оба типа подкрепления важны для самообучения.

■ Пример 11.3

В данном примере мы используем оптимизацию на основе роя частиц с отрицательным подкреплением уравнения (11.39) для оптимизации 20-мерной функции Швефеля 2.26. Мы используем популяцию размером 20 и параметр элитарности 2. Применяем номинальные значения

$$\begin{aligned}\phi_{1,\max} &= \phi_{2,\max} = \phi_{3,\max} = 2\\ \phi_{4,\max} &= \phi_{5,\max} = \phi_{6,\max} = 0\end{aligned}$$

$$K = \frac{2\alpha}{\phi_{1,\max} + \phi_{2,\max} + \phi_{3,\max} - 2}, \quad \alpha = 0.9 \tag{11.40}$$

Рисунки 11.12–11.14 показывают среднюю результативность оптимизации на основе роя частиц с отрицательным подкреплением для разных значений $\phi_{4,\max}$, $\phi_{5,\max}$ и $\phi_{6,\max}$; когда остальные параметры равны их номинальным значениям. На рис. 11.12 показано, что при увеличении значения параметра $\phi_{4,\max}$, который определяет, насколько каждая частица избегает своего предыдущего худшего положения, выше своего номинального значения 0 это может привести к значительному улучшению результативности. На рис. 11.13 показано аналогичное, но менее драматическое улучшение, когда $\phi_{5,\max}$, который определяет, насколько каждая частица избегает текущего худшего положения своей окрестности, увеличивается за пределы ее номинального значения 0. В конечном счете рис. 11.14 показывает, что результативность также улучшается, когда $\phi_{6,\max}$, который определяет, насколько каждая частица избегает предыдущего худшего положения всего роя в целом, увеличивается за пределы его номинального значения 0. Из этих цифр

следует, что $\phi_{6,\max}$ оказывает наибольшее влияние на результативность оптимизации на основе роя частиц с отрицательным подкреплением.

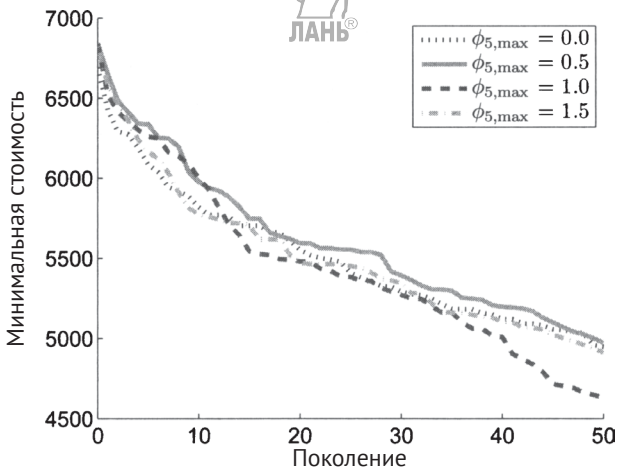


Рис. 11.12. Пример 11.3: результативность оптимизации на основе роя частиц с отрицательным подкреплением на 20-мерной функции Швефеля 2.26 для разных значений $\phi_{4,\max}$, усредненно по 20 симуляциям Монте-Карло. Частицы, которые избегают своего собственного предыдущего худшего положения, показывают значительно более высокую результативность, чем частицы, которые этого не делают

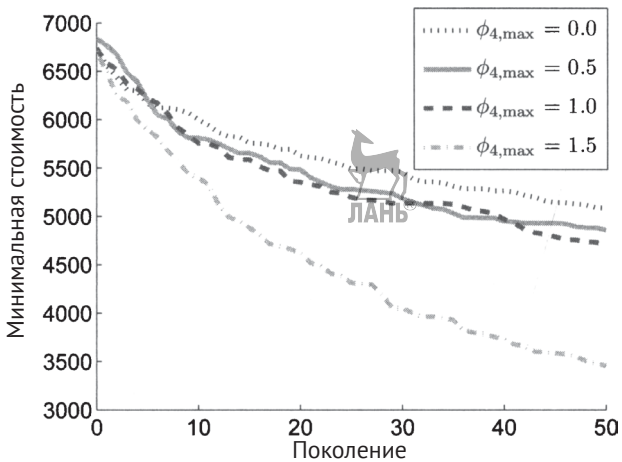


Рис. 11.13. Пример 11.3: результативность оптимизации на основе роя частиц с отрицательным подкреплением на 20-мерной функции Швефеля 2.26 для разных значений $\phi_{5,\max}$, усредненно по 20 симуляциям Монте-Карло. Частицы, которые избегают текущего худшего положения своих соседей, показывают заметно более высокую результативность, чем частицы, которые этого не делают

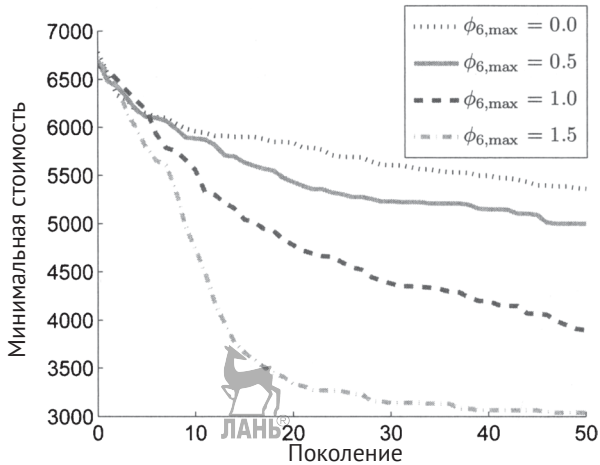


Рис. 11.14. Пример 11.3: результативность оптимизации на основе роя частиц с отрицательным подкреплением на 20-мерной функции Швефеля 2.26 для разных значений $\phi_{6,\max}$, усредненно по 20 симуляциям Монте-Карло. Частицы, которые избегают предыдущего худшего положения роя, показывают значительно более высокую результативность, чем частицы, которые этого не делают

■

Пример 11.3 показывает, что оптимизация на основе роя частиц с отрицательным подкреплением может показывать намного более высокую результативность, чем стандартная оптимизация на основе роя частиц. Обратите внимание, что в примере 11.3 мы изменяли только один из членов отрицательного подкрепления за один раз, оставляя двух других равными нулю. Мы не пытались комбинировать ненулевые значения $\phi_{4,\max}$, $\phi_{5,\max}$ и $\phi_{6,\max}$, но оставляем это читателям для будущих исследований. Также обратите внимание, что мы можем объединить идею отрицательного подкрепления с оптимизацией на основе полноинформированного роя частиц в уравнении (11.31). Мы также оставляем это расширение читателю для дальнейших исследований. Наконец, было бы интересно пересмотреть результаты расчетов стабильности раздела 11.3 для оптимизации на основе роя частиц с отрицательным подкреплением; это еще одна область для будущих исследований.

11.7. Заключение

Оптимизация на основе роя частиц зарекомендовала себя как эффективный эволюционный алгоритм для решения различных задач. Любое исследование вновь предлагаемого эволюционного алгоритма должно

включать сравнение с алгоритмом оптимизации на основе роя частиц из-за его хорошей результативности. Аналогично оптимизации на основе муравьиной кучи, некоторые исследователи не рассматривают оптимизацию на основе роя частиц как эволюционный алгоритм, а вместо этого считают его типом роевого интеллекта. Следует признать, что частицы оптимизации на основе роя частиц непосредственно не обмениваются информацией о кандидатном решении друг с другом. Однако оптимизация на основе роя частиц все-таки включает отбор на основе приспособленности, и частицы роя делятся информацией о скорости друг с другом, и информация о скорости непосредственно влияет на решения. Поэтому в этой книге мы классифицируем оптимизацию на основе роя частиц как эволюционный алгоритм.

Оптимизация на основе роя частиц с использованием стимулятора-зубатки (catfish PSO) – это модификация, которая была введена для борьбы с застаиванием в алгоритме оптимизации на основе роя частиц [Yang и соавт., 2011]. В аквариуме сардины часто устраиваются в локально оптимальное поведение и положение, но затем становятся вялыми и испытывают быстрое ухудшение здоровья. Если в аквариум добавляется зубатка, то сардины испытывают обновленное чувство стимуляции и остаются здоровыми в течение более длительного периода времени. Оптимизация на основе роя частиц с использованием стимулятора опирается на это наблюдение и стимулирует популяцию роя частиц, когда она ничинает застаиваться. Если лучшая особь в популяции роя частиц не улучшилась в течение m поколений подряд (m часто находится между 3 и 7), то каждая независимая переменная худших 10 % популяции принимается равной одной из границ поисковой области. Причина, почему частицы перемещаются к границам поисковой области, заключается в максимизации поискового пространства. Кроме того, оптимизационные задачи с ограничениями часто имеют решения, лежащие на границе ограничения [Bernstein, 2006].

Все обсуждение в этой главе было сосредоточено на оптимизации на основе роя частиц для задач непрерывной области. Алгоритм оптимизации на основе роя частиц был расширен несколькими различными способами для комбинаторной оптимизации [Kennedy и Eberhart, 1997], [Yoshida и соавт., 2001], [Clerc, 2004]. Прочие текущие направления исследований включают упрощение алгоритма оптимизации на основе роя частиц [Pedersen и Chipperfield, 2010], гибридизируя его с другими эволюционными алгоритмами [Niknam и Amiri, 2010], добавление операторов наподобие мутации для предотвращения преждевременного схождения [Xinchao, 2010], использование нескольких взаимодейству-

ющих роев [Chen и Montgomery, 2011], удаление случайности из алгоритма оптимизации на основе роя частиц [Clerc, 1999], использование динамических и адаптивных топологий [Ritscher и соавт., 2010], разведывание стратегии инициализации [Gutierrez и соавт., 2002] и адаптацию параметров оптимизации онлайн [Zhan и соавт., 2009]. Кроме того, обратите внимание, что подобно тому, как мы моделируем скорость каждой частицы роя частиц, мы можем также моделировать их ускорения [Tripathi и соавт., 2007]. Другая будущая исследовательская работа может включать анализ поведения и анализ схождения роя частиц, в котором учитываются случайность алгоритма и связи между частицами.

Дополнительное рекомендуемое чтение и исследование в области оптимизации на основе роя частиц включает книги [Kennedy и Eberhart, 2001], [Clerc, 2006], [Sun et al, 2011], а также публикации [Bratton и Kennedy, 2007], [Banks и соавт., 2007], [Banks и соавт., 2008]. Полезные и обширные веб-сайты по оптимизации на основе роя частиц содержат [PSC, 2012] и [Clerc, 2012a].

Задачи

Письменные упражнения

11.1. Какие аргументы существуют в пользу наличия статических окрестностей в алгоритме оптимизации на основе роя частиц? Какие аргументы существуют в пользу динамических окрестностей?

11.2. Ускорение в алгоритме оптимизации на основе роя частиц:

a) Как изменить алгоритм оптимизации на основе роя частиц на рис. 11.1, для того чтобы включить ускорение?

b) С учетом этой модификации алгоритма оптимизации на основе роя частиц как изменится уравнение (11.4) и каковы будут собственные значения?

11.3. Предположим, что $\phi_1 = 4$ в уравнении (11.4).

a) Каковы собственные значения матрицы?

b) Является ли система стабильной?

c) Дайте исходное условие и введите b_i, которое приведет к тому, что x_i и v_i будут ограничены как $t \to \infty$.

d) Дайте исходное условие и введите b_i, которые приведут к тому, что x_i и v_i будут не ограничены как $t \to \infty$.

11.4. В уравнении (11.35) для расчета веса w_{ij} используются стоимость и расстояние x_i. Какие еще признаки x_i мы могли бы использовать при расчете w_{ij}?

11.5. Исходя из того, что в уравнении (11.30) $p_i(t)$ постоянно, напишите уравнения динамического пространства состояний для $x_i(t + 1)$ и $v_i(t + 1)$. Каковы собственные значения системы?

11.6. При каких условиях уравнения (11.11) и (11.37) эквивалентны?

11.7. Обобщите обновление уравнения (11.39) алгоритма оптимизации на основе роя частиц с отрицательным подкреплением, для того чтобы получить обновление уравнения оптимизации на основе полноинформированного роя частиц.

11.8. Уравнение (11.25) рекомендует устанавливать коэффициент сужения следующим образом:

$$K = \frac{2\alpha}{\phi_T - 2},$$

где $\alpha \in (0, 1)$. Мы можем принять ϕ_T равным сумме максимально возможных значений членов ϕ_i; в этом случае ϕ_T является для алгоритма оптимизации на основе роя частиц постоянной; либо мы можем принять ϕ_T равным сумме фактических членов ϕ_i, которые случайно вычисляются для каждого обновления скорости; в этом случае ϕ_T – разное для каждого обновления скорости. Исходя из того, что для нашего обновления скорости мы используем уравнение (11.27), эти два варианта можно записать следующим образом:

$$K_1 = \frac{2\alpha_1}{\phi_{1,\max} + \phi_{2,\max} + \phi_{3,\max} - 2}$$

$$K_2 = \frac{2\alpha_2}{\phi_1 + \phi_2 + \phi_3 - 2}$$

где каждый ϕ_i равномерно распределен на $[0, \phi_{i,\max}]$. Какое значение α_2 в вышеприведенных уравнениях составляет $K_1 = K_2$ в среднем? (См. задачу 11.12 для компьютерного аналога упражнения для этой задачи.)

Компьютерные упражнения

11.9. Размеры окрестности: просимулируйте алгоритм оптимизации на основе роя частиц на рис. 11.1 для 40 поколений, для того чтобы минимизировать 10-мерную сферическую функцию (см. приложение С.1.1 относительно определения сферической функции). Используйте популяцию размером 20 и примените глобальное обновление скорости из уравнения (11.27). Применяйте $\phi_{1,\max} = \phi_{2,\max} = \phi_{3,\max} = 2$. Используйте $v_{\max} = \infty$ и $\alpha = 0.9$, для того чтобы найти коэффициент сужения K. Выполните 20 симуляций Монте-Карло для размеров окрестности $\sigma = 0$, 5 и 10. Постройте график средней результативности каждого набора симуляций Монте-Карло в зависимости от номера поколения. Какой вывод вы сделаете о важности локальных окрестностей в оптимизации на основе роя частиц?

11.10. Взвешивание расстояния на полноинформированном рое частиц: уравнение (11.35) может быть записано как

$$w_{ij} = w_{ij}(c) + Sw_{ij}(d),$$

$$\text{где } w_{ij}(c) = \max_k f(x_k) - f(x_j)$$

$$w_{ij}(d) = \max_k \left|x_i - x_k\right| - \left|x_i - x_j\right|$$

где $w_{ij}(c)$ – это стоимостной вклад x_j в w_{ij} и $w_{ij}(d)$ – вклад расстояния. Приведенное выше уравнение можно обобщить следующим образом:

$$w_{ij} = \frac{w_{ij}(c) + DSw_{ij}(d)}{(1+D)},$$

где D – это важность вклада расстояния по отношению к стоимости. Используйте эту весовую формулу для симуляции оптимизации на основе полноинформированного роя частиц для оптимизации 20-мерной функции Растригина (см. приложение С.1.11 относительно определения функции Растригина). Выполните 20 симуляций Монте-Карло для $D = 0$, 0.5, 1, 2 и 1000. Постройте график средней результативности каждого набора симуляций Монте-Карло как функции от номера поколения и предоставьте некоторые общие наблюдения о результатах.

11.11. Размеры окрестности полноинформированного роя частиц: реализуйте обновление скорости из уравнения (11.37) в симуляции оптимизации на основе роя частиц для минимизации 20-мерной функции Розенброка (см. приложение С.1.4 относительно определения функции Розенброка). Используйте размер популяции 20 и лимит числа поколений 40. Настройте параметры ϕ_{max} и K для получения хорошей результативности. Выполните 20 симуляций Монте-Карло с размерами окрестностей 2, 5, 10 и 20. Постройте график средней результативности каждого набора симуляций Монте-Карло как функции от номера поколения и предоставьте некоторые наблюдения о результатах.

11.12. Постоянное сужение против сужения, варьирующегося во времени: промоделируйте алгоритм оптимизации на основе роя частиц на рис. 11.1 для 50 поколений, для того чтобы минимизировать 10-мерную функцию Экли (см. приложение С.1.2 относительно определения функции Экли). Используйте популяцию размером 20 и примените глобальное обновление скорости из уравнения (11.27). Применяйте $\phi_{1,max} = \phi_{2,max} = \phi_{3,max} = 2.1$, $v_{max} = \infty$ и $\alpha_1 = 0.9$, для того чтобы найти коэффициент сужения K_1, определенный в задаче 11.8. Выполните 20 симуляций Монте-Карло с постоянным коэффициентом сужения K_1 и 20 симуляций Монте-Карло с изменяющимся во времени коэффициентом сужения K_2, который вы нашли в задаче 11.8. Постройте график средней результативности каждого набора симуляций Монте-Карло в зависимости от номера поколения и прокомментируйте свои результаты.

Глава 12

Дифференциальная эволюция

По сравнению с несколькими существующими эволюционными алгоритмами, дифференциальная эволюция гораздо проще и прямолинейнее в реализации. ...Простота программирования важна для практиков из других областей, так как они не могут быть экспертами в программировании....

– С. Дас, П. Сугантан и С. Коэльо Коэльо [Das и соавт., 2011]

Дифференциальная эволюция (differential evolution, DE) была разработана Райнером Сторном и Кеннетом В. Прайсом (Rainer Storn, Kenneth V. Price) примерно в 1995 году. Как и многие новые оптимизационные алгоритмы, дифференциальная эволюция была обусловлена реальными задачами: решением полиномиальных коэффициентов Чебышева и оптимизацией коэффициентов цифрового фильтра. Дифференциальная эволюция сделала быстрое и впечатляющее вхождение в мир эволюционных алгоритмов, заняв одно из первых мест на Первом Международном конкурсе по эволюционным вычислениям [Storn и Price, 1996] и на Втором Международном конкурсе по эволюционной оптимизации [Price, 1997]. Первые публикации по дифференциальной эволюции находились в материалах конференций [Storn, 1996a], [Storn, 1996b], а первая публикация в журнале вышла год спустя [Storn и Price, 1997]. Тем не менее первая широко читаемая публикация по дифференциальной эволюции была в нерецензируемом журнале [Price и Storn, 1997]. Дифференциальная эволюция является уникальным эволюционным алгоритмом, потому что он не обусловлен биологически.

Краткий обзор главы

В разделе 12.1 дается описание базового алгоритма дифференциальной эволюции для оптимизации на непрерывных областях. После первоначального введения дифференциальной эволюции исследователи разработали ряд ее вариаций, и мы обсудим несколько таких вариаций в разделе 12.2. После того как алгоритм дифференциальной эволюции оказался успешным для задач с непрерывной областью, исследователи распространили его на дискретные области, и поэтому в разделе 12.3 мы обсудим дифференциальную эволюцию для задач с дискретной областью. Алгоритм дифференциальной эволюции первоначально был представлен не как отдельный эволюционный алгоритм, а как вариация генетического алгоритма, и поэтому в разделе 12.4 мы посмотрим на алгоритм дифференциальной эволюции с этой точки зрения.

12.1. Базовый алгоритм дифференциальной эволюции

Дифференциальная эволюция – это популяционно-ориентированный алгоритм, который предназначен для оптимизации функций в n-мерной непрерывной области. Каждая особь в популяции является n-мерным вектором, который представляет собой кандидатное решение задачи. Алгоритм дифференциальной эволюции основывается на идее взятия разностного вектора между двумя особями и добавления шкалированной версии разностного вектора к третьей особи для создания нового кандидатного решения. Этот процесс показан на рис. 12.1.

Рисунок 12.1 изображает дифференциальную эволюцию в двумерном поисковом пространстве. Две особи, x_{r2} и x_{r3}, отбираются случайно при $r_2 \neq r_3$. Шкалированная версия разницы между этими двумя особями добавляется к третьей случайно отобранной особи, x_{r1}, где $r_1 \notin \{r_2, r_3\}$. В результате мы получаем мутанта v_i, который мог бы быть принят в популяцию в качестве нового кандидатного решения.

После создания мутантного вектора v_i он комбинируется (то есть скрещивается) с дифференциально-эволюционной особью x_i, где $i \notin \{r_1, r_2, r_3\}$, для того чтобы создать пробный вектор u_i. Скрещивание реализовано следующим образом:

$$u_{ij} = \begin{cases} v_{ij} & \text{если } (r_{cj} < c) \text{ или } (j = \mathcal{J}_r) \\ x_{ij} & \text{в противном случае} \end{cases} \qquad (12.1)$$

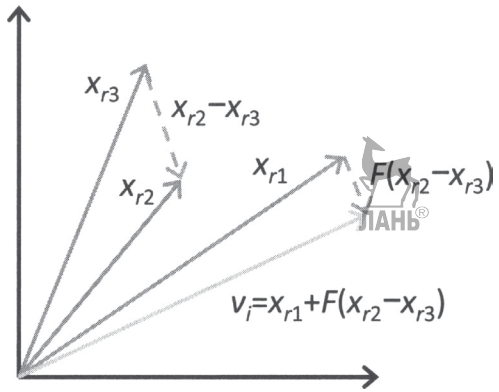


Рис. 12.1. Основная идея дифференциальной эволюции, проиллюстрированная для двумерной оптимизационной задачи (n = 2). x_{r1}, x_{r2} и x_{r3} – это кандидатные решения. Шкалированная версия разницы между особями x_{r2} и x_{r3} добавляется к x_{r1} для получения мутантного вектора v_i, который является новым кандидатным решением. Обратите внимание, что v_i индексируется индексом i, потому что каждое поколение мы генерируем n отдельных мутантных векторов, где n – это размер популяции

для $j \in [1, n]$, где n – это размерность задачи, а также размерность u_i, v_i и x_i. u_{ij} является j-м компонентом u_i; v_{ij} – это j-й компонент v_i; x_{ij} – это j-й компонент особи x_i; r_{cj} – это случайное число, которое берется из равномерного распределения [0, 1]; c – постоянная скорость скрещивания $\in$ [0, 1][1] и $\mathcal{J}_r$ – случайное число, которое берется из равномерного распределения [1, n]. Мы видим, что пробный вектор u_i представляет собой покомпонентную комбинацию текущей дифференциально-эволюционной особи x_i и мутантного вектора v_i. Цель $\mathcal{J}_r$ состоит в том, чтобы гарантировать, что u_i не является клоном x_i, хотя это осложнение для большинства задач может быть опущено (см. задачу 12.3). Скорость скрещивания c управляет тем, насколько вероятно, что каждый компонент u_i происходит от мутантного вектора v_i.

После того как N пробных векторов u_i были созданы, как описано выше, где N – это размер популяции, векторы u_i и x_i сравниваются. Наиболее приспособленный вектор в каждой паре (u_i, x_i) сохраняется для следующего дифференциально-эволюционного поколения, а наименее приспособленный отбрасывается. Базовый алгоритм дифференциальной эволюции для n-мерной задачи обобщен на рис. 12.2.

[1] В литературе по дифференциальной эволюции символ *Cr* по большей части используется для скорости скрещивания. Однако двухбуквенные символы могут быть неверно истолкованы как отдельные символы (например, *C*, умноженное на *r*), и поэтому в этой главе для скорости скрещивания используеся более стандартная математическая запись.

F = параметр размера шага $\in$ [0.4, 0.9]
c = скорость скрещивания $\in$ [0.1, 1]
Инициализировать популяцию кандидатных решений $\{x_i\}$ для $i \in [1, N]$.
While not (критерий останова)
  For each особь x_i $i \in [1, N]$
    $r_1 \leftarrow$ случайное целое $\in [1, N] : r_1 \notin i$
    $r_2 \leftarrow$ случайное целое $\in [1, N] : r_2 \notin \{i, r_1\}$
    $r_3 \leftarrow$ случайное целое $\in [1, N] : r_3 \notin \{i, r_1, r_2\}$
    $v_i \leftarrow x_{r1} + F(x_{r2} - x_{r3})$ (мутантный вектор)
    $\mathcal{J}_r \leftarrow$ случайное целое $\in [1, n]$
    For each размерность $j \in [1, n]$
      $r_{cj} \leftarrow$ случайное число $\in [0, 1]$
      If $(r_{cj} < c)$ **or** $(j = \mathcal{J}_r)$ **then**
        $u_{ij} \leftarrow v_{ij}$
      else
        $u_{ij} \leftarrow x_{ij}$
      End if
    Next размерность
  Next особь
  For each популяционный индекс $i \in [1, N]$
    If $f(u_i) < f(x_i)$ **then** $x_i \leftarrow u_i$
  Next популяционный индекс
Next поколение

Рис. 12.2. Простой алгоритм дифференциальной эволюции (DE) для минимизации n-размерной функции $f(x)$. Данный алгоритм называется классическим алгоритмом дифференциальной эволюции, или алгоритмом DE/rand/1/интервал

Как видно по рис. 12.2, алгоритм дифференциальной эволюции имеет несколько параметров, которые необходимо отрегулировать. Как и в любом другом эволюционном алгоритме, следует выбрать размер популяции. Специфичные для алгоритма дифференциальной эволюции параметры включают размер шага F, который также называется коэффициентом шкалирования, и скорость скрещивания c. Эти параметры зависят от задачи, но обычно (но не всегда) выбираются в диапазоне $F \in [0.4, 0.9]$ и $c \in [0.1, 1]$. Оптимальное значение F чаще всего уменьшается вместе с квадратным корнем размера популяции N. Оптимальное значение c обычно уменьшается вместе с разделимостью целевой функции [Price, 2013].

Алгоритм на рис. 12.2 часто называют классическим алгоритмом дифференциальной эволюции. Он еще носит название алгоритм

DE/rand/1/интервал, потому что основной вектор, x_{r1}, выбирается случайно; одновекторная разность (то есть $F(x_{r2} - x_{r3})$) добавляется к x_{r1}, и число элементов мутантного вектора, которые вносятся в пробный вектор, близко подчиняется биномиальному распределению. Оно бы в точности соответствовало биномиальному распределению, если бы не проверка «$j = \mathcal{J}_r$» (см. задачу 12.1).

Некоторые размышления о классическом алгоритме дифференциальной эволюции на рис. 12.2 показывают, почему он работает [Price, 2013]. Во-первых, возмущения (пертурбации) формы $(x_{r2} - x_{r3})$ уменьшаются по мере того, как популяция сужается к решению задачи. Во-вторых, размеры возмущений различаются от размерности к размерности в зависимости от шкалы задачи. То есть величина p-й компоненты $(x_{r2} - x_{r3})$ пропорциональна тому, насколько близка популяция к решению задачи вдоль p-й размерности. В-третьих, шаги возмущений коррелируют между размерностями, что делает поиск эффективным даже для чрезвычайно неразделимых задач (см. приложение С.7.2). Результатом этих признаков алгоритма дифференциальной эволюции является *совпадение контуров*, что означает, что дифференциально-эволюционная популяция распределяется вдоль контуров целевой функции. Дифференциально-эволюционная популяция имеет тенденцию адаптироваться к форме целевой функции.

12.2. Вариации дифференциальной эволюции

В этом разделе мы рассмотрим несколько вариаций алгоритма дифференциальной эволюции. В разделе 12.2.1 показан альтернативный способ создания пробного вектора u_i на каждой итерации. В разделе 12.2.2 даны некоторые альтернативные способы создания мутантного вектора v, а в разделе 12.2.3 обсуждаются возможности использования случайного коэффициента шкалирования F.

12.2.1. *Пробные векторы*

Обратите внимание, что метод рис. 12.2 не включает в себя никакого механизма содержания признаков решения вместе из v_i или x_i. То есть вероятность копирования v_{ij} в u_{ij} одинакова независимо от того, было или нет $v_{i,j-1}$ скопировано в $u_{i,j-1}$. Вместе с тем существует ряд задач, для которых приспособленность зависит от комбинаций признаков решения, а не отдельных признаков решения, поэтому бывает, что возникает потребность поддерживать признаки решения вместе. Алгоритм диф-

ференциальной эволюции DE/rand/1/L работает, генерируя случайное целое число $L \in [1, n]$, копируя L признаков подряд из v_i в u_i, а затем копируя оставшиеся признаки из x_i в u_i [Storn и Price, 1996].

Например, предположим, что мы имеем семимерную задачу ($n = 7$). Алгоритм DE/rand/1/L работает, сначала генерируя случайное целое число $L \in [1, n]$; предположим, что $L = 3$. Затем мы генерируем случайную исходную точку $s \in [1, n]$; предположим, что $s = 6$. При наличии этих параметров признаки решения из v_i и x_i копируются в пробный вектор u_i следующим образом:

$$\begin{aligned}
u_{i1} &\leftarrow v_{i1} \\
u_{i2} &\leftarrow x_{i2} \\
u_{i3} &\leftarrow x_{i3} \\
u_{i4} &\leftarrow x_{i4} \\
u_{i5} &\leftarrow x_{i5} \text{ (конечная точка)} \\
u_{i6} &\leftarrow v_{i6} \text{ (исходная точка } s\text{)} \\
u_{i7} &\leftarrow v_{i7}
\end{aligned} \tag{12.2}$$

Мы видим, что копируем элементы v_i в элементы u_i в шестой размерности, начиная с $s = 6$ (то есть шестого признака решения). Поскольку $L = 3$, мы копируем три элемента подряд из v_i в u_i, где *подряд* означает, что мы циклически переходим в начало векторов после достижения их конца. После копирования трех элементов из v_i в u_i начинаем копировать элементы x_i в u_i, останавливаясь, когда u_i будет полностью определен. Более формально, алгоритм DE/rand/1/L работает, заменяя цикл «For each размерность» на рис. 12.2 циклом на рис. 12.3.

```
L ← случайное целое ∈ [1, n]
s ← случайное целое ∈ [1, n]
J ← {s, min(n, s + L – 1)} ∪ {1, s + L – n – 1}
For each размерность j ∈ [1, n]
    If j ∈ J
        u_ij ← v_ij
    else
        u_ij ← x_ij j
    End if
Next размерность
```

Рис. 12.3. Цикл алгоритма DE/rand/1/L, копирующий элементы из x_i и мутантного вектора v_i в пробный вектор u_i. Функция *a mod b* возвращает остаток от деления *a*/*b*. Данный цикл заменяет цикл «For each размерность» на рис. 12.2

Интересно рассмотреть среднее число элементов мутантного вектора v_{ij}, которые копируются в признаки пробного вектора u_{ij} для заданного индекса i. Для алгоритма DE/rand/1/интервал существует n итераций цикла «For each размерность» на рис. 12.2. Одна из таких итераций имеет 100%-ную вероятность копирования v_{ij} в u_{ij}, остальные $(n-1)$ итераций имеют вероятность c копирования v_{ij} в u_{ij}. Из этого следует, что ожидаемое число элементов v_{ij}, копируемых в пробный вектор, равно

$$E(\text{число копируемых элементов } v_i) = 1 + c(n-1) \quad \text{для DE/rand/1/интервал.} \tag{12.3}$$

Для алгоритма DE/rand/1/L L признаков мутантного вектора v_{ij} копируются в пробный вектор. Поскольку L равномерно распределено в $[1, n]$:

$$E(\text{число копируемых элементов } v_i) = n/2 \quad \text{для DE/rand/1/L.} \tag{12.4}$$

При каких условиях ожидаемое число элементов мутантного вектора, копируемых в пробный вектор, эквивалентно для вариантов «интервал» и «L»? Приравняв уравнения (12.3) и (12.4), мы получим

$$c = \frac{n-2}{2(n-1)}. \tag{12.5}$$

Это значение параметра скрещивания c в алгоритме DE/rand/1/интервал на рис. 12.2, которое чуть меньше 0.5, приведет к эквивалентному среднему числу элементов мутантного вектора, копируемых в пробный вектор в алгоритме DE/rand/1/L на рис. 12.3.

В общем случае мы можем установить L равным случайному целому числу между 1 и L_{max} с задаваемой пользователем константой $L_{max} \in [1, n]$. Мы видим, что на рис. 12.3 $L_{max} = n$, но значения L_{max} меньше n в некоторых задачах, возможно, будут показывать более высокую результативность.

■ Пример 12.1

В данном примере мы применим алгоритм дифференциальной эволюции к 20-мерной функции Экли, описанной в приложении С.1.2. Мы используем следующие параметры:

- размер популяции = 50;
- шаг $F = 0{,}4$;
- скорость скрещивания $c = 0.49$ из уравнения 12.5.

Мы рассмотрим разницу между генерацией пробного вектора с помощью варианта «интервал», показанного на рис. 12.2, и варианта «L», показанного на рис. 12.3. На рис. 12.4 дана лучшая особь каждого поколения, усредненно по 20 симуляциям Монте-Карло. Мы видим, что вариант «L» сходится быстрее в начале симуляции, но вариант «интервал» дает значительно более высокую результативность в долгосрочной перспективе. Мы не ожидаем каких-то улучшений от использования варианта «L», потому что признаки решения в функции Экли никак не связаны; то есть функция Экли является разделимой задачей. Вместе с тем не ясно, почему вариант «интервал» показывает намного более высокую результатвиность, чем вариант *L*.

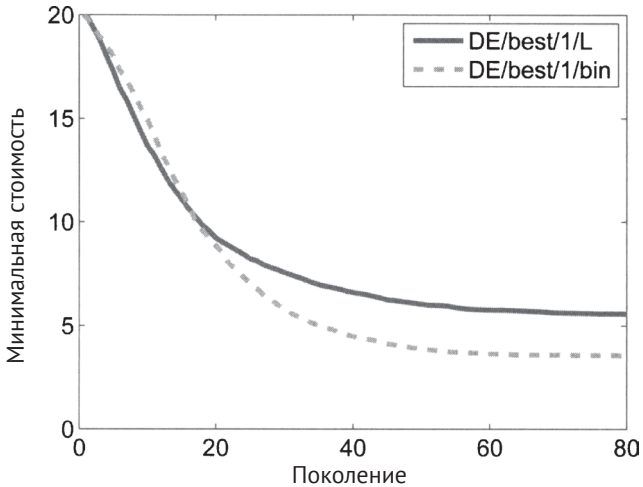


Рис. 12.4. Пример 12.1: результативность алгоритма дифференциальной эволюции для 20-мерной функции Экли для примера 12.1. Следы показывают стоимость лучшей особи в каждом поколении, усредненно по 20 симуляциям Монте-Карло. Вариант «интервал» для генерации пробных векторов показывает заметно более высокую результативность, чем вариант *L*

■

12.2.2. *Мутантные векторы*

В этом разделе мы рассмотрим некоторые альтернативы для создания мутантных векторов. Например, вместо случайного выбора основного вектора x_{r1} может быть полезно в качестве основного вектора всегда использовать лучшую особь в популяции. Благодаря этому все множество пробных векторов а для $i \in [1, n]$ целиком состоит из мутаций лучшей

особи. Такой подход называется DE/лучший/1/интервал [Storn и Price, 1996], [Storn, 1996b]. Он идентичен рис. 12.2, за исключением того что вычисление мутантного вектора заменено на

$$v_i \leftarrow x_b + F(x_{r2} - x_{r3}), \tag{12.6}$$

где x_b – это лучшая особь в популяции. Такой подход имеет своим следствием увеличение эксплуатации и снижение разведывания. Данная идея аналогична племенному эволюционному алгоритму, описанному в разделе 8.7.7. Если для создания мутантного вектора мы используем уравнение (12.6), а для копирования объектов в пробный вектор – рис. 12.3, то получим алгоритм DE/лучший/1/L.

Еще один вариант – для создания мутантного вектора использовать два разностных вектора [Storn и Price, 1996], [Storn, 1996b]. Этот подход может увеличить разведывание, так как общий разностный вектор не ограничивается локализацией в направлении разниц между парами векторов. Общий разностный вектор имеет больше степеней свободы. Каждую x_i итерацию цикла он может комбинироваться со случайной выборкой из основного вектора, как показано на рис. 12.2, либо с выборкой из лучшей особи x_i основного вектора, как показано в уравнении 12.6. В результате получим следующие два варианта генерации мутантного вектора:

$$\begin{aligned} r_4 &\leftarrow \text{случайное целое} \in [1, N] : r_4 \notin \{i, r_1, r_2, r_3\} \\ r_5 &\leftarrow \text{случайное целое} \in [1, N] : r_5 \notin \{i, r_1, r_2, r_3, r_4\} \\ v_i &\leftarrow \begin{cases} x_{r1} + F(x_{r2} - x_{r3} + x_{r4} - x_{r5}) & \text{DE/rand/2/?} \\ x_b + F(x_{r2} - x_{r3} + x_{r4} - x_{r5}) & \text{DE/лучший/2/?} \end{cases} \end{aligned} \tag{12.7}$$

Теперь дадим объяснение по поводу вопросительных знаков в конце приведенного выше уравнения. Если для создания мутантного вектора мы используем один из вариантов уравнения (12.7) и для копирования признаков в пробный вектор используем рис. 12.2, то получим алгоритм DE/rand/2/интервал или алгоритм DE/лучший/2/интервал. Если для создания мутантного вектора мы используем уравнение (12.7) и для копирования объектов в пробный вектор используем рис. 12.3, то получим алгоритм DE/rand/2/L или алгоритм DE/лучший/2/L.

Обратите внимание, что уравнение (12.7) увеличивает влияние разностных векторов на мутантный вектор. Если используется $F = F_0$ на рис. 12.2 или уравнение (12.6), то для справедливого сравнения в уравнении (12.7) следует использовать некоторое $F < F_0$. Точное соотношение между двумя значениями F зависит от формы целевой функции.

Алгоритм дифференциальной эволюции также может быть реализован с использованием текущего x_i в качестве основного вектора [Storn, 1996a]. Например,

$$v_i \leftarrow x_i + F\Delta x, \tag{12.8}$$

где Δx – это разностный вектор. В зависимости от метода, который мы используем для создания разностного вектора, и метода, который используем для создания пробного вектора. В результате получим алгоритмы DE/цель/1/интервал, DE/цель/2/интервал, DE/цель/1/L или DE/цель/2/L[2]. В отличие от алгоритмов DE/rand, описанных в разделах 12.1 и 12.2.1, алгоритмы DE/цель выглядят гораздо менее чувствительными к F [Price, 2013].

Еще один вариант – создавать разностный вектор, используя лучшую особь в популяции, x_b. Данный подход имеет тенденцию создавать мутантные векторы, которые все движутся к x_b. Вычитаемый из x_b вектор может быть случайной особью или базовой особью. Опираясь на эту идею, можно представить ряд возможностей. Например, в публикации [Storn, 1996a]:

$$\begin{aligned} v_i &\leftarrow x_i + F(x_b - x_i) \\ v_i &\leftarrow x_{r1} + F(x_b - x_{r3}) \\ v_i &\leftarrow x_b + F(x_{r2} - x_{r3} + x_b - x_{r3}) \\ v_i &\leftarrow x_i + F(x_b - x_i + x_{r2} - x_{r3}) \end{aligned} \tag{12.9}$$

и так далее. Если для генерации v_i используется последнее приведенное выше уравнение, то алгоритм называется DE/цель-к-лучшему/1/интервал [Price et al, 2005, раздел 3.3.1][3]. Обратите внимание, что для создания v_i иногда мы используем рекомбинацию, иногда мутацию, а иногда – обе операции. Первый вариант в уравнении (12.9) является операцией рекомбинации, поскольку он включает комбинацию x_i и другого вектора. Второй и третий варианты являются операциями мутации, потому что x_i в уравнениях не появляется. Четвертый вариант – это гибридная операция, потому что она включает в себя x_i, но она также включает и разность векторов $(x_{r2} - x_{r3})$, в которой x_i не появляется.

Мы можем комбинировать разные методы, принимая случайное решение, как генерировать мутантный вектор. Например, на рис. 12.5 показан метод генерации мутантного вектора, порождающий алгоритм

[2] Алгоритмы DE/цель также упоминаются в литературе как DE/текущий и DE/i.

[3] По-видимому, он должен называться DE/цель-к-лучшему/2/интервал, для того чтобы соответствовать другим соглашениям об именовании DE. С иной стороны, данное уравнение можно интерпретировать как добавление одной мутации к базовому вектору x_i, что оправдывает терминологию DE/цель-к-лучшему/1/интервал (DE/target-to-best/1/bin).

DE/rand/1/или-или [Price и соавт., 2005, раздел 2.6.5]. Если $\alpha < p_f$, то для создания B используется стандартный метод DE/rand/1/интервал. Однако если $\alpha > p_f$, то для создания B используется особый тип метода DE/rand/2.

На данный момент у нас получается уже слишком много метаморфоз алгоритма дифференциальной эволюции, чтобы с ними можно было справиться. Тем не менее стоит отметить, что большинство этих вариаций, как правило, имеет второстепенное значение. Основная идея алгоритма дифференциальной эволюции изображена на рис. 12.1 и 12.2, а все остальные возможные вариации – это лишь детали.

p_f = мутационная вероятность $\in$ [0, 1]
$\alpha \leftarrow$ случайное число $\in$ [0, 1]
If $\alpha < p_f$ **then**
  $v_i \leftarrow x_{r1} + F(x_{r2} - x_{r3})$
else
  $v_i \leftarrow x_{r1} + K(x_{r2} - x_{r1} + x_{r3} - x_{r1})$
End if

Рис. 12.5. Генерирование мутантного вектора, которое приводит к алгоритму DE/rand/1/или-или. Как правило, $K = (F + 1)/2$ дает хорошие результаты в эталонных задачах

■ Пример 12.2

В данном примере мы снова применяем алгоритм дифференциальной эволюции к 20-мерной функции Экли, описанной в приложении C.1.2. Поскольку пример 12.1 показал, что вариант «интервал» демонстрирует более высокую результативность, чем вариант «L», в этом примере мы используем вариант «интервал». Здесь мы рассмотрим, как изменяется результативность в зависимости от того, какой вектор используем в качестве основного. У нас есть три варианта. В качестве основного вектора мы можем использовать произвольный вектор x_{r1}, как показано на рис. 12.2, либо лучший вектор, как изображено в уравнении (12.6), либо в качестве основного вектора мы можем использовать текущий член популяции, как показано в уравнении (12.8). На рис. 12.6 дана лучшая особь каждого поколения, усредненно по 20 симуляциям Монте-Карло. Мы видим, что случайный и текущий варианты работают примерно одинаково. Это ожидаемо, поскольку: (1) текущий вариант приводит к тому, что каждая особь из текущей популяции используется в качестве основного вектора ровно один раз за поколение; (2) случайный вариант

приводит к тому, что каждая особь используется в качестве основного вектора в среднем один раз за поколение. С другой стороны, рис. 12.6 показывает, что «лучший» вариант явно превосходит два других варианта. Очевидно, что сосредоточение усилий на поиске лучшей особи для каждого поколения является выгодной стратегией.

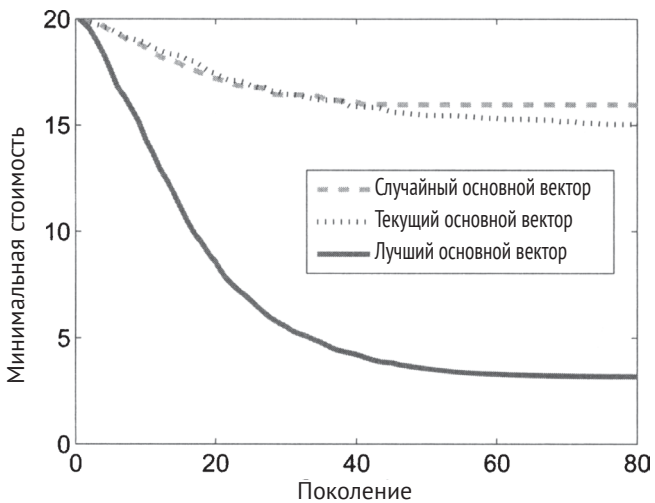


Рис. 12.6. Пример 12.2: результативность алгоритма дифференциальной эволюции для 20-мерной функции Экли. Следы показывают стоимость лучшей особи в каждом поколении, усредненно по 20 симуляциям Монте-Карло. Использование лучшей особи в качестве основного вектора дает значительно более высокую результативность

Поскольку использование лучшей особи в качестве основного вектора дает лучшую результативность, мы будем использовать этот вариант в остальной части данного примера. В качестве нашей последней симуляции в этом примере мы рассмотрим, как изменяется результативность в зависимости от того, сколько разностных векторов используется для генерации мутантного вектора. Мы можем использовать один разностный вектор, как показано в уравнении (12.6), либо два разностных вектора, как дано в уравнении (12.7). На рис. 12.7 показана лучшая особь каждого поколения, усредненно по 20 симуляциям Монте-Карло. Мы видим, что использование только одного разностного вектора работает немного лучше, чем использование двух разностных векторов, даже если F корректируется для эффекта использования двух разностных векторов, как описано в следующем далее уравнении (12.7). Причина этого неясна, но она согласуется с выводами, изложенными в публи-

кации [Price и соавт., 2005, раздел 2.4.7], и было бы интересно изучить данный вопрос более подробно.

■

В алгоритме дифференциальной эволюции мы также можем реализовать некоторые вариации эволюционного алгоритма. Например, в алгоритм дифференциальной эволюции мы можем легко добавить более стандартную операцию эволюционно-алгоритмической мутации, такую как гауссова или равномерная мутация, центрированная в кандидатном решении (см. раздел 8.9). Вместе с тем вполне возможно, что эта операция не окажет большого влияния, так как мутантный вектор алгоритма дифференциальной эволюции уже является в значительной степени разведывающим.

Напомним, что элитарность является общей чертой эволюционных алгоритмов, которая гарантирует, что мы не потеряем высокорезультативных особей и что результативность лучшей особи в популяции никогда не будет ухудшаться из поколения в поколение (см. раздел 8.4). Элитарность является привлекательным вариантом для всех эволюционных алгоритмов и обычно обеспечивает значительное улучшение результативности. Однако внедрять элитарность в алгоритм дифференциальной эволюции нет никакой необходимости, поскольку данный алгоритм автоматически оставляет лучших особей каждого поколения, как показано в цикле «For each популяционный индекс» на рис. 12.2. Но это поднимает вопрос, связанный с тем, что могут возникнуть задачи, для которых алгоритм дифференциальной эволюции работает лучше с менее агрессивной стратегией элитарности. Иногда, для того чтобы достичь хороших решений, эволюционным алгоритмам приходится пробиваться сквозь скудные участки поискового пространства. Неэлитарные эволюционные алгоритмы могут подходить лучше для некоторых задач с дорогими или динамическими функциями стоимости (см. главу 21).

12.2.3. *Корректировка коэффициента шкалирования*

Коэффициент шкалирования F алгоритма дифференциальной эволюции определяет влияние разностных векторов на мутантный вектор. До сих пор мы исходили из того, что F является постоянной величиной. Однако одним из признаков эволюционных алгоритмов является рандомизация, поэтому имеет смысл позволить коэффициенту F быть случайной величиной. Это даст возможность расширить диапазон мутантных векторов, что может привести к большему разведыванию во

время работы алгоритма дифференциальной эволюции. Кроме того, превращение коэффициента F в случайную величину позволит анализировать свойства сходимости алгоритма дифференциальной эволюции [Zaharie, 2002].

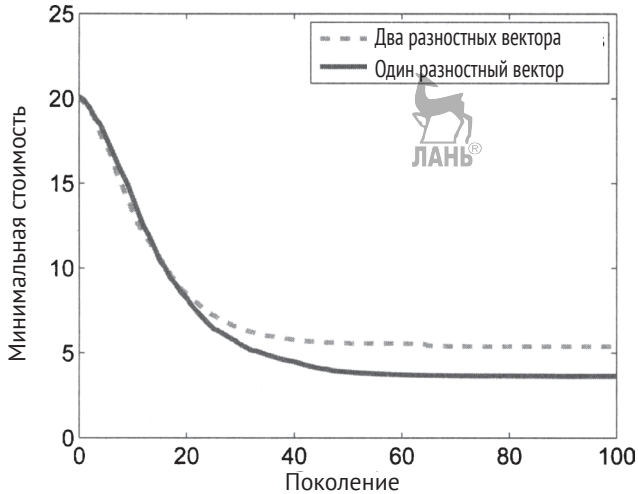


Рис. 12.7. Пример 12.2: результативность алгоритма дифференциальной эволюции для 20-мерной функции Экли для примера 12.2. Следы показывают стоимость лучшей особи в каждом поколении, усредненно по 20 симуляциям Монте-Карло. Использование только одного разностного вектора для генерации мутантных векторов выглядит несколько лучше, чем использование двух разностных векторов

В алгоритме дифференциальной эволюции мы можем варьировать коэффициент шкалирования двумя разными способами. Во-первых, мы можем позволить F оставаться скаляром и произвольно менять его на каждой итерации цикла «For each особь» на рис. 12.2. Этот тип вариации называется искажением «дитер» (dither). Во-вторых, мы можем заменить скаляр F на n-элементный вектор и случайно менять каждый элемент F в цикле «For each особь», так чтобы каждый элемент мутантного вектора v модифицировался уникально прошкалированной компонентой разностного вектора. Этот тип вариации называется искажением «джиттер».

Искажение «дитер» заменяет строку генерации мутантного вектора на рис. 12.2 на следующие:

$$\begin{aligned} F &\leftarrow U[F_{\min}, F_{\max}] \\ v_i &\leftarrow x_{r1} + F(x_{r2} - x_{r3}) \end{aligned} \tag{12.10}$$

То есть коэффициент шкалирования является случайным скаляром, равномерно распределенным между $F_{\min}$ и $F_{\max}$. Другие подходы с использованием дитера позволяют взять F из гауссова распределения [Price и соавт., 2005, раздел 2.5.2).

Искажение «джиттер» заменяет строку генерации мутантного вектора на рис. 12.2 на следующие:

For each размерность $j \in [1, n]$
$$F_j \leftarrow U[F_{\min}, F_{\max}]$$
$$v_{ij} \leftarrow x_{r1,j} + F_j(x_{r2,j} - x_{r3,j})$$
Next размерность (12.11)

То есть при создании мутантного вектора каждый элемент разностного вектора шкалируется на разную величину.

В общем случае постоянные значения F, по-видимому, работают лучше для простых функций (например, сферической функции), а рандомизированные значения F, по-видимому, лучше работают для большинства мультимодальных функций. Джиттер лучше всего работает для функций, которые являются в основном разделимыми, и дитер лучше всего работает на крайне неразделимых функциях [Price, 2013].

■ Пример 12.3

В данном примере мы рассмотрим использование искажений дитер и джиттер. Как и в предыдущих примерах, мы применяем популяцию размером 50. Мы используем $F_{\min} = 0.2$ и $F_{\max} = 0.6$ в уравнениях (12.10) и (12.11). На рис. 12.8 показана средняя результативность алгоритма дифференциальной эволюции на 20-мерной функции Экли со скоростью скрещивания $c = 0.9$ и с тремя разными реализациями коэффициента шкалирования: (1) постоянный F, (2) F с дитером и (3) F с джиттером. Мы видим, что варианты с искажениями дитер и джиттер показывают примерно одинаковую результативность, в то время как результативность варианта с постоянным F является лучшей. Это означает, что рандомизация F снижает результативность. Вместе с тем рис. 12.9 показывает те же самые результаты на оптимизационном эталоне Флетчера. В этом случае джиттер работает слегка, но явно лучше, чем вариант с постоянным коэффициентом F, который, в свою очередь, работает лучше, чем вариант F с дитером.

Эти результаты показывают, что эффекты искажений дитер и джиттер зависят от конкретной решаемой задачи, а также от других параметров в алгоритме дифференциальной эволюции. Эффект варьирова-

ния F зависит от того, какое значение постоянного коэффициента F мы сравниваем, скорости скрещивания, диапазона и типа распределения, используемого для варьирования F, и так далее.

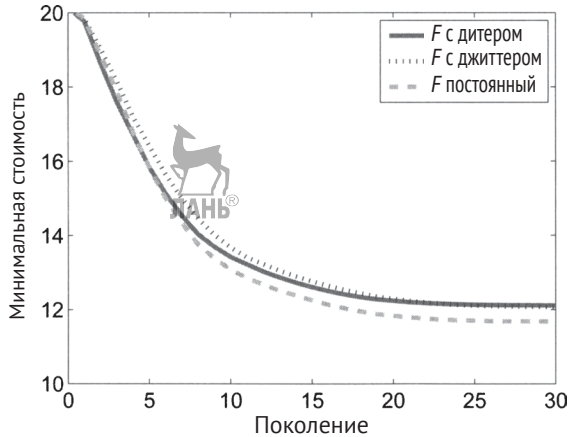


Рис. 12.8. Пример 12.3: результативность алгоритма дифференциальной эволюции для 20-мерной функции Экли со скоростью скрещивания c = 0.9. Следы показывают стоимость лучшей особи в каждом поколении, усредненно по 100 симуляциям Монте-Карло. Использование постоянного коэффициента шкалирования F работает немного лучше, чем искажения дитер или джиттер

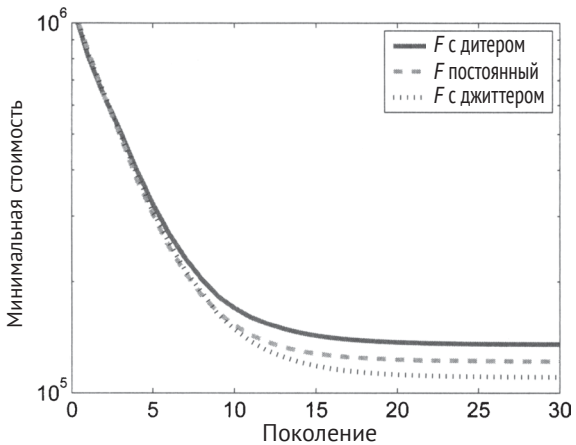


Рис. 12.9. Пример 12.3: результативность алгоритма дифференциальной эволюции для 20-мерной функции Флетчера со скоростью скрещивания c = 0.9. Следы показывают стоимость лучшей особи в каждом поколении, усредненно по 100 симуляциям Монте-Карло. Коэффициент F с джиттером работает немного лучше, чем F постоянный, который, в свою очередь, работает немного лучше, чем F с дитером

■

В уравнениях (12.10) и (12.11) используется равномерное распределение с нулевым средним для определения вариации коэффициента шкалирования ΔF от его номинального значения $(F_{\min} + F_{\max})/2$. Можно также использовать другие распределения, такие как равномерное с ненулевым средним и логарифмически нормальное. В некоторых задачах эти распределения показали улучшенную результативность [Price и соавт., 2005, раздел 2.5.2].

12.3. Дискретная оптимизация

В этом разделе мы обсудим, как алгоритм дифференциальной эволюции может быть использован для оптимизации функций в дискретной области. Единственное место, где дискретные области вызывают проблему в алгоритме дифференциальной эволюции, – это генерирование мутантного вектора. Напомним рис. 12.2, где мы видим, что

$$v_i \leftarrow x_{r1} + F(x_{r2} - x_{r3}). \tag{12.12}$$

Поскольку $F \in [0, 1]$, v_i вполне может не принадлежать области D задачи. Алгоритм дифференциальной эволюции изначально был разработан для задач с непрерывной областью, но он может быть модифицирован для работы с дискретными областями. Существует два похожих, но принципиально разных способа его модификации для дискретных задач. Во-первых, мы можем сгенерировать мутантный вектор v_i стандартными методами алгоритма дифференциальной эволюции, такими как метод уравнения (12.12), а затем модифицировать его так, чтобы он лежал в области D задачи; мы обсудим один из подходов в этом русле в разделе 12.3.1. Во-вторых, мы можем модифицировать метод генерации мутантных векторов, так чтобы мутантный вектор непосредственно генерировался, находясь в области D; мы обсудим один из подходов в этом контексте в разделе 12.3.2. Для получения дополнительных сведений об использовании алгоритма дифференциальной эволюции для дискретных областей обратитесь к публикации [Onwubolu и Davendra, 2009].

12.3.1. *Смешанно-целочисленная дифференциальная эволюция*

Один из очевидных подходов к обеспечению того, чтобы $v_i \in D$, заключается в том, чтобы просто проецировать его на D. Когда алгоритм дифференциальной эволюции модифицируется таким образом с целью

оптимизации на дискретной области, он часто называется алгоритмом смешанно-целочисленной дифференциальной эволюции [Huang и Wang, 2002], [Su и Lee, 2003]. Например, если D – это множество n-мерных целочисленных векторов, то уравнение (12.12) можно заменить следующим:

$$v_i \leftarrow \text{round}\,[x_{r1} + F(x_{r2} - x_{r3})], \tag{12.13}$$

где функция округления *round* оперирует на векторе поэлементно. Более общий способ сделать это выглядит так:

$$v_i \leftarrow P[x_{r1} + F(x_{r2} - x_{r3})], \tag{12.14}$$

где P – это проекционный оператор – такой, что $P(x) \in D$ для всех x. Уравнение (12.13) дает конкретную и простую возможность для P.

В общем случае P может быть сложнее, чем уравнение (12.13). Например, предположим, что предметная область D является множеством n-мерных целочисленных векторов. Тогда мы могли бы определить P как

$$P(x) = \arg\min_{\alpha} f(\alpha) : \alpha \in D, |x_j - \alpha_j| < 1 \quad \text{для всех } j \in [1, n]. \tag{12.15}$$

В результате вещественный вектор x спроецируется на целочисленный вектор α, что приведет к самой низкостоимостной функции, где каждый элемент α будет находиться в одном блоке соответствующего элемента x. Эта идея проиллюстрирована на рис. 12.10 для двух размерностей. В качестве проекционного оператора могут использоваться другие возможные формы, и они вполне могут зависеть от конкретной задачи.

12.3.2. *Дискретная дифференциальная эволюция*

Еще один способ модифицировать алгоритм дифференциальной эволюции для дискретных задач – изменить метод генерирования мутантных векторов таким образом, чтобы он непосредственно создавал мутантные векторы, которые лежат в дискретной области D. Когда алгоритм дифференциальной эволюции модифицируется подобным образом, он часто называется алгоритмом дискретной дифференциальной эволюции [Pan и соавт., 2008]. При таком подходе мы заменяем уравнение (12.12) на

$$v_i \leftarrow G(x_{r1}, x_{r2}, x_{r3}), \tag{12.16}$$

где $G(\cdot) \in D$, если все его аргументы находятся в D. Такой подход является обобщением уравнения (12.14), и поэтому мы видим, что алгоритм дискретной дифференциальной эволюции является обобщением смешанно-целочисленной дифференциальнй эволюции. Функция $G(\cdot)$ может быть написана для обработки общих дискретных задач или может быть сформулирована как специфичная для конкретной задачи функция. Например, предположим, что D – это множество n-мерных целочисленных векторов. Тогда для генерации мутантного вектора мы можем использовать следующие ниже варианты:

$$\begin{aligned} &\text{Вариант 1:} \quad v_i \leftarrow x_{r1} + \text{round}\,[F(x_{r2} - x_{r3})] \\ &\text{Вариант 2:} \quad v_i \leftarrow x_{r1} + \text{sign}\,(x_{r2} - x_{r3}) \end{aligned} \tag{12.17}$$

где функции *round* и *sign* оперируют на векторах поэлементно.

Напомним, что основная идея создания мутантного вектора алгоритмом дискретной дифференциальной эволюции состоит в том, чтобы получить v_i путем модификации вектора кандидатного решения (в приведенном выше уравнении это x_{r1}), используя разность двух других векторов кандидатного решения (в приведенном выше уравнении это x_{r2} и x_{r3}). Для алгоритма дискретной дифференциальной эволюции подходит любой метод, который дает $v_i \in D$. Существует ряд возможных методов, которые могут быть изучены в будущих исследованиях.

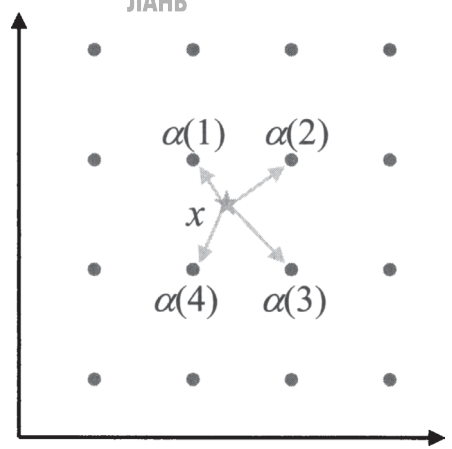


Рис. 12.10. Проекция непрерывнозначного вектора x на дискретный вектор α. Данный двумерный пример показывает, что x не находится в области задачи дискретной оптимизации. Значения стоимостной функции четырех ближайших к x точек в области задачи проходят проверку. $\alpha(i)$, которое приводит к наименьшему значению стоимостной функции, равно $P(x)$, проецируемому значению x

12.4. Дифференциальная эволюция и генетические алгоритмы

В этом разделе мы покажем, что алгоритм дифференциальной эволюции является особым типом непрерывного генетического алгоритма. Предположим, что мы ничего не знаем об алгоритме дифференциальной эволюции, но хотим разработать эволюционный алгоритм на основе материала, который прочитали во второй части данной книги. В частности, предположим, что мы хотим разработать модифицированную версию генетического алгоритма. По каждой особи x_i мы хотим вероятностно скопировать независимые переменные из случайно отобранной особи v_i, которую называем мутантным вектором, в x_i, чтобы получить ребенка u_i. Мы используем параметр c, который называем скоростью скрещивания, для того чтобы обозначить вероятность того, что независимая переменная в x_i будет заменяться соответствующей независимой переменной из v_i. Эта идея очень похожа на равномерное скрещивание в разделе 8.8.4, если в том разделе определить $x_\alpha = x_i$, $x_b = v_i$ и $y = u_i$. Кроме того, мы хотим заменить x_i на ребенка u_i, если ребенок лучше; эта идея похожа на эволюционную стратегию (1 + 1) раздела 6.1. С учетом данных идей мы предлагаем модифицированный генетический алгоритм, который приведен на рис. 12.11.

```
Инициализировать популяцию кандидатных решений {x_i}, i ∈ [1, N].
While not (критерий останова)
    For each особь x_i, i ∈ [1, N]
        r_1 ← случайное целое ∈ [1, N] : r_1 ≠ i
        v_i ← x_r1
      For each размерность j ∈ [1, n]
        If rand(0, 1) < c then
            u_ij ← v_ij
        else
            u_ij ← x_ij
        End if
      Next размерность
    Next особь
    For each i ∈ [1, N], If f(u_i) < f(x_i) then x_i ← u_i
Next поколение
```

Рис. 12.11. Приведенный выше псевдокод описывает версию 1 модифицированного генетического алгоритма минимизации $f(x)$, где c – это скорость скрещивания, а rand(0, 1) – случайное число $\in [0, 1]$

Теперь предположим, что мы хотим отрегулировать наш алгоритм, для того чтобы повысить его результативность. Вместо того чтобы назначать v_i случайную особь, мы решили для получения v_i пертурбировать случайную особь. В частности, мы решили пертурбировать случайную особь, как показано на рис. 12.1. Данное концептуальное изменение заключается в том, как мы получаем v_i, но оно, в конце концов, по-прежнему основано на текущих членах популяции. Мы также понимаем, что из-за инструкции «If rand(0, 1) < c» на рис. 12.11 существует возможность, что $u_{ij} = x_{ij}$ для всех $j \in [1, n]$, то есть существует возможность, что ребенок u_i будет клоном своего родителя x_i. Мы хотим это предотвратить, и поэтому думаем о том, чтобы по крайней мере одна независимая переменная в u_i копировалась из v_i. Мы делаем это, добавляя в инструкцию «If rand(0, 1) < c» еще одно условие; мы меняем данный оператор на «If (rand(0, 1) < c) or (j = случайный индекс $\in [1, n]$)», где n – это размерность задачи. С учетом этих идей мы получаем обобщение рис. 12.11, как показано в алгоритме рис. 12.12.

Теперь отметим, что рис. 12.12 идентичен базовому алгоритму дифференциальной эволюции на рис. 12.2, то есть данный алгоритм является особым типом генетического алгоритма. В этой связи возникают два вопроса.

1. Должен ли генетический алгоритм называться генетическим алгоритмом, либо его следует рассматривать как частный случай алгоритма дифференциальной эволюции?
2. Должен ли алгоритм дифференциальной эволюции называться алгоритмом дифференциальной эволюции, либо его следует рассматривать как вариант генетического алгоритма?

Для того чтобы ответить на первый вопрос, мы знаем, что титул генетического алгоритма никогда не устареет из-за его истории и его основополагающей важности в развитии эволюционных алгоритмов. Кроме того, титул генетического алгоритма целесообразен, потому что данный алгоритм поощряет встраивание биологических признаков в алгоритм (половое размножение, старение, островные популяции и т. д.), что может приводить к интересным и полезным расширениям генетического алгоритма.

Для того чтобы ответить на второй приведенный выше вопрос, сообщество исследователей и разработчиков эволюционных алгоритмов еще с 1990-х годов осознало, что алгоритм дифференциальной эволюции достаточно особенный, чтобы считать его отдельным эволюционным алгоритмом, и он не должен рассматриваться как частный случай

какого-либо другого эволюционного алгоритма. Вместе с тем, несмотря на то что на эти вопросы в отношении алгоритма дифференциальной эволюции были даны ответы, они имеют далеко идущие последствия для других эволюционных алгоритмов. Каждый год предлагаются новые эволюционные алгоритмы, некоторые из которых мы обсуждаем в главе 17. Какие из них заслуживают того, чтобы принадлежать своему классу, а какие следует рассматривать как обобщения или частные случаи уже установившихся эволюционных алгоритмов? По мере того как в литературе будет появляться все больше и больше эволюционных алгоритмов, находить свою нишу для новых эволюционных алгоритмов все труднее будет. Однако, так же как и алгоритм дифференциальной эволюции, который, несмотря на его сходство с генетическим алгоритмом, заслуживает своего собственного класса, некоторые из этих новых эволюционных алгоритмов также могут заслуживать своего собственного класса. Мы обсудим эту тему подробнее в главе 17[4].

Инициализировать популяцию кандидатными решениями $\{x_i\}$, $i \in [1, N]$.
While not (критерий останова)
  For each особь x_i, $i \in [1, N]$
    $r_1 \leftarrow$ случайное целое $\in [1, N] : r_1 \neq i$
    $r_2 \leftarrow$ случайное целое $\in [1, N] : r_2 \notin \{i, r_1\}$
    $r_3 \leftarrow$ случайное целое $\in [1, N] : r_3 \notin \{i, r_1, r_2\}$
    $v_i \leftarrow x_{r1} + F(x_{r2} - x_{r3})$
    $\mathcal{J}_r \leftarrow$ случайное целое $\in [1, n]$
    For each размерность $j \in [1, n]$
      If (rand(0, 1) < c) **or** ($j = \mathcal{J}_r$) **then**
        $u_{ij} \leftarrow v_{ij}$
      else
        $u_{ij} \leftarrow x_{ij}$
      End if
    Next размерность
  Next особь
  For each $i \in [1, N]$, **If** $f(u_i) < f(x_i)$ **then** $x_i \leftarrow u_i$
Next поколение

Рис. 12.12. Приведенный выше псевдокод описывает версию 2 модифицированного генетического алгоритма минимизации $f(x)$, где F – размер шага, c – скорость скрещивания и rand (0, 1) – случайное число $\in [0, 1]$

[4] Одна из первых публикаций по алгоритму дифференциальной эволюции включает подзаголовок «простая эволюционная стратегия быстрой оптимизации» [Price, 1997]. Однако алгоритм дифференциальной эволюции, по всей видимости, имеет не слишком много общего с эволюционными стратегиями.

12.5. Заключение

Текущие исследования в области алгоритма дифференциальной эволюции отражают текущие исследования в области других эволюционных алгоритмов: упрощение алгоритма дифференциальной эволюции [Omran и соавт., 2009], онлайновая адаптация контрольных параметров алгоритма дифференциальной эволюции [Qin и соавт., 2009], гибридизация с другими поисковыми алгоритмами [Noman и Iba, 2008] и расширение алгоритма дифференциальной эволюции на специальные типы оптимизационных задач, такие как динамические задачи [Brest и соавт., 2009], многокритериальные задачи [Mezura-Montes и соавт., 2008], [Dominguez и Pulido, 2011] и ограниченные задачи [Lampinen, 2002], [Mezura-Montes и Coello Coello, 2008]. Как и во многих других эволюционных алгоритмах, существует масса мест для теоретического и математического анализа, поэтому алгоритм дифференциальной эволюции будет плодотворной областью для дальнейших исследований. Было бы интересно сравнить подход на основе алгоритма дифференциальной эволюции к сопоставлению контуров (вспомните обсуждение в конце раздела 12.1) с подходом ковариационно-матричной адаптивной эволюционной стратегии (CMA-ES) (см. конец раздела 6.5). Дополнительную информацию по алгоритму дифференциальной эволюции можно найти в книгах [Price и соавт., 2005], [Feoktistov, 2006], [Qing, 2009], [Zhang и Sanderson, 2009], а также в учебно-справочных статьях [Das и Suganthan, 2011], [Neri и Tirronen, 2010] и главах книги [Syswerda, 2010].

Задачи

Письменные упражнения

12.1. В разделе 12.1 говорится, что число элементов k мутантного вектора, которые вносят вклад в пробный вектор, близко подчиняется биномиальному распределению. (См. задачу 12.9 для компьютерного аналога упражнения к этой задаче.)

а) С учетом эксперимента с вероятностью успеха c какова вероятность получения k успешных результатов в n независимых экспериментах?

б) С учетом эксперимента классического алгоритма дифференциальной эволюции какова вероятность того, что мутантный вектор внесет k компонент в пробный вектор?

12.2. Классический алгоритм дифференциальной эволюции раздела 12.1 требует генерирования трех случайных целых чисел, но генерирование случайных чисел, возможно, придется повторять вследствие их ограниченных допустимых значений.

a) Сколько в среднем требуется поколений случайных чисел для получения приемлемых значений r_1, r_2 и r_3? (*Подсказка*: используйте геометрическое распределение.)

b) С учетом $N = 20$ сколько в среднем потребуется поколений случайных чисел для получения приемлемых значений r_1, r_2 и r_3?

12.3. Предположим, что мы пропускаем проверку «$j = \mathcal{J}_r$» на рис. 12.2. Какова вероятность того, что u_i является клоном x_i? Какова эта вероятность, если $c = 0.5$ и $n = 20$?

12.4. Предположим, что мы хотим реализовать алгоритм DE/rand/1/L, как описано в разделе 12.2.1, за исключением того, что мы не хотим давать элементам v циклически переходить в начало при достижении конца вектора при копировании их в u_i. В этом случае мы можем заменить значение J на рис. 12.3 инструкцией $J \leftarrow \{s, \min(n, s + L - 1)\}$. Каким будет среднее число признаков v, копируемых в пробный вектор?

12.5. Как изменить алгоритм дифференциальной эволюции, так чтобы он стал неэлитарным?

12.6. Предложите стохастический проекционный оператор для алгоритма смешанно-целочисленной дифференциальной эволюции.

12.7. Предложите стохастический генератор мутантных векторов для алгоритма дискретной дифференциальной эволюции.

12.8. Модифицируйте инструкцию «If $f(u_i) < f(x_i)$, then $x_i \leftarrow u_i$» на рис. 12.2 так, чтобы алгоритм дифференциальной эволюции стал больше похож на эволюционную стратегию $(\mu + \lambda)$.

Компьютерные упражнения

12.9. Число мутантных вкладов: в задаче 12.1 получены две вероятности: (1) вероятность k успешных результатов из n независимых испытаний, каждое из которых имеет вероятность успеха c; (2) вероятность того, что мутантный вектор вносит k компонент

в пробный вектор. Постройте график этих двух вероятностей как функции k для $n = 20$ и $c = 0.5$.

12.10. Размер шага алгоритма дифференциальной эволюции: реализуйте классический алгоритм дифференциальной эволюции на рис. 12.2 для минимизации 10-мерной функции Розенброка (см. приложение С.1.4 относительно определения функции Розенброка). Используйте популяцию размером $N = 100$, скорость скрещивания $c = 0.9$ и лимит поколений 30. Выполните 40 симуляций Монте-Карло для каждого из следующих значений размера шага F: 0.1, 0.3, 0.5, 0.7 и 0.9. Для каждого набора симуляций Монте-Карло вычислите среднее значение лучшей стоимости 40 симуляций в каждом поколении. Постройте график средней результативности каждого набора симуляций Монте-Карло как функции от номера поколения и прокомментируйте свои результаты.

12.11. Скорость скрещивания алгоритма дифференциальной эволюции: реализуйте классический алгоритм дифференциальной эволюции на рис. 12.2 для минимизации 10-мерной функции Растригина (см. приложение С.1.11 для определения функции Растригина). Используйте популяцию размером $N = 100$, шаг $F = 0.4$ и лимит поколений 50. Выполните 40 симуляций Монте-Карло для каждого из следующих значений скрещивания CR: 0.1, 0.5 и 0.9. Для каждого набора симуляций Монте-Карло вычислите среднее значение лучшей стоимости 40 симуляций в каждом поколении. Постройте график средней результативности каждого набора симуляций Монте-Карло как функции от номера поколения и прокомментируйте свои результаты.

Глава 13

Алгоритмы оценивания вероятностных распределений

Алгоритмы оценивания вероятностных распределений принимают другой подход к выборке из поискового пространства. Популяция используется для оценивания распределения вероятностей на поисковом пространстве, которое отражает то, что рассматривается как важные характеристики популяции.

– Олден Райт [Wright и соавт., 2004]

Алгоритм оценивания вероятностного распределения (estimation-of-distribution algorithm, EDA) оптимизирует функцию, отслеживая статистические показатели популяции кандидатных решений [Larrañaga и Lozano, 2002]. Поскольку при таком подходе поддерживаются статистические показатели о популяции, сама популяция не нуждается в поддержании из поколения в поколение. Популяция создается в каждом поколении из статистических показателей популяции предыдущего поколения, а затем вычисляются статистические показатели наиболее приспособленных особей в популяции. Наконец, новая популяция создается с использованием статистических показателей наиболее приспособленных особей. Этот процесс повторяется из поколения в поколение. Таким образом, алгоритмы оценивания вероятностных распределений – это популяционно-ориентированные алгоритмы, которые каждое поколение отбрасывают по крайней мере часть популяции и заменяют ее, используя статистические свойства высоко приспособленных особей. Алгоритмы оценивания вероятностных распределений отличаются от большинства эволюционных алгоритмов

тем, что обычно не включают в себя рекомбинацию. Алгоритмы оценивания вероятностных распределений также называются генетическими алгоритмами с построением вероятностной модели (probabilistic model-building genetic algorithm, PMBGA) [Pelikan и соавт., 2002] и итерированными алгоритмами оценивания плотности вероятности (iterated density estimation algorithm, IDEA) [Bosman и Thierens, 2003].

Краткий обзор главы

Раздел 13.1 начинает эту главу с представления в общих чертах типичного алгоритма оценивания вероятностного распределения и демонстрации того, как содержательные статистические показатели могут быть рассчитаны на основе популяции эволюционно-алгоритмических особей. Во всех алгоритмах оценивания вероятностных распределений используются статистические показатели, подобные тем, которые вычисляются в разделе 13.1.2, для создания следующего поколения особей. В разделе 13.2 описываются некоторые популярные алгоритмы оценивания вероятностных распределений для дискретных оптимизационных задач, которые опираются только на статистические показатели первого порядка, включая алгоритм одномерного маржинального распределения (univariate marginal distribution algorithm, UMDA), компактный генетический алгоритм (compact genetic algorithm, cGA) и популяционное инкрементное самообучение (population based incremental learning, PBIL), которое является обобщением алгоритма одномерного маржинального распределения UMDA. В разделе 13.3 описываются некоторые дискретные алгоритмы оценивания вероятностных распределений, в которых используются статистические показатели второго порядка, включая максимизацию взаимной информации для кластеризации входных данных (mutual information maximization for input clustering, MIMIC), объединение оптимизаторов с деревьями взаимной информации (combining optimizers with mutual information trees, COMIT) и алгоритм двумерного маржинального распределения (bivariate marginal distribution algorithm, BMDA). В разделе 13.4 обсуждаются многомерные алгоритмы оценивания вероятностных распределений, то есть алгоритмы оценивания вероятностных распределений, в которых используются статистические показатели более высокого порядка, и дается схематичное описание расширенного компактного генетического алгоритма (extended compact genetic algorithm, ECGA).

Все упомянутые выше алгоритмы оценивания вероятностных распределений предназначены для задач с двоичными областями. Мы

завершим эту главу, показав, как распространять эти алгоритмы оценивания вероятностных распределений на задачи с непрерывными областями. Мы проиллюстрируем эту идею, представив в разделе 13.5 алгоритмы одномерного маржинального непрерывного распределения UMDA и популяционного инкрементного самообучения PBIL.

13.1. Алгоритмы оценивания вероятностных распределений: основные понятия

В этом разделе (в подразделе 13.1.1) представлено базовое схематичное описание типичного алгоритма оценивания вероятностного распределения, а также в подразделе 13.1.2 показано, как из популяции эволюционно-алгоритмических особей вычисляют содержательные статистические показатели.

13.1.1. *Простой алгоритм оценивания вероятностного распределения*

На рис. 13.1 схематично показан базовый алгоритм оценивания вероятностного распределения. Каждый алгоритм оценивания вероятностного распределения имеет свой собственный уникальный подход к трем основным шагам алгоритма рис. 13.1. Во-первых, каким образом M особей отбираются из общей популяции N кандидатных решений? Во-вторых, какой статистический показатель вычисляется из M особей, и как этот статистический показатель вычисляется? В-третьих, каким образом этот статистический показатель используется для создания новой популяции следующего поколения? Именно ответы на эти вопросы порождают различные типы алгоритмов оценивания вероятностных распределений, и это будет занимать наше внимание на протяжении большей части оставшейся главы.

Первым шагом в цикле рис. 13.1 является отбор M особей из популяции N кандидатных решений, где $M < N$. Это можно сделать разными способами. В методе отбора, применяемом в алгоритме оценивания вероятностного распределения, нет ничего особенного. Отбор может быть выполнен точно так же, как и в любом другом эволюционном алгоритме, как описано в разделе 8.7, поэтому дальше в этой главе тема отбора обсуждаться больше не будет.

Инициализировать популяцию кандидатных решений $\{x_i\}$, $i \in [1, N]$.
While not (критерий останова)

Отобрать M особей из $\{x_i\}$ согласно приспособленности, где $M < N$.

Вычислить статистические показатели из M отобранных выше особей.

Применить статистические показатели для создания новой популяции $\{x_i\}$, $i \in [1, N]$.

Next поколение

Рис. 13.1. Схематичное описание алгоритма оценивания вероятностного распределения (EDA)

13.1.2. *Вычисление статистических показателей*

В этом разделе дается описание того, как вычисляются статистические показатели популяции особей, что является вторым шагом в цикле рис. 13.1. Мы рассмотрим эту тему на простом примере. Предположим, что у нас есть бинарная оптимизационная задача и N кандидатных решений и что мы вычисляем их значения приспособленности, используя некоторый метод на основе приспособленности для отбора M особей, где $M < N$. Отбор должен быть смещен в сторону более приспособленных особей, как и в любом другом эволюционном алгоритме (см. раздел 8.7). Предположим, что M = 10, и мы отбираем следующих 10 особей:

$$\begin{aligned} x_1 &= (0, 1, 1, 1, 1, 0), \quad x_2 = (0, 1, 1, 1, 1, 1), \\ x_3 &= (1, 0, 0, 1, 1, 0), \quad x_4 = (1, 1, 1, 0, 1, 0), \\ x_5 &= (0, 1, 0, 0, 0, 1), \quad x_6 = (0, 1, 0, 0, 1, 0), \\ x_7 &= (0, 0, 1, 1, 1, 0), \quad x_8 = (1, 0, 1, 0, 1, 0), \\ x_9 &= (0, 1, 0, 0, 0, 0), \quad x_{10} = (0, 1, 1, 1, 1, 1). \end{aligned} \tag{13.1}$$

Среднее значение этих особей можно легко вычислить как

$$\bar{x} = (0.3, 0.7, 0.6, 0.5, 0.8, 0.3). \tag{13.2}$$

Среднее значение – это статистический показатель первого порядка. Мы видим, что первый бит этой относительно приспособленной субпопуляции имеет только 30%-ный шанс равняться 1. Поэтому, когда мы генерируем следующую популяцию, первый бит каждой особи должен иметь 30%-ный шанс равняться 1 и 70%-ный шанс равняться 0. Мы также видим, что второй бит имеет 70%-ный шанс равняться 1. Поэтому когда мы генерируем следующую популяцию, второй бит каждой особи должен иметь 70%-ный шанс равняться 1.

Однако мы также можем воспользоваться статистическими показателями второго порядка. Обратите внимание, что если в уравнении (13.1) первый (самый левый) бит $x_i(1) = 1$, то второй бит $x_i(2)$ имеет только 1/3-й шанс равняться 1, и если в уравнении (13.1) $x_i(1) = 0$, то второй бит имеет 6/7-й шанс равняться 1. Похоже, что между первым и вторым значения-

ми бит вполне может существовать некая корреляция. Таким образом, вместо того чтобы дать второму биту 70%-ный шанс равняться 1 в каждом члене новой популяции, возможно, мы должны подождать до тех пор, пока не создадим первый бит. Тогда если первый бит равен 1, то мы должны дать второму биту 1/3-й шанс равняться 1, и если первый бит равен 0, то мы должны дать второму биту 6/7-й шанс равняться 1.

Наконец, обратите внимание, что для создания следующего поколения мы могли бы использовать статистические показатели третьего или даже более высокого порядка. Например, мы видим, что если четвертый и пятый биты в уравнении (13.1) равны соответственно 0 и 1, то последний бит всегда равен 0.

13.2. Алгоритмы оценивания вероятностных распределений на основе статистических показателей первого порядка

В этом разделе представлены три алгоритма оценивания вероятностного распределения с использованием статистических показателей первого порядка, включая алгоритм одномерного маржинального распределения (UMDA) в разделе 13.2.1, компактный генетический алгоритм (cGA) в разделе 13.2.2 и популяционное инкрементное самообучение (PBIL) в разделе 13.2.3.

13.2.1. *Алгоритм одномерного маржинального распределения (UMDA)*

Алгоритм одномерного маржинального распределения (univariate marginal distribution algorithm, UMDA) – это наиболее элементарный алгоритм оценивания вероятностного распределения, который введен Хайнцом Мюленбайном (Heinz Mühlenbein) для бинарных задач в конце 1990-х годов [Mühlenbein и Paaβ, 1996], [Mühlenbein и Schlierkamp-Voosen, 1997]. В нем для генерации популяции в каждом поколении используются статистические показатели только первого порядка. Рисунок 13.2 дает схематичное представление об алгоритме одномерного маржинального распределения для бинарных оптимизационных задач.

Хотя стандартный алгоритм одномерного маржинального распределения не включает элитарность, она может использоваться с данным алгоритмом точно так же, как и с любым другим эволюционным алгоритмом. Параметр элитарности *e* означает, что мы оставляем лучших *e*

особей из поколения в поколение. За счет этого обеспечивается то, что лучшая особь в каждом поколении никогда не будет хуже лучшей особи предыдущего поколения, и гарантируется непрерывное улучшение (см. раздел 8.4).

Инициализировать популяцию кандидатных решений $\{x_i\}$, $i \in [1, N]$.
Отметить, что каждый x_i включает n бит $x_i(1)$, ..., $x_i(n)$.
While not (критерий останова)
  Отобрать M особей из $\{x_i\}$ согласно приспособленности, где $M < N$.
  Индексировать M отобранных особей как $\{x_i\}$, $i \in [1, M]$.
  $\Pr\{x_i(k) = 1) \leftarrow \sum_{i=1}^{M} \sigma(x_i(k) - 1) / M$, для $k \in [1, n]$
  For i – 1 **to** N (размер популяции)
    For $k = 1$ **to** n (число бит в каждом кандидатном решении)
      $r \leftarrow U[0, 1]$
      If $r < \Pr(x(k) = 1)$
        $x_i(k) \leftarrow 1$
      else
        $x_i(k)) \leftarrow 0$
      End if
    Next бит
  Next особь
Next поколение

Рис. 13.2. Схематичное представление алгоритма одномерного маржинального распределения (UMDA) для оптимизации в N-битной двоичной области. $\sigma(y)$ – это дельта-функция Кронекера, то есть $\sigma(y) = 1$, если $y = 0$ и $\sigma(y) = 0$, если $y \neq 0$. $U[0, 1]$ – это случайное число, генерируемое из равномерного распределения между 0 и 1. $x_i(k)$ – это k-й бит в i-й особи

■ Пример 13.1

В данном примере мы используем алгоритм одномерного маржинального распределения на рис. 13.2 для минимизации 20-мерной функции Экли, которая определена в приложении С.1.2. Мы используем шесть бит на размерность, поэтому минимизационная задача включает $n = 120$ бит. Мы используем область [–5, +5] для каждой из 20 размерностей, что дает нам разрешающую способность $10/(2^6 - 1) = 0.16$ для каждой размерности. Мы применяем популяцию размером $N = 100$ и параметр элитарности 2, то есть мы всегда оставляем лучших двух особей из поколения в поколение. Мы испытали четыре разных значения для M: 2, 10, 40 и 70. На рис. 13.3 показана результативность алгоритма одномерного маржинального распределения, усредненно по 50 симу-

ляциям Монте-Карло. Рисунок 13.3 показывает, что если для расчета вероятностей мы используем слишком мало или слишком много особей, то его результативность будет не очень хорошей. Для того чтобы получить хорошую результативность, в расчете вероятности мы должны использовать только нужное число особей.

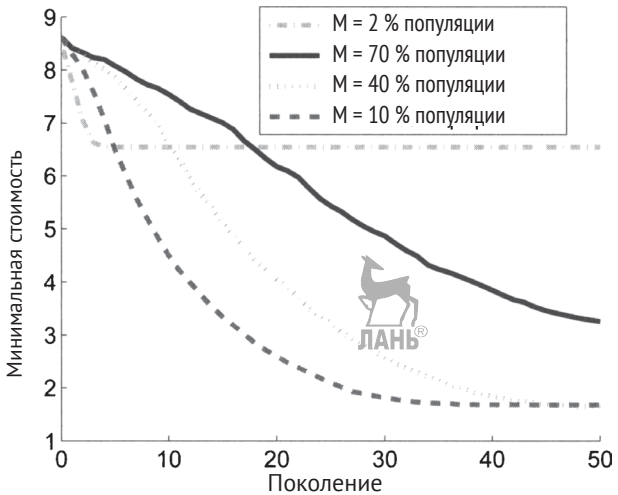


Рис. 13.3. Пример 13.1: Результаты работы алгоритма одномерного маржинального распределения для минимизации 20-мерной функции Экли с шестью битами на размерность. Результаты показывают стоимость лучшей особи в каждом поколении, усредненно по 50 симуляциям Монте-Карло

■

Дальнейшие исследования алгоритма одномерного маржинального распределения могли бы включать модификацию расчета вектора вероятности с использованием взвешенных вкладов каждой особи. На рис. 13.2 показано, что вектор вероятности вычисляется как

$$\Pr(x(k) = 1) \leftarrow \frac{1}{M}\sum_{i=1}^{M}\sigma(x_i(k) - 1), \quad k \in [1, n]. \tag{13.3}$$

Каждая из M лучших особей в популяции имеет одинаковый вклад в вычисление вектора вероятности. Но имеет смысл взвешивать лучших особей сильнее, чем худших. Такое расширение будет похоже на оптимизацию на основе полноинформированного роя частиц. В стандартной оптимизации на основе роя частиц для регулировки скорости каждой частицы используются окрестности определенного размера, но

в оптимизации на основе полноинформированного роя частиц раздела 11.5 для регулировки скорости каждой частицы используется вся популяция в целом. Эта идея может привести к замене уравнения (13.3) на нечто вроде следующего:

$$\Pr(x(k) = 1) \leftarrow \sum_{i=1}^{N} w_i \sigma(x_i(k) - 1) \Big/ \sum_{i=1}^{N} w_i, \quad k \in [1, n], \tag{13.4}$$

где w_i пропорциональна приспособленности x_i.

13.2.2. *Компактный генетический алгоритм (cGA)*

Компактный генетический алгоритм (compact genetic algorithm, cGA) был разработан Жоржем Хариком, Фернандо Лобо и Дэвидом Голдбергом (Georges Harik, Fernando Lobo и David Goldberg) в 1999 году [Harik и соавт., 1999]. Как следует из названия, он представляет собой минималистский подход к эволюционным вычислениям. Хотя в его названии находится термин генетический алгоритм, компактный генетический алгоритм больше похож на алгоритм оценивания вероятностного распределения, чем на генетический алгоритм. Как и алгоритм одномерного маржинального распределения (UMDA), для создания особей в каждом поколении в нем используются статистические показатели только первого порядка. При наличии оптимизационной задачи на двоичной n-элементной области мы начинаем с n-элементного вектора вероятности p, где каждый элемент инициализируется значением 1/2. Затем мы случайно генерируем двух особей x_1 и x_2, используя p, для того чтобы определить вероятность значения для каждого бита в каждой из двух особей. Потом мы измеряем приспособленность двух особей. Если одна особь приспособлена больше, чем другая, и i-й бит у двух особей отличается, то мы соответственно корректируем i-й элемент вектора вероятности p. Мы переходим к следующему поколению, используя обновленный вектор вероятности. На рис. 13.4 показан базовый алгоритм компактного генетического алгоритма.

Инициализировать n-элементный вектор вероятности $p = [0.5, \cdots, 0.5]$.
Назначить p_{min} и p_{max}, минимальное и максимальное значения по каждому элементу p.
Определить α, прирост обновления вероятности.
While not (критерий останова)
  For $i = 1$ **to** 2 (размер популяции)
    For $k = 1$ **to** n (число бит в каждом кандидатном решении)
      $r \leftarrow U[0, 1]$

```
        If r < p(k)
            x_i(k) ← 1
        else
            x_i(k)) ← 0
        End if
    Next бит
Next особь
Вычислить x_1 и x_2 и переупорядочить их так, чтобы x_1 был более
        приспособленным, чем x_2.
For k – 1 to n (число бит в каждом кандидатном решении)
    If x_1(k) ≠ x_2(k) then
        If x_1(k) = 1 then
            p(k) ← p(k) + α
        else
            p(k) ← p(k) – α
        End if
    p(k) ← max(min(p(k), p_max), p_min)
    End if
Next бит
Next поколение
```

Рис. 13.4. Схематичное представление компактного генетического алгоритма (cGA) для оптимизации на n-битной двоичной области. $U[0, 1]$ – это случайное число, генерируемое из равномерного распределения между 0 и 1. $\alpha \in (0, 1)$ определяет скорость схождения. $x_i(k)$ – это k-й бит в i-й особи

Как и в любой другой эволюционный алгоритм, в компактный генетический алгоритм может быть встроена элитарность. В этом случае лучшая особь в конце каждого поколения присоединяется к двум новым особям, создаваемым в следующем поколении, и лучшая из трех особей используется для корректировки вектора вероятности.

■ Пример 13.2

В данном примере мы снова пытаемся минимизировать 20-мерную функцию Экли, на этот раз используя компактный генетический алгоритм на рис. 13.4. Наши параметры задачи такие же, как и в примере 13.1: шесть бит на размерность, что приводит к задаче минимизации с $N = 120$ бит; и область $[-5, +5]$ для каждой из 20 размерностей, дающая разрешающую способность $10/(2^6 - 1) = 0.16$ для каждой размерности. Мы используем популяцию размером $N = 2$, как показано на рис. 13.4. Мы использовали $p_{min} = 0.05$ и $p_{max} = 0.95$. Мы также применяем элитар-

ность, то есть всегда оставляем лучшую особь из поколения в поколение. Мы испытали три разных значения α: 0.001, 0.01 и 0.1. На рис. 13.5 показана результативность компактного генетического алгоритма, усредненно по 50 симуляциям Монте-Карло. Мы видим, что если α слишком велик, то имеется очень много скачков по поисковому пространству, и схождение оказывается слабым. Если же α слишком мал, то в течение первых поколений вектор вероятности сходится немного медленнее, но более высокая результативность достигается в долгосрочной перспективе. Однако, как видно из примера 13.1, даже при оптимальном значении α схождение получается не таким хорошим, как в алгоритме одномерного маржинального распределения (UMDA). Тем не менее следует отметить, что в каждом поколении компактный генетический алгоритм требует только двух новых особей и только одно сравнение функции приспособленности. Поэтому не совсем справедливо сравнивать алгоритм одномерного маржинального распределения и компактный генетический алгоритм с равным числом поколений; вместо этого их следует сравнивать с равным числом оцениваний функции приспособленности (см. задачи 13.3 и 13.12).

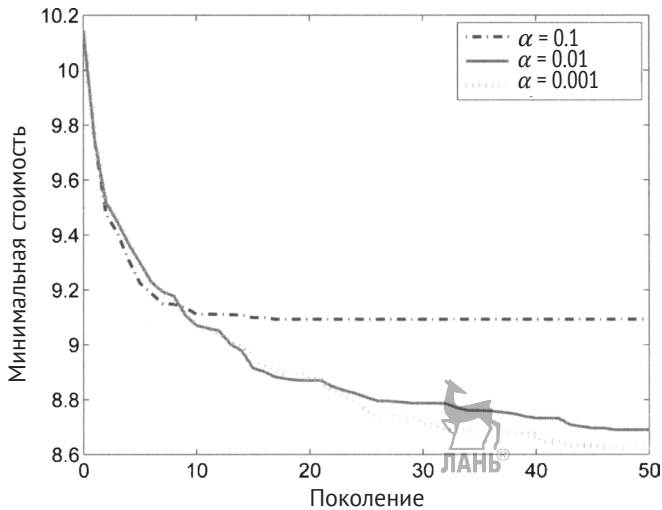


Рис. 13.5. Пример 13.2: результаты работы компактного генетического алгоритма для минимизации 20-мерной функции Экли с шестью битами на размерность. Результаты показывают лучшую особь в каждом поколении, усредненно по 50 симуляциям Монте-Карло

■

В примере 13.2 показаны результаты только с элитарностью, но читатель может подтвердить своими экспериментами, что устранение элитарности приводит к слабой результативности компактного генетического алгоритма. p_{min} и p_{max} также могут оказывать сильное влияние на результативность данного алгоритма. Наконец, обратите внимание, что ничто не мешает нам использовать более двух особей на поколение. Если мы создаем более двух особей на поколение, то можем модифицировать вектор вероятности, сравнивая лучшую особь с худшей особью. См. рис. 13.6 относительно обобщенной версии компактного генетического алгоритма.

Инициализировать n-элементный вектор вероятности $p = [0.5, ..., 0.5]$
Назначить p_{min} и p_{max}, минимальное и максимальное значения по каждому элементу p
Определить α, размер обновления вероятности
Определить N, размер популяции
Инициализировать элитарную особь $x_e \leftarrow \emptyset$ (нулевой вектор)
While not (критерий останова)
  For $i = 1$ **to** N (размер популяции)
    For $k = 1$ **to** n (число бит в каждом кандидатном решении)
      $r \leftarrow U[0, 1]$
      If $r < p(k)$
        $x_i(k) \leftarrow 1$
      else
        $x_i(k)) \leftarrow 0$
      End if
    Next бит
  Next особь
  $x_{лучший} \leftarrow$ лучший из $\{x_e, x_1, ..., x_N\}$
  $x_{худший} \leftarrow$ худший из $\{x_e, x_1, ..., x_N\}$
  For $k = 1$ **to** n (число бит в каждом кандидатном решении)
    If $x_{лучший}(k) \neq x_{худший}(k)$ **then**
      If $x_{лучший}(k) = 1$ **then**
        $p(k) \leftarrow p(k) + \alpha$
      else
        $p(k) \leftarrow p(k) - \alpha$
      End if
    $p(k) \leftarrow \max(\min(p(k), p_{max}), p_{min})$
    End if
  Next бит

$x_e \leftarrow x_{\text{лучший}}$

Next поколнение

Рис. 13.6. Обобщенная версия компактного генетического алгоритма (cGA) с элитарностью для оптимизации в n-битной двоичной области. $U[0, 1]$ – это случайное число, генерируемое из равномерного распределения между 0 и 1. $\alpha \in (0, 1)$ управляет скоростью схождения. $x_i(k)$ – это k-й бит в i-й особи

■ Пример 13.3

В данном примере мы снова пытаемся минимизировать 20-мерную функцию Экли, на этот раз с помощью обобщенного компактного генетического алгоритма рис. 13.6. Параметры задачи такие же, как и в примере 13.2, но мы задаем $\alpha = 0.001$. Мы испытали три разных значения для размера популяции: $N = 2$ (данное значение используется в компактном генетическом алгоритме по умолчанию), $N = 5$ и $N = 20$. На рис. 13.7 показана результативность компактного генетического алгоритма, усредненно по 50 симуляциям Монте-Карло. Мы видим, что по мере увеличения размера популяции его результативность улучшается. Однако вычислительные усилия прямо пропорциональны размеру популяции. Вместо сравнения с равным числом поколений в более справедливом сравнении будет использоваться равное число оцениваний функции (см. задачи 13.4 и 13.13).

■

13.2.3. *Популяционное инкрементное самообучение (PBIL)*

В этом разделе представлен алгоритм популяционного инкрементного самообучения (population based incremental learning, PBIL), представляющий собой алгоритм оценивания вероятностного распределения, в котором используются статистические показатели первого порядка. Алгоритм популяционного инкрементного самообучения является обобщением алгоритма одномерного маржинального распределения (UMDA). Данный алгоритм был описан в публикациях [Baluja, 1994], [Baluja и Caruana, 1995]. Он еще носит название восхождения к вершине холма с самообучением (hill climbing with learning, HCwL) [Kvasnicka и соавт., 1996] и инкрементного алгоритма одномерного маржинального распределения (incremental univariate marginal distribution algorithm (IUMDA) [Mihlenbein и Schlierkamp-Voosen, 1997]. При наличии n-мер-

ной бинарной оптимизационной задачи алгоритм популяционного инкрементного самообучения поддерживает n-мерный вектор вероятности p. k-й элемент p обозначает вероятность того, что k-й элемент кандидатного решения будет равен 1. Данный алгоритм мотивирован конкурентным самообучением, которое представляет собой простой метод самообучения в искусственных нейронных сетях [Fausett, 1994].

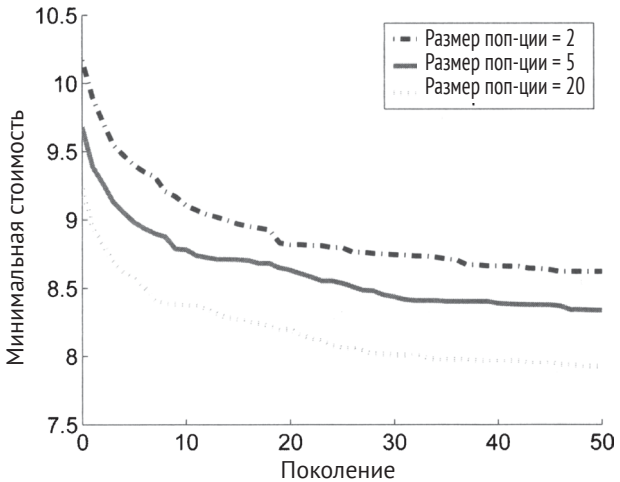


Рис. 13.7. Пример 13.3: результаты работы обобщенного компактного генетического алгоритма для минимизации 20-мерной функции Экли с шестью битами на размерность. Результаты показывают лучшую особь в каждом поколении, усредненно по 50 симуляциям Монте-Карло. Результативность обобщенного компактного генетического алгоритма и вычислительные усилия прямо пропорциональны размеру популяции

В каждом поколении мы используем вероятностный вектор p для вероятностного генерирования случайного множества кандидатных решений. Затем мы проверяем приспособленность каждого решения. Потом корректируем вектор вероятности так, чтобы следующее поколение с большей вероятностью было более похожим на наиболее приспособленных особей и менее похожим на наименее приспособленных особей. Получив этот новый вектор вероятности, мы переходим к следующему поколению, используя p для создания еще одной случайной популяции кандидатных решений. Этот процесс продолжается до тех пор, пока не будет удовлетворен определяемый пользователем критерий схождения. На рис. 13.8 представлен базовый алгоритм популяционного инкрементного самообучения для n-мерной бинарной оптимизационной задачи.

N = размер популяции
$N_{\text{лучшие}}$ = число хороших особей, используемых для корректировки p
$N_{\text{худшие}}$ = число плохих особей, используемых для корректировки p
$p_{\max} \in [0, 1]$ = максимально допустимое значение p
$p_{\min} \in [0, 1]$ = минимально допустимое значение p
N = скорость самообучения $\in (0, 1)$
Инициализировать n-элементный вектор вероятности $p = [0.5, ..., 0.5]$.
While not (критерий останова)
  Применить p для случайной генерации N особей $\{x_i\}$ следующим образом:
  For $i \in [1, N]$ (для каждой особи)
    For $k \in [1, n]$ (для каждого бита)
      $r \leftarrow$ случайное число в $U[0, 1]$
      If $r < p_k$
        $x_i(k) \leftarrow 0$
      else
        $x_i(k) \leftarrow 1$
      End if
    Next размерность k
  Next особь i
  Отсортировать особи так, чтобы $f(x_1) \le f(x_2) \le ... \le f(x_N)$.
  For $i \in [1, N_{\text{лучшие}}]$
    $p \leftarrow p + \eta(x_i - p)$
  Next i
  For $i \in [N - N_{\text{худшие}} + 1, N]$
    $p \leftarrow p - \eta(x_i - p)$
  Next i
  Вероятностно мутировать p
  $p(k) \leftarrow \max(\min(p, p_{\max}), p_{\min})$
Next поколение

Рис. 13.8. Простой алгоритм популяционного инкрементного самообучения (PBIL) для минимизации $f(x)$, где предметная область имеет n двоичных размерностей и $x_i(k) \in \{0, 1\}$ – это k-й элемент i-й особи

Рисунок 13.8 показывает, что в алгоритме популяционного инкрементного самообучения существует несколько регулировочных параметров.

- Мы должны принять решение о размере популяции N, как и во всех других эволюционных алгоритмах.
- Мы должны выбрать $N_{\text{лучшие}}$ и $N_{\text{худшие}}$, которые представляют собой число особей, используемых для модифицирования вектора вероятности в каждом поколении. Большие значения (близкие к

$N/2$) для этих параметров приведут к относительно застойному, медленному процессу эволюции. Малые значения (1 или немного больше) приведут к агрессивному процессу самообучения.

- Мы должны выбрать скорость самообучения η. Этот параметр имеет эффект, противоположный эффекту $N_{\text{лучшие}}$ и $N_{\text{худшие}}$. Небольшое значение η приведет к медленной оптимизации, и большое значение p приведет к быстрой оптимизации. Если оптимизация выполняется слишком быстро, то алгоритм популяционного инкрементного самообучения будет иметь тенденцию проскакивать мимо оптимального значения.
- Мы должны определиться с алгоритмом мутации, как и со всеми другими эволюционными алгоритмами (см. раздел 8.9).

По рис. 13.8 видно, что мы не сохраняем популяцию из поколения в поколение. Мы отслеживаем вектор вероятности и создаем популяцию в каждом поколении, но в каждом поколении популяция создается полностью заново.

На рис. 13.8 показано, что вектор вероятности корректируется таким образом, что последующие поколения особей будут с большей вероятностью похожи на высоко приспособленных особей. И наоборот, вектор вероятности корректируется таким образом, что последующие поколения с меньшей вероятностью будут похожи на низко приспособленных особей. Эта идея проиллюстрирована для двумерного случая на рис. 13.9, где мы исходим из того, что x_1 – это высоко приспособленная особь и x_N – низко приспособленная особь. Обратите внимание на рис. 13.9; если добавить к p некий результат умножения $(x_1 - p)$, то в результате получится новый p, который придвинется к x_1:

$$\begin{aligned} p_{\text{new}} &\leftarrow p + \eta(x_1 - p) \\ ||p_{\text{new}} - x_1||_2 &< ||p - x_1||_2 \end{aligned} \tag{13.5}$$

где $\eta \in (0, 1)$ – это скорость самообучения. И наоборот, рис. 13.9 показывает, что если из p вычесть некий результат умножения $(x_N - p)$, то в результате получится новый p, который отодвинется от x_N:

$$\begin{aligned} p_{\text{new}} &\leftarrow p + \eta(x_N - p) \\ ||p_{\text{new}} - x_N||_2 &< ||p - x_N||_2 \end{aligned} \tag{13.6}$$

В алгоритме популяционного инкрементного самообучения объединены уравнения (13.5) и (13.6) для нескольких кандидатных особей ($N_{\text{лучшие}}$ хороших особей и $N_{\text{худшие}}$ плохих особей) в поисковом пространстве, которое, как правило, имеет высокую размерность. В резуль-

тате вектор вероятности приближается к хорошим особям и отдаляется от плохих. Последующие поколения, скорее всего, будут ближе к хорошим особям и дальше от плохих.

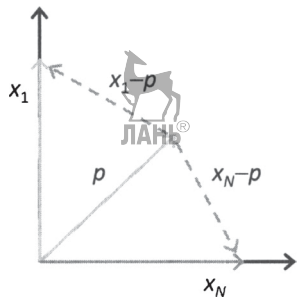


Рис. 13.9. Корректировка вектора вероятности для двумерной оптимизационной задачи. Мы корректируем p, для того чтобы двигаться к x_1, то есть к хорошей особи. Мы корректируем p, чтобы отодвигаться от x_N, то есть от плохой особи

13.3. Алгоритмы оценивания вероятностных распределений на основе статистических показателей второго порядка

В идеале при создании следующего поколения мы хотели бы использовать все вероятностное распределение лучших M особей целиком. Однако это было бы неосуществимо с точки зрения вычислений, и поэтому мы ослабляем строгость ради простоты и практичности. В алгоритмах оценивания вероятностных распределений, таких как алгоритм одномерного маржинального распределения (UMDA), компактный генетический алгоритм (cGA) и алгоритм популяционного инкрементного самообучения (PBIL), которые обсуждались в предыдущем разделе, для генерации популяции используются статистические показатели только первого порядка; этим подчеркиваются простота и практичность над строгостью. В алгоритмах оценивания вероятностных распределений в данном разделе ради повышенной строгости принята некоторая дополнительная сложность, и поэтому для генерации популяции используются статистические показатели второго порядка. В разделе 13.3.1 обсуждается максимизация взаимной информации для кластеризации входных данных (MIMIC), в разделе 13.3.2 – объединение оптимизаторов с деревьями взаимной информации (COMIT), и в разделе 13.3.3 обсуждается алгоритм двумерного маржинального распределения (BMDA).

13.3.1. *Максимизация взаимной информации для кластеризации входных данных (MIMIC)*

В этом разделе рассматривается алгоритм максимизации взаимной информации для кластеризации входных данных (mutual information maximization for input clustering, MIMIC), представляющий собой алгоритм оценивания вероятностного распределения, где используются статистические показатели второго порядка и который был разработан Джереми де Бонэ, Чарльзом Исбеллом и Полом Виолой (Jeremy De Bonet, Charles Isbell и Paul Viola) в 1997 году [De Bonet et al, 1997]. Функция плотности вероятности (PDF) случайной особи x может быть записана как

$$\begin{aligned} p(x) &= p(x(1), x(2), \ldots, x(n)) \\ &= p(x(1) \mid x(2), x(3), \ldots, x(n))\ \ p(x(2) \mid x(3), x(4), \ldots, x(n)) \ldots \\ &= p(x(n-1) \mid x(n))\ \ p(x(n)) \end{aligned} \tag{13.7}$$

где $x(k)$ – это k-й бит, принадлежащий относительно приспособленному кандидатному решению. Например, если мы заметим, что бит 4 лучших M особей имеет 78%-ный шанс равняться 1 при битах 5, 8, 9, 14 и 15, равных соответственно 0, 1, 1, 1 и 0, то мы хотели бы использовать эту информацию при создании следующего поколения. Однако для задач, содержащих более нескольких бит, вычислительно нереально получить полное распределение. Поэтому в алгоритме одномерного маржинального распределения (UMDA), компактном генетическом алгоритме (cGA) и алгоритме популяционного инкрементного самообучения (PBIL) используются статистические показатели только первого порядка; они делают неявное допущение о том, что функция плотности вероятности популяции может быть аппроксимирована статистическими показателями первого порядка:

$$\hat{p}(x) = p(x(1))\ p(x(2)) \ldots p(x(n)) \approx p(x). \tag{13.8}$$

Алгоритм максимизации взаимной информации для кластеризации входных данных (MIMIC) пытается найти более качественную аппроксимацию, чем в уравнении (13.8):

$$\hat{p}(x) = p(x(k_1) \mid x(k_2))\ p(x(k_2) \mid x(k_3)) \ldots p(x(k_{n-1}) \mid x(k_n))\ p(x(k_n)), \tag{13.9}$$

где $(k_1, k_2, \ldots, k_n)$ – это перестановка $\{1, 2, \ldots, n\}$. Напомним, что перестановка (пермутация) – это просто переупорядочение целых чисел. Например, если $n = 5$, то оба ряда (4, 5, 1, 3, 2) и (5, 1, 2, 4, 3) являются

перестановками ряда {1, 2, 3, 4, 5}. Задача, которую пытается решить алгоритм MIMIC, – это определить такую перестановку $(k_1, k_2, \ldots, k_n)$, которая делает уравнение (13.9) как можно ближе к истинной функции плотности вероятности $p(x)$; затем каждое поколение данный алгоритм использует $\hat{p}(x)$ для создания новой популяции кандидатных решений.

Прежде чем мы сможем найти перестановку, которая минимизирует ошибку аппроксимации $\hat{p}(x)$, нам нужно определить ошибку аппроксимации. Сходство двух дискретных функций плотности вероятности $p(x)$ и $\hat{p}(x)$ можно квантифицировать с помощью расхождения (дивергенции) Кульбака-Лейблера [Bishop, 2006]:

$$\begin{aligned} D(p, \hat{p}) &= \sum_x p(x) \log_2 (p(x) / \hat{p}(x)) \\ &= \sum_x p(x) \left(\log_2 p(x) - \log_2 \hat{p}(x)\right) \\ &= \sum_x \left(p(x) \log_2 p(x) - p(x) \log_2 \hat{p}(x)\right) \end{aligned} \tag{13.10}$$

где сумма берется по всем точкам x, где $p(x)$ или $\hat{p}(x)$ ненулевые. Мы хотим минимизировать $D(p, \hat{p})$ по $\hat{p}$. Первый член в правой части уравнения (13.10) не является функцией от $\hat{p}$, поэтому наша функция стоимости может быть записана как

$$\begin{aligned} J(\hat{p}) &= -\sum_x p(x) \log_2 \hat{p}(x) \\ &= -\sum_x p(x) \log_2 [p(x(k_1) \mid x(k_2))\, p(x(k_2) \mid x(k_3)) \ldots p(x(k_{n-1}) \mid x(k_n))\, p(x(k_n))] \\ &= -\sum_x [p(x) \log_2 p(x(k_1) \mid x(k_2)) + p(x) \log_2 p(x(k_2) \mid x(k_3)) + \ldots + \\ &\qquad p(x) \log_2 p(x(k_{n-1}) \mid x(k_n)) + p(x) \log_2 p(x(k_n))] \\ &= -E\left[\log_2 p(x(k_1) \mid x(k_2))\right] - E\left[\log_2 p(x(k_2) \mid x(k_3))\right] - \ldots \\ &\quad -E\left[\log_2 p(x(k_{n-1}) \mid x(k_n))\right] - E\left[\log_2 p(x(k_n))\right] \end{aligned} \tag{13.11}$$

где мы заменили уравнение (13.9) на $\hat{p}(x)$. Наша стоимость теперь может быть записана как

$$J(\hat{p}) = h(k_1 \mid k_2) + h(k_2 \mid k_3) + \ldots + h(k_{n-1} \mid k_n) + h(k_n), \tag{13.12}$$

где энтропийные члены $h(\cdot)$ определяются следующим образом [Gray, 2011]:

$$h(k_i) = -E\,[\log_2 p(x(k_i))]$$
$$h(k_i \mid k_j) = -E\,[\log_2 p(x(k_i) \mid x(k_j))] \qquad (13.13)$$

Теперь наша задача выглядит яснее. Мы хотим найти перестановку $(k_1, k_2, \ldots, k_n)$ $\{1, 2, \ldots, n\}$ – такую, что объединенная энтропия в правой части уравнения (13.12) будет минимизирована. Энтропия одного бита может быть записана как

$$h(k_i) = -\sum_{\alpha} \Pr(x_i = \alpha) \log_2 \Pr(x(i) = \alpha). \qquad (13.14)$$

Условная энтропия бита k_i, при условии что бит k_j равен β, может быть записана как

$$h(x(k_i) \mid x(k_j) = \beta) = -\sum_{\alpha} \Pr(x(k_i) = \alpha \mid x(k_j) = \beta) \log_2 \Pr(x(k_i) = \alpha \mid x(k_j) = \beta). \qquad (13.15)$$

Наконец, условная энтропия бита k_i, при условии что имеется бит k_j, может быть записана как

$$h(x(k_i) \mid x(k_j)) = \sum_{\beta} h(x(k_i) \mid x(k_j) = \beta) \Pr(x(k_j) = \beta). \qquad (13.16)$$

■ Пример 13.4

Рассмотрим несколько простых примеров расчета энтропии. Предположим, что у нас есть четыре трехбитные эволюционно-алгоритмические особи:

$$x_1 = (0, 0, 0),\ x_2 = (0, 0, 0),\ x_3 = (1, 0, 0),\ x_4 = (1, 1, 0). \qquad (13.17)$$

Энтропия первого (левого) бита равна

$$\begin{aligned} h(1) &= -E\,[\log_2 p(x(1))] \\ &= -[\Pr(x(1) = 0) \log_2 \Pr(x(1) = 0) + \Pr(x(1) = 1) \log_2 \Pr(x(1) = 1)] \\ &= -[0.5 \log_2 0.5 + 0.5 \log_2 0.5] = 1 \end{aligned} \qquad (13.18)$$

Энтропия второго (среднего) бита равна

$$\begin{aligned} h(2) &= -E\,[\log_2 p(x(2))] \\ &= -[\Pr(x(2) = 0) \log_2 \Pr(x(2) = 0) + \Pr(x(2) = 1) \log_2 \Pr(x(2) = 1)] \\ &= -[0.75 \log_2 0.75 + 0.25 \log_2 0.25] = 0.81 \end{aligned} \qquad (13.19)$$

Энтропия третьего (правого) бита равна

$$\begin{aligned} h(3) &= -E\left[\log_2 p(x(3))\right] \\ &= -[\Pr(x(3)=0)\log_2\Pr(x(3)=0) + \Pr(x(3)=1)\log_2\Pr(x(3)=1)] \\ &= -[1\log_2 1 + 0\log_2 0] = 0 \end{aligned} \tag{13.20}$$

где мы использовали соглашение о том, что $0 \log_2 0 = 0$, которое основано на том, что $\lim_{z\to 0}(z \log_2 z) = 0$. Мы видим, что энтропия первого бита является максимально возможным значением, а это значит, что первый бит ничего не говорит нам о самой приспособленной особи в популяции. Основываясь на четырех особях, которые у нас есть, наиболее приспособленная особь с одинаковой вероятностью будет иметь либо 0, либо 1 в качестве левого бита. С другой стороны, энтропия третьего бита является минимально возможным значением, что означает, что третий бит содержит максимально возможную информацию о самой приспособленной особи в популяции. Основываясь на четырех особях уравнения (13.17), наиболее приспособленная особь имеет 100%-ный шанс иметь 0 в качестве самого правого бита.

Условные энтропии бита 1 могут быть вычислены как

$$\begin{aligned} h(x(1) \mid x(2)=0) &= -\Pr(x(1)=0 \mid x(2)=0)\log_2\Pr(x(1)=0 \mid x(2)=0) \\ &\quad -\Pr(x(1)=1 \mid x(2)=0)\log_2\Pr(x(1)=1 \mid x(2)=0) \\ &= -(2/3)\log_2(2/3) - (1/3)\log_2(1/3) = 0.92 \\ h(x(1) \mid x(2)=1) &= -\Pr(x(1)=0 \mid x(2)=1)\log_2\Pr(x(1)=0 \mid x(2)=1) \\ &\quad -\Pr(x(1)=1 \mid x(2)=1)\log_2\Pr(x(1)=1 \mid x(2)=1) \\ &= -0\log_2 0 - 1\log_2 1 = 0 \end{aligned} \tag{13.21}$$

Мы видим, что условная энтропия бита 1, при условии что бит 2 = 0 является относительно высоким, а это означает, что знание о том, что бит 2 = 0, немного говорит нам о значении бита 1. С другой стороны, условная энтропия бита 1, при условии что бит 2 = 1, является минимально возможным значением, что означает, что знание о том, что бит 2 = 1, дает нам 100%-ную уверенность в значении бита 1. Объединение этих результатов дает условную энтропию бита 1, при условии что имеется бит 2, как

$$\begin{aligned} h(x(1) \mid x(2)) &= h(x(1) \mid x(2)=0)\Pr(x(2)=0) + h(x(1) \mid x(2)=1)\Pr(x(2)=1) \\ &= (0.92)(3/4) + (0)(1/4) = 0.69 \end{aligned} \tag{13.22}$$

которая является взвешенной суммой двух условно-энтропийных членов для двух особей.

■

Мы хотим найти перестановку $\{1, 2, \ldots, n\}$, минимизирующую уравнение (13.12). Но существует много возможных перестановок $\{1, 2, \ldots, n\}$. В общем случае существует $n!$ перестановок $\{1, 2, \ldots, n\}$. Это число становится чрезвычайно большим даже при малых значениях n, поэтому исчерпывающий поиск оптимальной перестановки методом грубой силы невозможен. Вместо этого мы используем жадный алгоритм [De Bonet и соавт., 1997], который приближенно минимизирует уравнение (13.12) и быстро находит хорошую аппроксимацию для $\hat{p}(x)$. Жадный алгоритм показан на рис. 13.10. Первый шаг – найти бит k_n, который имеет наименьшую энтропию (бóльшую часть информации). Второй шаг – найти бит k_{n-1}, который имеет наименьшую условную энтропию для данного бита k_n. На каждом последующем шаге мы находим бит, который имеет наименьшую условную энтропию, заданную ранее обнаруженным битом, при этом обеспечивая, что один и тот же бит не будет использован более одного раза.

$k_n = \arg\min_j h(j)$
For $i = n - 1, n - 2, \ldots, 1$
  $k_i = \arg\min_j h(j \mid k_{i+1}) : j \in \{k_{i+1}, k_{i+2}, \ldots, k_n\}$
Next i

Рис. 13.10. Жадный алгоритм для приближенной минимизации уравнения (13.12)

Алгоритм MIMIC работает путем отбора подмножества высоко приспособленных особей из популяции. В нем для нахождения почти оптимального решения уравнения (13.12) используется жадный алгоритм, показанный на рис. 13.10. Затем он использует эти вероятности для генерации следующей популяции кандидатных решений. На рис. 13.11 представлен алгоритм MIMIC для оптимизации на двоичной области.

Инициализировать популяцию кандидатных решений $\{x_i\}$, $i \in [1, N]$.
Отметить, что каждый x_i включает n бит $x_i(1), \ldots x_i(n)$.
While not (критерий останова)
  Отобрать M особей из $\{x_i\}$ согласно приспособленности, где $M < N$.
  Проиндексировать M отобранных особей как $\{x_i\}$, $i \in [1, M]$.
  $k_n \leftarrow \arg\min_j h(j)$
  For $m = n - 1, n - 2, \ldots, 1$
    $k_m \leftarrow \arg\min_j h(x(j) \mid x(k_{m+1})) : j \notin \{k_n, k_{n-1}, \ldots, k_{m+1}\}$
  Next m
  $\Pr\{x_i(k) = 1) \leftarrow \sum_{i=1}^{M} \sigma(x_i(k_n) - 1) / M$
  For $m = n - 1, n - 2, \ldots, 1$
    Определить $1_{m+1} = \{ i \in [1, M] : x_i(k_{m+1}) = 1\}$

```
    Определить 0_{m+1} = { i ∈ [1, M] : x_i(k_{m+1}) = 0}
    Pr(x(k_m) = 1 | x(k_{m+1}) = 1) ← Σ_{i ∈ 1_{m+1}} σ(x_i(k_m) − 1) / |1_{m+1}|
    Pr(x(k_m) = 1 | x(k_{m+1}) = 0) ← Σ_{i ∈ 0_{m+1}} σ(x_i(k_m) − 1) / |0_{m+1}|
  Next m
  For i = 1 to N (размер популяции)
    r ← U[0, 1]
    If r < Pr(x(k_n) = 1) then x_i(k_n) ← 1; else x_i(k_n) ← 0
    For m = n − 1, n − 2, ..., 1
      r ← U[0, 1]
      If x_i(k_{m+1}) = 0
          If r < Pr(x(k_m) = 1 | x(k_{m+1}) = 0)
              x_i(k_m) ← 1
          else
              x_i(k_m) ← 0
          End if
      else
          If r < Pr(x(k_m) = 1 | x(k_{m+1}) = 1)
                x_i(k_m) ← 1
          else
                x_i(k_m) ← 0
          End if
      End if
    Next бит m
  Next особь i
Next поколение
```

Рис. 13.11. Схематичное представление алгоритма максимизации взаимной информации для кластеризации входных данных (MIMIC) для оптимизации в N-битной двоичной области. $h(y)$ – это энтропия функции плотности вероятности от y и эмпирически оценивается из кандидатных решений. $\sigma(y)$ – это дельта-функция Кронекера, то есть $\sigma(y) = 1$, если $y = 0$, и $\sigma(y) = 0$, если $y \neq 0$. $U[0, 1]$ – это случайное число, генерируемое из равномерного распределения между 0 и 1

Алгоритм MIMIC включает в себя много вероятностных вычислений. Во время его реализации мы хотим убедиться, что ни одна из этих вероятностей не равна 0 или 1, поскольку это приведет к невозможности полностью разведать поисковое пространство оптимизационной задачи. Поэтому во время реализации рис. 13.11 после вычисления каждой вероятности p мы, возможно, захотим ограничить значение вероятности:

$$\begin{aligned} p_i &\leftarrow \max(p_i, \epsilon) \\ p_i &\leftarrow \min(p_i, 1-\epsilon) \end{aligned} \tag{13.23}$$

где ϵ – это малый положительный регулировочный параметр, часто равный 0.01. Пример алгоритма MIMIC данных приведен далее в примере 13.7.

13.3.2. *Объединение оптимизаторов с деревьями взаимной информации (COMIT)*

Алгоритм объединения оптимизаторов с деревьями взаимной информации (combining optimizers with mutual information trees, COMIT) был описан в публикации [Baluja и Davies, 1998]. Данный алгоритм похож на алгоритм MIMIC. Однако в алгоритме MIMIC мы находим почти оптимальную перестановку $(k_1, k_2, \ldots, k_n)$ путем минимизации членов условной энтропии, как показано на рис. 13.10. Вместо этого в алгоритме COMIT мы находим почти оптимальную перестановку, максимизируя члены взаимной информации. Вместо того чтобы находить перестановку, которая минимизирует уравнение (13.12), мы находим перестановку, которая максимизирует уравнение

$$J_c(\hat{p}) = I(k_1 \mid k_2) + I(k_2 \mid k_3) + \ldots + I(k_{n-1} \mid k_n) - h(k_n). \tag{13.24}$$

Взаимная информация между битами k и m определяется следующим образом [Cover и Thomas, 1991]:

$$I(k, m) = \sum_{i,j} \Pr(x(k) = i, x(m) = j) \log_2 \left[\frac{\Pr(x(k) = i, x(m) = j)}{\Pr(x(k) = i) \Pr(x(m) = j)} \right], \tag{13.25}$$

где суммирование берется на $i \in [0, 1]$ и $j \in [0, 1]$.

■ Пример 13.5

В данном примере, который основан на публикации [Chow и Liu, 1968], мы иллюстрируем вычисление взаимной информации. Предположим, что мы выполняем эволюционный алгоритм на четырехбитной оптимизационной задаче. Из алгоритма у нас есть большое число особей, может быть, 100 или около того. Мы отбираем 20 относительно приспособленных особей. Предположим, что эти 20 особей заданы следующим образом:

$$
\begin{array}{llll}
x_1 = (0,0,0,0), & x_2 = (0,0,0,0) \\
x_3 = (0,0,0,1), & x_4 = (0,0,0,1) \\
x_5 = (0,0,1,0), & x_6 = (0,0,1,1) \\
x_7 = (0,1,1,0), & x_8 = (0,1,1,0) \\
x_9 = (0,1,1,1), & x_{10} = (1,0,0,0) \\
x_{11} = (1,0,0,1), & x_{12} = (1,0,0,1) \\
x_{13} = (1,1,0,0), & x_{14} = (1,1,0,1) \\
x_{15} = (1,1,1,0), & x_{15} = (1,1,1,0) \\
x_{17} = (1,1,1,0), & x_{18} = (1,1,1,1) \\
x_{19} = (1,1,1,1), & x_{20} = (1,1,1,1)
\end{array} \qquad (13.26)
$$

Битовые числа индексируются от 1 до 4 слева направо. Например, $x_{13}(1) = x_{13}(2) = 1$ и $x_{13}(3) = x_{13}(4) = 0$. Подсчитывая биты и следуя процедуре, приведенной в примере 13.4, мы находим следующее:

$$
\begin{array}{lll}
\Pr(x(1) = 1) = 0.55 & \rightarrow & h(1) = 0.993 \\
\Pr(x(2) = 1) = 0.55 & \rightarrow & h(2) = 0.993 \\
\Pr(x(3) = 1) = 0.55 & \rightarrow & h(3) = 0.993 \\
\Pr(x(4) = 1) = 0.50 & \rightarrow & h(4) = 1
\end{array} \qquad (13.27)
$$

Мы видим, что биты 1, 2 и 3 имеют одинаковое количество информации, в то время как бит 4 имеет наименьшее количество информации. Теперь мы вычисляем взаимную информацию между битом 1 и другими битами. Уравнение (13.25) показывает, что, прежде чем вычислять взаимную информацию, необходимо вычислить индивидуальные битовые вероятности $\Pr(x_i)$. Рассмотрим особей в уравнении (13.26). Уравнение (13.27) показывает следующее:

$$
\begin{array}{ll}
\Pr(x(1) = 0) = 0.45, & \Pr(x(1) = 1) = 0.55 \\
\Pr(x(2) = 0) = 0.45, & \Pr(x(2) = 1) = 0.55
\end{array} \qquad (13.28)
$$

Теперь обратите внимание, что в уравнении (13.26) есть шесть особей таких, что $x(1) = 0$ и $x(2) = 0$; три особи такие, что $x(1) = 0$ и $x(2) = 1$; три особи такие, что $x(1) = 1$ и $x(2) = 0$; и восемь особей таких, что $x(1) = 1$ и $x(2) = 1$. Поэтому

$$
\begin{array}{lll}
\Pr(x(1) = 0,\ x(2) = 0) & = & 0.30 \\
\Pr(x(1) = 0,\ x(2) = 1) & = & 0.15 \\
\Pr(x(1) = 1,\ x(2) = 0) & = & 0.15 \\
\Pr(x(1) = 1,\ x(2) = 1) & = & 0.40
\end{array} \qquad (13.29)
$$

Теперь мы применим уравнение (13.25) для вычисления взаимной информации между битами 1 и 2 следующим образом:

$$
\begin{aligned}
I(1,2) &= \sum_{i,j} \Pr(x(1)=i, x(2)=j) \log_2 \left[\frac{\Pr(x(1)=i, x(2)=j)}{\Pr(x(1)=i)\Pr(x(2)=j)} \right] \\
&= \Pr(x(1)=0, x(2)=0) \log_2 \left[\frac{\Pr(x(1)=0, x(2)=0)}{\Pr(x(1)=0)\Pr(x(2)=0)} \right] + \\
&\quad \Pr(x(1)=0, x(2)=1) \log_2 \left[\frac{\Pr(x(1)=0, x(2)=1)}{\Pr(x(1)=0)\Pr(x(2)=1)} \right] + \\
&\quad \Pr(x(1)=1, x(2)=0) \log_2 \left[\frac{\Pr(x(1)=1, x(2)=0)}{\Pr(x(1)=1)\Pr(x(2)=0)} \right] + \\
&\quad \Pr(x(1)=1, x(2)=1) \log_2 \left[\frac{\Pr(x(1)=1, x(2)=1)}{\Pr(x(1)=1)\Pr(x(2)=1)} \right] \\
&= 0.30 \log_2[0.30 / (0.45 \times 0.45)] + 0.15 \log_2[0.15 / (0.45 \times 0.55)] + \\
&\quad 0.15 \log_2[0.15 / (0.55 \times 0.45)] + 0.40 \log_2[0.40 / (0.55 \times 0.55)] \\
&= 0.1146
\end{aligned}
\tag{13.30}
$$

Для вычисления взаимной информации между другими битами мы используем тот же метод, и он приводит к следующему:

$$
\begin{aligned}
I(1,2) &= 0.1146 \\
I(1,3) &= 0.0001 \\
I(1,4) &= 0.0073 \\
I(2,3) &= 0.2727 \\
I(2,4) &= 0.0073 \\
I(3,4) &= 0.0073
\end{aligned}
\tag{13.31}
$$

■

Взаимная информация $I(i, j)$ количественно определяет, сколько информации передается между битами i и j. Она говорит нам, сколько мы можем узнать о значении одного бита, если знаем значение другого. Если дан бит j, то максимизация взаимной информации $I(i, j)$ на всех i аналогична минимизации условной энтропии $h(i \mid j)$ на всех i. Поэтому алгоритм COMIT может выполняться с помощью такого же алгоритма, что и алгоритм MIMIC, но, вместо того чтобы использовать рис. 13.10 для отбора перестановки, мы используем рис. 13.12. Следовательно, алгоритм COMIT является таким же, что и алгоритм MIMIC на рис. 13.11, за исключением того, что первый цикл «For $m = n - 1, n - 2, \ldots, 1$» на рис. 13.11 заменен на рис. 13.12.

$k_n = \arg\min_j h(j)$
For $i = n - 1, n - 2, ..., 1$
  $k_i = \arg\max_j I(j, k_{i+1}) : j \in \{k_{i+1}, k_{i+2}, ..., k_n\}$
Next i

Рис. 13.12. Жадный алгоритм приближенной максимизации уравнения (13.25). Сравните с рис. 13.10

■ Пример 13.6

В данном примере, который является продолжением примера 13.5, мы иллюстрируем жадный алгоритм рис. 13.12. Жадный алгоритм сначала находит бит с наибольшим количеством информации, что эквивалентно нахождению бита с наименьшей энтропией. В уравнении (13.27) в примере 13.5 мы увидели, что биты 1, 2 и 3 имеют одинаковое количество информации, в то время как бит 4 имеет наименьшее количество информации. Следовательно, решение задачи $k_n = \arg\min_j h(j)$ на рис. 13.12 равно 1, 2 или 3. Мы произвольно отбираем бит 1 в качестве решения. Теперь мы находим бит, который делится большей информацией с битом 1; уравнение (13.31) показывает нам, что бит 2 делится большей информацией с битом 1. Теперь мы находим бит (не включая бит 1), который делится большей информацией с битом 2; уравнение (13.31) показывает нам, что бит 3 делится большей информацией с битом 2. Бит 4 является единственным оставшимся битом, и поэтому жадный алгоритм завершен, а уравнение (13.9) становится

$$\hat{p}(x) = p(x(1))\, p(x(2) \mid x(1))\, p(x(3) \mid x(2))\, p(x(4) \mid x(3)). \qquad (13.32)$$

Мы можем использовать уравнение (13.26) для расчета этих вероятностей. В табл. 13.1 показаны вероятность каждой битовой комбинации, оцененная вероятность с использованием аппроксимации первого порядка уравнения (13.8) и оцененная вероятность из уравнения (13.32).

Таблица 13.1. Результаты примера 13.6: истинные вероятности (второй столбец) и оценочные вероятности (третий и четвертый столбцы) всех возможных комбинаций бит

$x(1)x(2)x(3)x(4)$	$p(x(1), x(2), x(3), x(4))$	UMDA: $p(x(1))\, p(x(2)) \bullet p(x(3))\, p(x(4))$	COMIT: $p(x(1))\, p(x(2) \mid x(1)) \bullet p(x(3) \mid x(2))\, p(x(4) \mid x(3))$
0000	0.100	0.0456	0.1037
0001	0.100	0.0456	0.1296
0010	0.050	0.0557	0.0364

$x(1)x(2)x(3)x(4)$	$p(x(1), x(2), x(3), x(4))$	UMDA: $p(x(1))\,p(x(2)) \bullet p(x(3))\,p(x(4))$	COMIT: $p(x(1))\,p(x(2) \mid x(1)) \bullet p(x(3) \mid x(2))\,p(x(4) \mid x(3))$
0011	0.050	0.0557	0.0303
0100	0.000	0.0557	0.0121
0101	0.000	0.0557	0.0152
0110	0.100	0.0681	0.0669
0111	0.050	0.0681	0.0558
1000	0.050	0.0557	0.0519
1001	0.100	0.0557	0.0648
1010	0.000	0.0681	0.0182
1011	0.000	0.0681	0.0152
1100	0.050	0.0681	0.0323
1101	0.050	0.0681	0.0404
1110	0.150	0.0832	0.1785
1111	0.150	0.0832	0.1488

Беглый взгляд на цифры в табл. 13.1 показывает, что истинная вероятность второго столбца оценивается точнее четвертым столбцом, чем третьим столбцом. Мы можем использовать уравнение (13.10) для квантификации схожести распределений вероятностей. Мы находим, что вероятности во втором и третьем столбцах имеют меру близости 0.53, в то время как вероятности во втором и четвертом столбцах имеют меру близости 0.14.

Обратите внимание, что в алгоритме COMIT мы не используем вероятности табл. 13.1. Мы только показываем их в этом примере, чтобы проиллюстрировать эффективность жадного алгоритма рис. 13.12. Алгоритм COMIT использует вероятности в правой части уравнения (13.32) для генерации популяции в следующем поколении.

■

В примере 13.6 показано, как найти аппроксимацию второго порядка для распределения вероятностей, используя критерий взаимной информации (COMIT), а не критерий условной энтропии (MIMIC). Если мы получим оценку распределения вероятностей высоко приспособленных особей в эволюционном алгоритме, то сможем применить эту оценку для генерации кандидатных решений в каждом поколении. В этом за-

ключается суть работы алгоритма объединения оптимизаторов с деревьями взаимной информации (COMIT), и это проиллюстрировано в следующем далее примере.

■ Пример 13.7

В данном примере мы используем алгоритмы MIMIC и COMIT на рис. 13.10, 13.11 и 13.12 для минимизации функции Экли, которая определена в приложении С.1.2. Мы применяем шесть бит на размерность, поэтому задача минимизации включает $n = 6D$ бит, где D – это размерность функции Экли. Применяем область [–5, +5] для каждой размерности, которая дает нам разрешающую способность $10/(2^6 - 1) = 0.16$ для каждой размерности. Мы используем популяцию размером $N = 100$, $M = 40$ и параметр элитарности 2, что означает, что мы всегда оставляем двух лучших особей из поколения в поколение. Мы используем $\epsilon = 0.01$ в уравнении (13.23).

Для изучения результативности алгоритма MIMIC сравниваем его с алгоритмом одномерного маржинального распределения (UMDA) (см. рис. 13.2), в котором используются статистические показатели только первого порядка. Мы также реализуем алгоритм MIMIC, в котором перестановка $(k_1, k_2, \ldots, k_n)$ назначается случайно, а не с помощью жадного алгоритма рис. 13.10. На рис. 13.13 показана результативность алгоритмов MIMIC, UMDA и MIMIC со случайной перестановкой на двумерной функции Экли, усредненно по 20 симуляциям Монте-Карло. Мы видим, что использование статистических показателей второго порядка превосходит алгоритм UMDA первого порядка, даже если перестановка случайна. Вместе с тем алгоритм MIMIC работает лучше всего, потому что в нем используется почти оптимальный жадный алгоритм для определения перестановки битовых индексов.

На рис. 13.14 показана результативность алгоритмов MIMIC, UMDA и MIMIC со случайной перестановкой на 10-мерной функции Экли, усредненно по 20 симуляциям Монте-Карло. Мы видим, что алгоритм MIMIC лучше всего работает в течение первых нескольких поколений, но после нескольких поколений алгоритм UMDA и алгоритм MIMIC со случайной перестановкой догоняют и превосходят алгоритм MIMIC. Это показывает, что нет никаких гарантий, что алгоритм MIMIC будет работать лучше, чем алгоритмы первого порядка, но он может быть ценным инструментом оптимизации в зависимости от конкретной задачи.

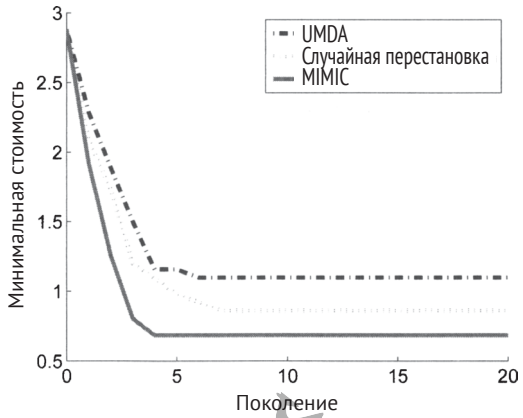


Рис. 13.13. Пример 13.7: результаты алгоритмов UMDA и MIMIC для минимизации двумерной функции Экли с шестью битами на размерность. Результаты показывают стоимость лучшей особи в каждом поколении, усредненно по 20 симуляциям Монте-Карло

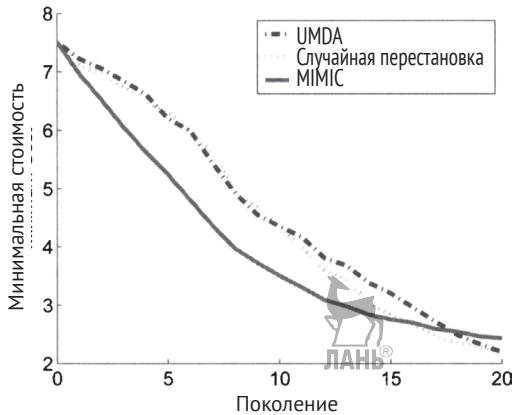


Рис. 13.14. Пример 13.7: результаты алгоритмов UMDA и MIMIC для минимизации 10-мерной функции Экли с шестью битами на размерность. Результаты показывают стоимость лучшей особи в каждом поколении, усредненно по 20 симуляциям Монте-Карло

Наконец, мы сравниваем алгоритмы COMIT и MIMIC. Оба алгоритма одинаково описаны на рис. 13.11, за исключением того, что в алгоритме MIMIC используется жадный алгоритм на рис. 13.10, а в алгоритме COMIT – жадный алгоритм на рис. 13.12, для того чтобы решить, какие битовые индексы спаривать для генерации кандидатных решений в каждом поколении. На рис. 13.15 показана результативность алгоритмов

COMIT и MIMIC на 10-мерной функции Экли, усредненно по 20 симуляциям Монте-Карло. Мы видим, что в предыдущих поколениях алгоритм COMIT работает гораздо лучше. Вместе с тем примерно через 15 поколений алгоритм MIMIC догоняет и превосходит результативность алгоритма COMIT.

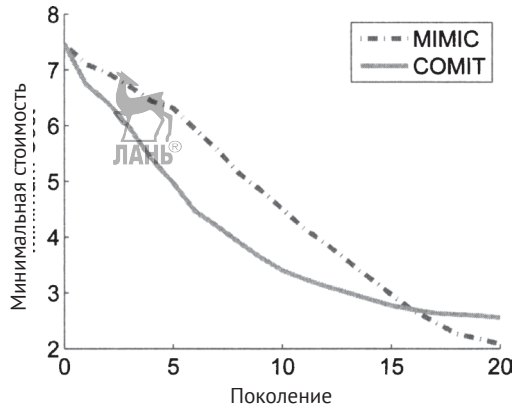


Рис. 13.15. Пример 13.7: результаты работы алгоритмов MIMIC и COMIT для минимизации 10-мерной функции Экли с шестью битами на размерность. Результаты показывают стоимость лучшей особи в каждом поколении, усредненно по 20 симуляциям Монте-Карло

■

13.3.3. *Алгоритм двумерного маржинального распределения (BMDA)*

Алгоритм двумерного маржинального распределения (bivariate marginal distribution algorithm, BMDA) был представлен в публикации [Pelikan и Mühlenbein, 1998]. В алгоритме двумерного маржинального распределения, как и в алгоритмах MIMIC и COMIT, используются статистические показатели второго порядка. Однако он имеет несколько существенных отличий. Во-первых, в нем используются проверки на основе статистического показателя хи-квадрат Пирсона [Boslaugh и Watters, 2008] для установления связей между взаимозависимыми битами. Во-вторых, он создает аппроксимацию функции плотности вероятности, которая не обязательно использует все биты в одной связной цепи. Вспомним из уравнения (13.9), что алгоритмы MIMIC и COMIT находят перестановку $(k_1, k_2, \ldots, k_n)$ $\{1, 2, \ldots, n\}$ (где n – это число бит в x) такую, что

$$p(x) \approx p(x(k_1) \mid x(k_2))\, p(x(k_2) \mid x(k_3)) \ldots p(x(k_{n-1}) \mid x(k_n))\, p(x(k_n)). \qquad (13.33)$$

Приведенная выше аппроксимация может рассматриваться как единая цепочка значений k_i:

$$k_1 \to k_2 \to \ldots \to k_{n-1} \to k_n. \tag{13.34}$$

Алгоритм двумерного маржинального распределения (BMDA), с другой стороны, находит более общую аппроксимацию $p(x)$:

$$p(x) \approx \prod_{x(r)\in R} p(x(r)) \prod_{x(i)\in V\setminus R} p(x_i(i) \mid x(m_i)). \tag{13.35}$$

В приведенной выше аппроксимации R – это множество корневых битовых индексов, которое определяется алгоритмом BMDA, и V – множество всех битовых индексов, то есть $V = \{1, \ldots, n\}$. Биты $x(i)$ принадлежат $V\setminus R$, то есть множеству всех индексов, которые не принадлежат R, то есть $V\setminus R = \{i \in V : i \notin R\}$. Наконец, $m(i)$ – это битовый индекс, определяемый алгоритмом BMDA, который имеет высокую степень зависимости от бита i.

Пример может прояснить интерпретацию уравнения (13.35). Предположим, у нас в поисковой области есть девять бит. Уравнение (13.33) используется алгоритмами MIMIC и COMIT и может привести к цепочке

$$3 \to 9 \to 1 \to 5 \to 8 \to 2 \to 4 \to 7 \to 6 \tag{13.36}$$

которая дает аппроксимацию

$$\begin{aligned} p(x) \approx\ & p(x(3) \mid x(9))\, p(x(9) \mid x(1))\, p(x(1) \mid x(5))\, p(x(5) \mid x(8)) \times \\ & p(x(8) \mid x(2))\, p(x(2) \mid x(4))\, p(x(4) \mid x(7))\, p(x(7) \mid x(6))\, p(x(6)) \end{aligned} \tag{13.37}$$

Уравнение (13.35), использующееся алгоритмом BMDA, может привести к цепочкам

$$\begin{aligned} & 3 \to 9 \to 1 \\ & 3 \to 5 \\ & 8 \to 2 \to 4 \to 7 \\ & 8 \to 6 \end{aligned} \tag{13.38}$$

которые дают аппроксимацию

$$\begin{aligned} p(x) \approx\ & p(x(3))\, p(x(9) \mid x(3))\, p(x(1) \mid x(9))\, p(x(5) \mid x(3)) \times \\ & p(x(8))\, p(x(2) \mid x(8))\, p(x(4) \mid x(2))\, p(x(7) \mid x(4))\, p(x(6) \mid x(8)) \end{aligned} \tag{13.39}$$

Алгоритм BMDA определил корневые битовые индексы 3 и 8, а также многочисленные цепочки битовых индексов для каждого корня.

Алгоритм BMDA работает, сначала выбирая случайный битовый индекс p в качестве первого корневого битового индекса. Затем вычисляется статистический показатель хи-квадрат для корневого битового

индекса r и всех остальных битов. Статистический показатель хи-квадрат между битами r и j вычисляется как

$$\chi^2_{rj} = M \sum_{\alpha,\beta} \frac{\left[\Pr(x(r)=\alpha, x(j)=\beta) - \Pr(x(r)=\alpha)\Pr(x(j)=\beta)\right]^2}{\Pr(x(r)=\alpha)\Pr(x(j)=\beta)}, \qquad (13.40)$$

где M – это число битовых строк, и суммирование берется по всем значениям α таким, что $\Pr(x(r) = \alpha) \neq 0$, и по всем значениям β таким, что $\Pr(x(j) = \beta) \neq 0$. Статистический показатель χ^2_{rj} измеряет зависимость между битами r и j. Высокое значение χ^2_{rj} указывает на то, что существует высокая степень корреляции между битами r и j. В науке статистике $\chi^2_{rj} < 3.84$ часто используется как пороговое значение для независимости бит r и j. Это значение χ^2 означает, что существует 95%-ная вероятность того, что битовые значения являются независимыми.

После того как алгоритм BMDA вычисляет χ^2_{rj} для корневого бита r и всех остальных бит j, он отбирает j с наибольшим значением χ^2_{rj} в качестве следующиго бита в цепи. Затем алгоритм BMDA вычисляет χ^2_{rk} и χ^2_{jk} для всех $k \neq \{r, j\}$. Любой k, который имеет самый высокий статистический показатель χ^2, становится следующим битом в цепочке вслед за битом r или битом j. Этот процесс продолжается до тех пор, пока все статистические показатели χ^2 не будут ниже некоторого порога. Когда это происходит, выбирается еще один случайный корневой бит, и процесс повторяется для следующей цепочки. Когда все биты будут использованы, аппроксимирование вероятности будет завершено. Алгоритм BMDA обобщен на рис. 13.16 [Pelikan и Muhlenbein, 1998].

(1) $V \leftarrow \{1, 2, ..., n\}$
$A \leftarrow V$

(2) $v \leftarrow$ случайно выбранный элемент из A
Добавить $\Pr(v)$ в аппроксимацию функции плотности вероятности (PDF)

(3) Удалить v и A
Если $A = \emptyset$, то завершить

(4) Вычислить χ^2_{ij} для всех $i \in A$ и всех $j \in V \backslash A$
Если $\max_{i,j} \chi^2_{ij} < 3.84$, то перейти в (2)

(5) $\{v, v'\} = \arg\max_{i,j} \chi^2_{ij} : i \in A, j \in V \backslash A$
Добавить $\Pr(v \mid v')$ в аппроксимацию функции плотности вероятности
Перейти в (3)

Рис. 13.16. Схематичное представление алгоритма двумерного маржинального распределения (BMDA) для генерации n-мерной аппроксимации функции плотности вероятности (PDF). V – это постоянное множество всех битовых индексов, и A – множество доступных битовых индексов для включения в цепочку бит. Значение 3.84 используется здесь в качестве 95%-го уровня доверия для независимости

■ Пример 13.8

Рассмотрим несколько простых примеров вычисления статистического показателя χ^2. Предположим, что у нас есть четыре четырехбитных эволюционных алгоритма:

$$x_1 = (0, 0, 0, 1), x_2 = (0, 0, 0, 1), x_3 = (1, 0, 0, 0), x_4 = (1, 1, 0, 0). \quad (13.41)$$

Мы находим маржинальные вероятности бит 1 и 2 как

$$\begin{aligned} &\Pr(x(1) = 0) = 1/2, \quad \Pr(x(1) = 1) = 1/2 \\ &\Pr(x(2) = 0) = 3/4, \quad \Pr(x(2) = 1) = 1/4 \end{aligned} \quad (13.42)$$

где мы определяем $x(1)$ (бит 1) в качестве левого бита, $x(2)$ (бит 2) в качестве следующего бита и так далее. Найдем совместные вероятности бит 1 и 2 как

$$\begin{aligned} &\Pr(x(1) = 0,\ x(2) = 0) = 1/2, \quad \Pr(x(1) = 0,\ x(2) = 1) = 0 \\ &\Pr(x(1) = 1,\ x(2) = 0) = 1/4, \quad \Pr(x(1) = 1,\ x(2) = 1) = 1/4. \end{aligned} \quad (13.43)$$

Статистический показатель χ^2_{12} вычисляется из уравнения (13.40) как

$$\chi^2_{12} = 4(1/24 + 1/8 + 1/24 + 1/8) = 4/3. \quad (13.44)$$

Мы видим, что существует некоторая связь между битами 1 и 2, но у нас так мало выборок (только четыре), что мы не можем с большой уверенностью сказать, что зависимость статистически значима, то есть $\chi^2_{12} < 3.84$.

Теперь рассмотрим биты 1 и 3. В этом случае статистический показатель χ^2_{13} вычисляется как

$$\chi^2_{13} = 4(0 + 0 + 0 + 0) = 0. \quad (13.45)$$

Между битами 1 и 3 нет никакой связи. Мы видим это, глядя на уравнение (13.41); бит 1 равен 0 в половине случаев и равен 1 в половине случаев, а бит 3 всегда равен 0.

Наконец, рассмотрим биты 1 и 4. В этом случае статистический показатель χ^2_{14} вычисляется как

$$\chi^2_{14} = 4(1 + 1 + 1 + 1) = 4. \quad (13.46)$$

Мы видим, что между этими двумя битами существует статистически значимая связь. Несмотря на то что у нас только четыре особи, так как биты 1 и 4 всегда дополняют друг друга, мы можем быть совершенно уверены, что они зависят друг от друга.

■

13.4. Алгоритмы оценивания многомерных вероятностных распределений

Мы видели, что алгоритмы оценивания вероятностных распределений с использованием статистических показателей первого порядка подчеркивают простоту над математической строгостью. Алгоритмы оценивания вероятностных распределений с использованием статистических показателей второго порядка уделяют больше внимания математической строгости, что приводит к более сложным, но потенциально более эффективным алгоритмам. Алгоритмы оценивания многомерных вероятностных распределений делают еще один шаг в этом направлении, используя статистические показатели, которые даже выше второго порядка, и в этом разделе мы представляем один такой алгоритм: расширенный компактный генетический алгоритм (ECGA).

13.4.1. *Расширенный компактный генетический алгоритм (ECGA)*

Как следует из названия, расширенный компактный генетический алгоритм (extended compact genetic algorithm, ECGA) является обобщением компактного генетического алгоритма (cGA) раздела 13.2.2. Расширенный компактный генетический алгоритм был предложен в публикации [Harik, 1999] и далее разъясняется в публикациях [Sastry и Goldberg, 2000], [Lima и Lobo, 2004], [Harik и соавт., 2010]. Алгоритм ECGA пытается найти распределение вероятностей, удовлетворяющее двум свойствам: во-первых, оно простое; и во-вторых, оно точно аппроксимирует распределение вероятностей множества высоко приспособленных особей. Приближенное распределение вероятностей называется моделью маржинального продукта (marginal product model, MPM). Пример модели маржинального продукта $\hat{p}(x)$ для 10-мерной задачи выглядит следующим образом:

$$\hat{p}(x) = p(x(1), x(3), x(6))\, p(x(2))\, p(x(4), x(5), x(7), x(10))\, p(x(8), x(9)). \qquad (13.47)$$

Переменные в каждом маржинальном распределении на правой стороне этого уравнения называются строительным блоком, поэтому приведенная выше модель маржинального продукта имеет четыре строительных блока: ($x(1)$, $x(3)$, $x(6)$), ($x(2)$), ($x(4)$, $x(5)$, $x(7)$, $x(10)$) и ($x(8)$, $x(9)$). Число переменных в i-м строительном блоке BB модели маржинального продукта называется длиной L_i строительного блока. Поэтому модель

маржинального продукта уравнения (13.47) имеет длины строительных блоков

$$L_1 = 3, \quad L_2 = 1, \quad L_3 = 4, \quad L_4 = 2. \tag{13.48}$$

Переменные в строительных блоках различны, поэтому если $x(k)$ принадлежит B_i, то она не принадлежит B_j для любого $j \neq i$. Сложность модели маржинального продукта определяется как

$$C_m = (\log_2(M+1)) \sum_{i=1}^{N_b} (2^{L_i} - 1), \tag{13.49}$$

где M – это число высоко приспособленных особей, отобранных из популяции, N_b – число строительных блоков, а L_i – длина i-го строительного блока. Точность модели маржинального продукта определяется как

$$C_p = \sum_{i=1}^{N_b} \sum_{j=1}^{2^{L_i}} N_{ij} \log_2(M / N_{ij}), \tag{13.50}$$

где N_{ij} – это число особей в популяции, которые включают в себя j-ю битовую последовательность в i-м строительном блоке. Если i-й строительный блок имеет длину L_i, то он включает в себя 2^{Li} возможных битовых последовательностей, и мы упорядочиваем их как (0, ..., 0, 0), (0, ..., 0, 1), ..., (1, ..., 1, 1). Задача алгоритма ECGA состоит из в том, чтобы найти модель маржинального продукта, которая минимизирует общую стоимость

$$C_c = C_m + C_p. \tag{13.51}$$

В следующем далее примере показано, как вычислить C_c для модели маржинального продукта (MPM).

■ Пример 13.9

Предположим, что у нас есть следующие пять высоко приспособленных особей ($M = 5$):

$$\begin{aligned} x_1 &= (0, 0, 0, 0), \quad x_2 = (0, 0, 1, 0), \quad x_3 = (0, 0, 1, 1), \\ x_4 &= (1, 0, 1, 0), \quad x_5 = (1, 1, 0, 1). \end{aligned} \tag{13.52}$$

Одной из возможных моделей маржинального продукта является одномерная модель

$$\hat{p_1}(x) = p(x(1))\, p(x(2))\, p(x(3))\, p(x(4)) \tag{13.53}$$

при $N_b = 4$ и $L_i = 1$ для $i \in [1, 4]$. Сложность этой модели равна

$$C_m = (\log_2 6)\sum_{i=1}^{4}(2^{L_i} - 1) = 10.4. \tag{13.54}$$

Упорядочиваем строительные блоки как $B_i = p(x(i))$ для $i \in [1, 4]$. Мы упорядочиваем битовые последовательности в двоичном порядке так, что N_{i1} является числом особей, для которых $x(i) = 0$, а N_{i2} – числом особей, для которых $x(i) = 1$. Следовательно, точность модели определяется как

$$C_p = \sum_{i=1}^{4}\sum_{j=1}^{2} N_{ij} \log_2 (M / N_{ij}) = 18.2. \tag{13.55}$$

Еще одной возможной моделью маржинального продукта для популяции уравнения (13.52) является модель

$$\hat{p_2}(x) = p(x(1), x(2))\, p(x(3))\, p(x(4)) \tag{13.56}$$

при $N_b = 3$ и $L_i = 2$ для $L_2 = L_3 = 1$. Она выглядит сложнее, чем уравнение (13.53), но мы можем предположить, что она будет точнее, потому что включает совместное распределение $x(1)$ и $x(2)$, а популяция уравнения (13.52) указывает на то, что существует значительная корреляция между этими двумя битами, то есть в четырех из пяти особей $x(1)$ и $x(2)$ имеют одинаковое значение. Сложность $\hat{p_2}(x)$ равна

$$C_m = (\log_2 6)\ (3 + 1 + 1) = 15.4, \tag{13.57}$$

что, как и ожидалось, выше, чем сложность $\hat{p_2}(x)$, как показано в уравнении (13.54). Мы упорядочиваем строительные блоки $\hat{p_2}(x)$, как показано в уравнении (13.56). Упорядочиваем битовые последовательности в двоичном порядке, так что N_{11} является числом особей, для которых $(x(1), x(2)) = (0, 0)$, N_{12} – число особей, для которых $(x(1), x(2)) = (0, 1)$, N_{13} – число особей, для которых $(x(1), x(2)) = (1, 0)$, и N_{14} – число особей, для которых $(x(1), x(2)) = (1, 1)$. Аналогичным образом, $N_2 1$ – это число особей, для которых $x(3) = 0$, а N_{22} – число особей, для которых $x(3) = 1$. Наконец, N_{31} – число особей, для которых $x(4) = 0$, и N_{32} – это число особей, для которых $x(4) = 1$. С учетом этого обозначения мы можем рассчитать точность модели как

$$C_p = \sum_{i=1}^{3}\sum_{j=1}^{2^{L_i}} N_{ij} \log_2 (M / N_{ij}) = 16.6. \tag{13.58}$$

Как и ожидалось, точность $\hat{p_2}(x)$ лучше (то есть меньше), чем точность $\hat{p_1}(x)$. Мы объединяем эти результаты, получив

$$C_m + C_p = 10.4 + 18.2 = 28.6 \quad \text{для} \quad \hat{p}_1(x)$$
$$C_m + C_p = 15.4 + 16.6 = 32.0 \quad \text{для} \quad \hat{p}_2(x) \tag{13.59}$$

Расширенный компактный генетический алгоритм (ECGA) указывает, что $\hat{p}_1(x)$ является более качественной моделью из-за ее меньшей сложности.

■

Теперь, когда мы знаем, как квантифицировать стоимость модели маржинального продукта (MPM), мы используем алгоритм наискорейшего спуска, для того чтобы найти модель маржинального продукта, которая приближенно минимизирует C_c. Получив модель маржинального продукта с N_b строительными блоками, мы можем сформировать $N_b(N_b - 1)/2$ альтернативных множеств строительных блоков путем слияния каждой возможной пары строительных блоков. Например, если даны четыре строительных блока уравнения (13.47), то мы можем сформировать следующие шесть альтернативных моделей маржинального продукта:

$$p(x(1), x(3), x(6), x(2))\ p(x(4), x(5), x(7), x(10))\ p(x(8), x(9))$$
$$p(x(1), x(3), x(6), x(4), x(5), x(7), x(10))\ p(x(2))\ p(x(8), x(9))$$
$$p(x(1), x(3), x(6), x(8), x(9))\ p(x(2))\ p(x(4), x(5), x(7), x(10))$$
$$p(x(1), x(3), x(6))\ p(x(2), x(4), x(5), x(7), x(10))\ p(x(8), x(9))$$
$$p(x(1), x(3), x(6))\ p(x(2), x(8), x(9))\ p(x(4), x(5), x(7), x(10))$$
$$p(x(1), x(3), x(6))\ p(x(2))\ p(x(4), x(5), x(7), x(10), x(8), x(9)) \tag{13.60}$$

Алгоритм ECGA работает, отбирая модель маржинального продукта из этого множества, которая минимизирует стоимость уравнения (13.51). Алгоритм ECGA обобщен на рис. 13.17. Обратите внимание, что рис. 13.17 исполняет каждое поколение. Для того чтобы реализовать алгоритм ECGA, нам нужно выбрать M, то есть некое число меньше, чем размер популяции N. Мы также должны выбрать P_c, то есть долю детей, которых мы создаем с использованием лучшей идентифицированной модели маржинального продукта. Мы создаем этих детей, случайно отбирая подмножества модели маржинального продукта из M лучших особей, указанных в начале рис. 13.17. Это эквивалентно $(N_b - 1)$-точечному скрещиванию, в результате чего каждый ребенок имеет N_b родителей, некоторые из которых могут повторяться.

Отобрать M лучших особей из текущей популяции
$\hat{p}_0(x) \leftarrow p(x(1))\ p(x(2))\ \ldots\ p(x(n))$
While (true)

$N_b \leftarrow$ число строительных блоков в $\hat{p}_0(x)$
Применить $\hat{p}_0(x)$ для формирования альтенативных MPM-моделей $\hat{p}_i(x)$
для $i \in [1, N_b(N_b - 1)/2]$
$\hat{p}(x) \leftarrow \arg\min(C_c(\hat{p}_i(x)) : i \in [1, N_b(N_b - 1)/2])$
If $\hat{p}(x) = \hat{p}_0(x)$ **then** выйти из этого цикла
Next итерация
Применить строительные блоки в $\hat{p}_0(x)$ для создания NP_c особей
для следующего поколения.
Случайно создать $N(1 - P_c)$ особей для следующего поколения.

Рис. 13.17. Построение модели маржинального продукта (MPM) с использованием наискорейшего спуска в расширенном компактном генетическом алгоритме (ECGA). Данный алгоритм исполняет каждое поколение алгоритма ECGA

13.4.2. *Другие алгоритмы оценивания многомерных вероятностных распределений*

Исследователи предложили ряд других алгоритмов оценивания многомерных вероятностных распределений, включая алгоритм факторизованного распределения (factorized distribution algorithm, FDA) [Muhlenbein и соавт., 1999], обучающийся алгоритм факторизованного распределения (learning FDA) [Muhlenbein и соавт., 1999], алгоритм оценивания байесовых сетей (estimation of Bayesian networks algorithm, EBNA) [Larrañaga и соавт., 1999a], [Larrañaga и соавт., 2000], байесов оптимизационный алгоритм (Bayesian optimization algorithm, BOA) [Pelikan и соавт., 1999], алгоритм факторизованного распределения на основе марковской сети (Markov network factorized distribution algorithm, MN-FDA) [Santana, 2003] и алгоритм оценивания вероятностного распределения на основе марковской сети (Markov network EDA, MN-EDA) [Santana, 1998]. Иерархический байесов оптимизационный алгоритм (hierarchical BOA, hBOA) пытается уменьшить вычислительную сложность байесова оптимизационного алгоритма (BOA) путем декомпозиции оптимизационной задачи на подзадачи [Pelikan, 2005].

13.5. Алгоритмы оценивания непрерывных вероятностных распределений

В предыдущих разделах обсуждались различные алгоритмы оценивания вероятностных распределений для задач с дискретной областью.

Этот раздел расширяет концепцию алгоритмов оценивания вероятностных распределений для задач с непрерывной областью. Для задач с дискретной областью эволюционно-алгоритмические особи создаются из дискретных вероятностных распределений. Та же идея используется в задачах с непрерывной областью, за исключением того, что распределения вероятностей являются непрерывными, а не дискретными.

Для того чтобы подготовить почву для алгоритмов оценивания непрерывных вероятностных распределений, сначала вспомните процедуру для дискретных алгоритмов оценивания вероятностных распределений. Предположим, что мы имеем задачу с дискретной областью, где вероятность наличия 0-го бита в i-й позиции кандидатного решения, $x(i)$, равна 0.75, а вероятность наличия 1-го бита равна 0.25. Мы можем сгенерировать $x(i)$ с помощью следующего исходного кода:

$$\begin{aligned} r &\leftarrow U[0, 1] \\ x(i) &\leftarrow \begin{cases} 0 & \text{если } r < 0.75 \\ 1 & \text{в противном случае} \end{cases} \end{aligned} \tag{13.61}$$

где r – это случайное число, равномерно распределенное на интервале [0, 1]. Если же, с другой стороны, наша задача имеет непрерывную область [0, 1], то i-я позиция кандидатного решения не является битом, а является непрерывной величиной. Мы можем создать эту величину, написав следующий исходный код:

$$\begin{aligned} r &\leftarrow U[0, 1] \\ x(i) &\leftarrow 3/2 - \sqrt{(9/4) - 2r} \end{aligned} \tag{13.62}$$

В результате получим функцию плотности вероятности (PDF) для $x(i)$, показанную на рис. 13.18. Ее можно рассматривать как непрерывный аналог уравнения (13.61), потому что вероятность $x(i)$ линейно возрастает по мере приближения $x(i)$ к 0. Обратите внимание, что функции, которые трансформируют одну функцию плотности вероятности в другую, можно найти стандартными методами из пособий по теории вероятностей (см. задачи 13.11 и 13.15) [Grinstead и Snell, 1997].

Все алгоритмы оценивания вероятностных распределений для задач с непрерывной областью работают на той же самой идее. Если дана субпопуляция относительно приспособленных особей, то мы генерируем приближенную непрерывную функцию плотности вероятности, а затем используем ее для создания следующего поколения кандидатных решений.

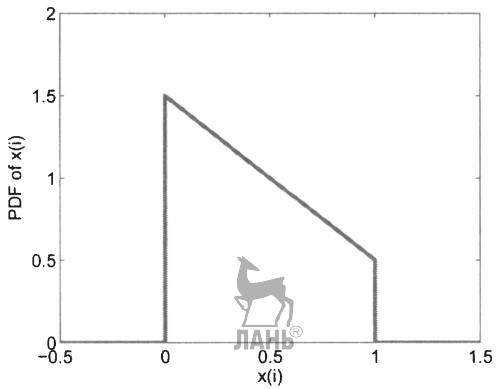


Рис. 13.18. Образец функции плотности вероятности (PDF) для непрерывной величины, описываемой уравнением (13.62)

13.5.1. *Алгоритм одномерного маржинального непрерывного распределения*

В этом разделе мы проиллюстрируем алгоритмы оценивания вероятностных распределений для задач с непрерывной областью с использованием простой модификации бинарного алгоритма одномерного маржинального распределения UMDA раздела 13.2.1. Для создания следующего поколения мы будем использовать гауссовы распределения, и поэтому этот алгоритм обозначается как UMDA® [Gallagher et al, 2007]. Алгоритм UMDA®, возможно, является простейшим алгоритмом оценивания непрерывного вероятностного распределения и схематично представлен на рис. 13.19. В алгоритме UMDA® мы вычисляем среднее значение и дисперсию каждого элемента отобранного подмножества популяции, а затем используем гауссовы случайные числа для создания следующего поколения. Мы также можем модифицировать рис. 13.19, для того чтобы создать следующее поколение распределений, отличных от гауссовых. Обратитесь к уравнению (8.18) для обоснования использования $M - 1$ вместо M в оценке σ_k.

Инициализировать популяцию кандидатных решений $\{x_i\}$, $i \in [1, N]$.
Отметить, что каждый x_i включает n непрерывных переменных $x_i(1), ..., x_i(n)$.
While not (критерий останова)
  Отобрать M особей из $\{x_i\}$ согласно приспособленности, где $M < N$.
  Индексировать M отобранных особей как $\{x_i\}$, $i \in [1, M]$.
  $\mu_k \leftarrow 1/M \sum_{j=1}^{M} x_j(k)$

$\sigma_k \leftarrow \left[1/(M-1)\sum_{j=1}^{M}(x_j(k)-\mu_k)^2\right]^{1/2}$
For $i = 1$ **to** N (размер популяции)
For $k = 1$ **to** n (число переменных в каждом кандидатном решении)
$x_i(k) \leftarrow N(\mu_k, \sigma_k^2)$
Next переменная
Next особь
Next поколение

Рис. 13.19. Алгоритм гауссова одномерного маржинального непрерывного вероятностного распределения (UMDA®) для оптимизации на n-мерной непрерывной области. $N(\mu_k, \sigma_k^2)$ – это гауссова случайная величина со средним μ_k и дисперсией σ_k^2. $x_i(k)$ – это k-й элемент i-й особи

13.5.2. *Непрерывное популяционное инкрементное самообучение*

В этом разделе мы проиллюстрируем алгоритмы оценивания непрерывных вероятностных распределений, представив модификацию бинарного алгоритма популяционного инкрементного самообучения (PBIL) раздела 13.2.3 для задач с непрерывной областью [Sebag и Ducoulombier, 1998]. Алгоритм популяционного инкрементного самообучения для задач с непрерывной областью также называется стохастическим восхождением к вершине холма с самообучением на основе векторов нормальных распределений (stochastic hill climbing with learning by vectors of normal distributions, SHCLVND) [Rudlof и Köppen, 1996], [Pelikan, 2005, раздел 2.3]. Предположим, что каждая независимая переменная $x_i(k)$ кандидатного решения x_i ограничена некоторой областью:

$$x_i(k) \in [x_{\min}(k), x_{\max}(k)] \tag{13.63}$$

для $i \in [1, N]$ и $k \in [1, n]$, где N – это размер популяции, а n – размерность задачи. Предположим, что у нас есть n-мерный вектор p – такой, что $p_k \in [x_{\min}(k), x_{\max}(k)]$ для $k \in [1, n]$. Тогда мы можем создать элемент $x_i(k)$ кандидатного решения для каждой особи x_i, генерируя гауссово случайное число, имеющее среднее p_k. По мере увеличения числа поколений мы ожидаем, что p сойдется к оптимальному решению; следовательно, мы, как правило, уменьшаем стандартное отклонение гауссова генератора случайных чисел по мере увеличения числа поколений. Эта идея проиллюстрирована на рис. 13.20.

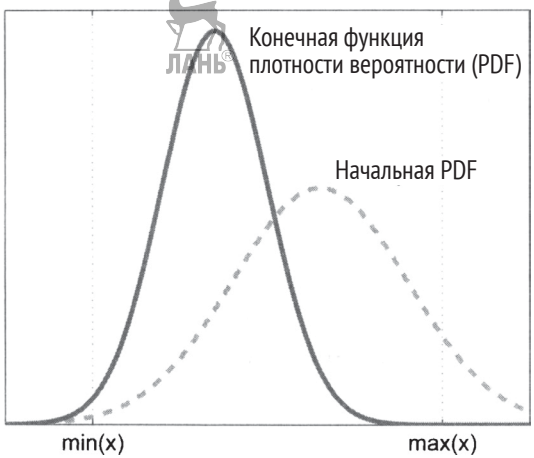


Рис. 13.20. Иллюстрация эволюции функции плотности вероятности (PDF) в непрерывном алгоритме PBIL. Функция плотности вероятности в начале работы алгоритма имеет большую дисперсию, что позволяет много разведывать поисковое пространство. Функция плотности вероятности имеет меньшую дисперсию в последующих поколениях, что позволяет алгоритму сузить оптимальное решение

Рисунок 13.21 дает простой алгоритм популяционного инкрементного самообучения (PBIL) для задач с непрерывными областями. Он очень похож на бинарный алгоритм PBIL на рис. 13.8. Его основное отличие заключается в том, что вектор P используется для генерации кандидатных решений в непрерывной области задачи, а стандартное отклонение, которое используется для генерации этих кандидатных решений, уменьшается в каждом поколении, как показано на рис. 13.20. Это дает два дополнительных регулировочных параметра, α и β на рис. 13.21. Обратите внимание,что мы должны ограничивать $x_i(k)$ областью $[x_{\min}(k), x_{\max}(k)]$ всякий раз, когда обновляем его на рис. 13.21.

N = размер популяции

$N_{\text{лучшие}}$, $N_{\text{худшие}}$ = число хороших и плохих особей, используемых для корректировки P

η = скорость самообучения $\in (0, 1)$

$[x_{\min}(k), x_{\max}(k)]$ = область k-элементного поискового пространства, $k \in [1, n]$

β = (исходное стандартное отклонение) ÷ (параметрический диапазон) $\in (0, 1)$

α = фактор сжатия стандартного отклонения $\in (0, 1)$

$\sigma_k \leftarrow \beta\{x_{\min}(k) - x_{\max}(k))$ = исходные стандартные отклонения, $k \in [1, n]$

$p_k \leftarrow U[x_{\min}(k), x_{\max}(k)]$ для $k \in [1, n]$ (равномерно распределенные случайные числа)

While not (критерий останова)

Применить p для случайного генерирования N кандидатных решений следующим образом:
For $i \in [1, N]$
  For $k \in [1, n]$
    $x_i(k) \leftarrow p_k + N(0, \sigma_k)$
  Next размерность k
Next особь i
Отсортировать особей так, чтобы $f(x_1) < f(x_2) < ... < f(x_N)$
For $i \in [1, N_{\text{лучшие}}]$
  $p \leftarrow p + \eta(x_i - p)$
Next i
For $i \in [N - N_{\text{худшие}} + 1, N]$
  $p \leftarrow p - \eta(x_i - p)$
Next i
Вероятностно мутировать p
$\sigma_k \leftarrow \alpha\sigma_k$ для $k \in [1, n]$
Next поколение

Рис. 13.21. Алгоритм PBIL для минимизации $f(x)$ на n непрерывных размерностях. $xi(k) \in [x_{\min}(k), x_{\max}(k)]$ – это k-й элемент i-го кандидатного решения. $N(0, \sigma_k)$ – это гауссова случайная величина со стандартным отклонением σ_k

■ Пример 13.10

В данном примере мы пытаемся минимизировать 20-мерную функцию Экли, определенную в приложении С.1.2. Мы используем непрерывный алгоритм PBIL на рис. 13.21 со следующими настройками:

- размер популяции $N = 50$;
- σ_k линейно уменьшается с 10%-го параметрического диапазона в исходном поколении до 2%-го параметрического диапазона в конечном поколении. Параметрический диапазон составляет [–30, 30] для каждой размерности, поэтому σ_k линейно уменьшается начиная с начального значения 6 до конечного значения 6/5. Он неточно следует профилю σ_k, показанному на рис. 13.21, но выполняет ту же самую главную цель;
- мы используем пять лучших и пять худших особей ($N_{\text{лучшие}} = N_{\text{худшие}} = 5$) для обновления вектора вероятности P в каждом поколении;
- не используем никаких мутаций.

На рис. 13.22 показано влияние скорости самообучения η на результативность алгоритма популяционного инкрементного самообучения (PBIL). Если скорость самообучения слишком мала, то P не будет достаточно агрессивным и сходимость будет медленной. Если скорость самообучения слишком высока, то P будет агрессивно скакать к хорошим решениям, что даст хорошую исходную результативность. Вместе с тем это может привести к проскакиванию мимо оптимального вектора вероятности и повернуть P к обманчивым направлениям в поисковом пространстве.

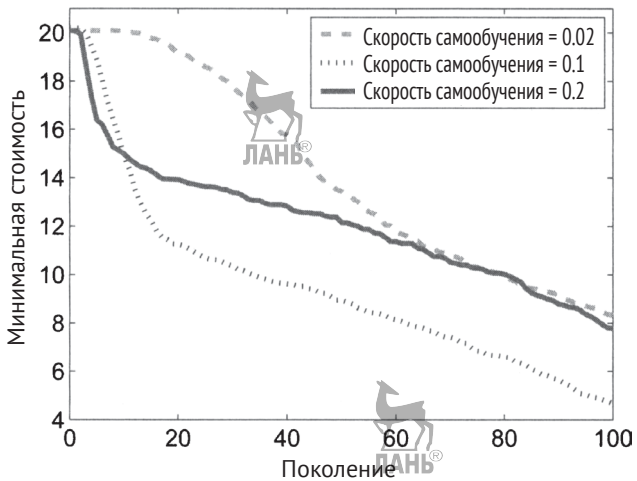


Рис. 13.22. Пример 13.10: результаты работы непрерывного алгоритма PBIL для 20-мерной функции Экли. На графике показана стоимость лучшей особи в каждом поколении, усредненно по 20 симуляциям Монте-Карло. Для того чтобы получить лучшую результативность, мы должны использовать подходящие значения скорости самообучения η

Далее мы исследуем влияние параметров $N_{\text{лучшие}}$ и $N_{\text{худшие}}$ на результативность алгоритма PBIL. Мы используем $\eta = 0.1$, так как это лучшая скорость самообучения на рис. 13.22. Рисунок 13.23 показывает результативность алгоритма PBIL для трех разных значений $N_{\text{лучшие}}$ и $N_{\text{худшие}}$. Мы видим, что если эти параметры слишком малы, то алгоритм PBIL уделяет очень много внимания нескольким особям и не получает достаточно широкой картины результативности разных особей в популяции. Однако если эти параметры слишком велики, то алгоритм PBIL корректирует свой вектор вероятности, используя очень много особей, некоторые из которых могут быть непригодны для такого использования.

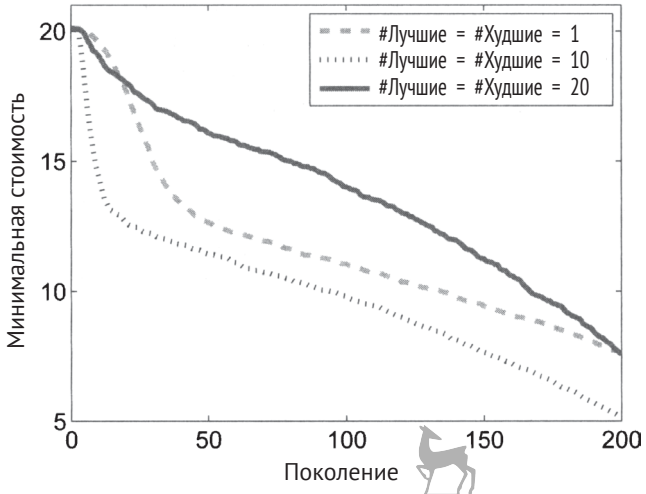


Рис. 13.23. Пример 13.10: результаты работы непрерывного алгоритма PBIL для 20-мерной функции Экли. На графике показана стоимость лучшей особи в каждом поколении, усредненно по 50 симуляциям Монте-Карло. Для того чтобы получить лучшую результативность, мы должны использовать подходящие значения $N_{\text{лучшие}}$ и $N_{\text{худшие}}$

Наконец, мы исследуем влияние параметра σ_k на результативность алгоритма PBIL. Мы используем $\eta = 0.1$ и $N_{\text{лучшие}} = N_{\text{худшие}} = 5$. Варьируем σ_k линейно $k_0(x_{\max}(k) - x_{\min}(k))$ в первое поколение до $k_f(x_{\max}(k) - x_{\min}(k))$ в последнем поколении. На рис. 13.24 показана результативность алгоритма PBIL для трех разных комбинаций k_0 и k_f. Мы видим, что если k_0 слишком мал, то начальное схождение медленное из-за относительной вялости P. Если k_f слишком велик, то алгоритм PBIL не сходится хорошо, потому что вариация кандидатных решений очень велика. Мы можем провести дальнейшие эксперименты для изучения эффекта при слишком большом k_0 или при слишком малом k_f.

■

13.6. Заключение

Алгоритмы оценивания вероятностных распределений (EDA) в этой главе оценивают вероятности с целью моделирования поискового пространства и нахождения глобального оптимума. Для оценивания выборочного среднего и ковариации популяции мы могли бы использовать максимальное правдоподобие, и именно этот подход принят в алгорит-

ме оценивания многомерного нормального распределения (estimation of multivariate normal algorithm, EMNA) [Larrañaga, 2002]. Если мы моделируем поисковое пространство с помощью гауссовых сетей, то получаем алгоритмы оценивания гауссовых сетей (estimation of Gaussian network algorithm, EGNA) [Larrañaga, 2002], [Paul and Iba, 2003].

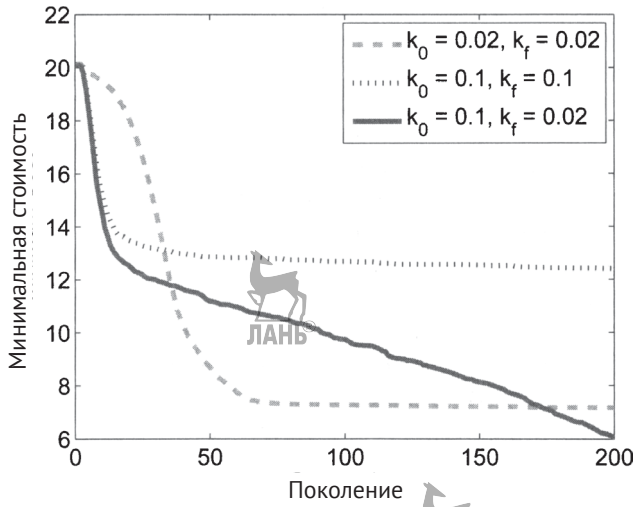


Рис. 13.24. Пример 13.10: результаты работы непрерывного алгоритма PBIL для 20-мерной функции Экли. На графике показана стоимость лучшей особи в каждом поколении, усредненно по 50 симуляциям Монте-Карло, k_0 и k_f управляют стандартным отклонением генерации кандидатных решений в первом и последнем поколениях. Для того чтобы получить лучшую результативность, мы должны использовать подходящие значения для k_0 и k_f

Алгоритмы оценивания вероятностных распределений математически моделировались с использованием цепей Маркова [González и соавт., 2001], теории динамических систем [González и соавт., 2000], [Mahnig и Mühlenbein, 2000] и других методов [González et al, 2002]. Мы рассматривали математическое моделирование эволюционных алгоритмов в главе 4, где обсуждались генетические алгоритмы, и в главе 7, где обсуждалось генетическое программирование, но в этой книге мы не рассматриваем математические модели для алгоритмов оценивания вероятностных распределений.

Алгоритмы оценивания вероятностных распределений являются относительно недавними инновациями, и поэтому существует ряд возможностей для дополнительных исследований и приложений. Текущие направления исследований алгоритмов оценивания вероятностных

распределений включают многокритериальную оптимизацию [Bureerat и Sriworamas, 2007], динамическую оптимизацию [Yang и Yao, 2008a], гибридизацию алгоритмов оценивания вероятностных распределений с другими алгоритмами [Peña и соавт., 2004] и онлайновую адаптацию параметров данного алгоритма [Santana и соавт., 2008].

В этой главе представлены алгоритмы оценивания вероятностных распределений, в которых используются статистические показатели первого и второго порядков. В первом абзаце в этом заключении упоминается несколько алгоритмов оценивания вероятностных распределений, в которых используются статистические показатели более высокого порядка. Это естественным образом повышает возможности данного алгоритма, который постепенно увеличивает порядок статистических показателей по мере приближения алгоритма к схождению. На ранних этапах алгоритма оценивания вероятностных распределений мы можем использовать статистические показатели первого порядка для получения популяции, которая достаточно близка к локальным оптимумам, а на более поздних этапах алгоритма мы можем использовать статистические показатели более высокого порядка для точной регулировки наших результатов.

Еще одним перспективным направлением будущей работы могло бы стать сочетание алгоритмов оценивания вероятностных распределений с более традиционными эволюционными алгоритмами. Это привело бы к слиянию идей рекомбинации, мутации и теории вероятностей для создания следующего поколения особей. Другие важные направления исследований алгоритмов оценивания вероятностных распределений с непрерывной областью включают изучение методов аппроксимации непрерывных функций плотности вероятности (PDF) на основе дискретного множества особей данного алгоритма. Аппроксимация функции плотности вероятности также требуется в фильтрации частиц, и поэтому существует ряд возможностей для перекрестного оплодотворения между исследованиями алгоритмов оценивания вероятностных распределений и исследованиями фильтров частиц [Simon, 2006, раздел 15.3]. Дополнительные вводные, обзорные и исследовательские материалы по алгоритмам оценивания вероятностных распределений можно найти в публикациях [Larrañaga и Lozano, 2002], [Pelikan и соавт., 2002], [Kern и соавт., 2004], [Lozano и соавт., 2006], [Shakya и Santana, 2012], и [Larrañaga и соавт., 2012]. Набор инструментов языка MATLAB для оптимизации на основе алгоритмов оценивания вероятностных распределений доступен в интернете по адресу [Santana и Echegoyen, 2012].

Задачи

Письменные упражнения

13.1. С учетом уравнения (13.1) какова вероятность того, что бит 5 равен 1, если биты 3 и 4 равны 1? Какова вероятность того, что бит 4 равен 1, если биты 3 и 5 равны 1? Какова вероятность того, что бит 3 равен 1, если биты 4 и 5 равны 1?

13.2. Каковы значения w_i в уравнении (13.4), которые сводят его к уравнению (13.3)?

13.3. Сколько поколений в компактном генетическом алгоритме (cGA) на рис. 13.4 дают такое же число оцениваний функции приспособленности, как одно поколение в алгоритме одномерного маржинального распределения (UMDA) на рис. 13.2?

13.4. Сколько поколений в обобщенном компактном генетическом алгоритме (cGA) рис. 13.6 с размером популяции N_1 эквивалентно одному поколению с размером популяции N_2 с точки зрения числа оцениваний функции приспособленности?

13.5. В примере 13.4 вычислите условную энтропию бита 2, при условии что дан бит 1.

13.6. Дано множество бинарных эволюционно-алгоритмических особей. Какова условная энтропия бита k, при условии что дан бит k?

13.7. Предположим, что энтропия пяти бит в популяции эволюционного алгоритма равна $h(1) = 0.3$, $h(2) = 0.4$, $h(3) = 0.5$, $h(2) = 0.5$ и $h(1) = 0.6$. Предположим также, что в следующей ниже таблице указана условная энтропия бита j, при условии что дан бит k.

	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
$j = 1$	0.0	0.1	0.4	0.3	0.4
$j = 2$	0.4	0.0	0.5	0.6	0.7
$j = 3$	0.9	0.8	0.0	0.7	0.6
$j = 4$	0.8	0.2	0.1	0.0	0.1
$j = 5$	0.2	0.5	0.2	0.5	0.0

a) Используйте жадный алгоритм рис. 13.10 для минимизации уравнения (13.12). Какое значение данный алгоритм дает для уравнения (13.12)?

b) Начните с $k_5 = 2$ и продолжайте жадный алгоритм, для того чтобы получить оставшиеся значения k_i. Какое значение этот подход дает для уравнения (13.12)?

13.8. Покажите, что взаимная информация между битами m и k такая же, как между битами k и m.

13.9. Проверьте расчет $I(1, 3)$ в примере 13.5.

13.10. Рассчитайте χ^2_{23} для примера 13.8.

13.11. Дана случайная величина $x \sim U[0, 1]$. Найдите функцию $y(x)$ с помощью функции плотности вероятности (PDF)

$$g(y) \leftarrow \begin{cases} 2\alpha & \text{если } 0 < y < 3/4 \\ \alpha & \text{если } 3/4 < y < 1 \end{cases}$$

Какое значение α требуется, для того чтобы сделать $g(y)$ допустимой функцией плотности вероятности?

Компьютерные упражнения

13.12. Компактный генетический алгоритм (cGA) против алгоритма одномерного маржинального распределения (UMDA): повторите пример 13.2, но используйте лимит поколений алгоритма cGA, который позволяет выполнить справедливое сравнение с результатами работы алгоритма UMDA примера 13.1 (см. задачу 13.3). Какой лимит поколений алгоритма cGA вы должны использовать? Как ваши результаты работы алгоритма cGA соотносятся с результатами работы алгоритма UMDA?

13.13. Размер популяции cGA: повторите пример 13.3 с $N = 2$ и $N = 20$, но используйте больший лимит поколений при $N = 2$, для того чтобы обеспечить справедливое сравнение между двумя разными размерами популяции (см. задачу 13.4). Какой лимит поколений следует использовать при $N = 2$? Как ваши результаты работы алгоритма cGA соотносятся при $N = 2$ и $N = 20$?

13.14. Алгоритм популяционного инкрементного самообучения (PBIL): смоделируйте алгоритм PBIL на рис. 13.8, для того чтобы

минимизировать 20-мерную функцию Экли, используя шесть бит на размерность. Выполните для 50 поколений, используйте $N_{\text{лучшие}} = N_{\text{худшие}} = 5$, $P_{\min} = 0$, $P_{\max} = 1$ и не мутируйте P. Выполните 20 симуляций Монте-Карло. Постройте график стоимости лучшей особи в каждом поколении, усредненно по 20 симуляциям Монте-Карло. Выполните это для следующих значений скорости самообучения η: 0.001, 0.01 и 0.1. Прокомментируйте свои результаты.

13.15. Трансформация функции плотности вероятности: сгенерируйте 100 000 случайных чисел $\{x_i\}$, равномерно распределенных на [0, 1]. Примените найденную вами в задаче 13.11 функцию к $\{x_i\}$, для того чтобы получить $\{y_i\}$. Постройте гистограмму $\{y_i\}$, для того чтобы убедиться, что вы получили нужную функцию плотности вероятности.

Глава 14

Биогеографическая оптимизация

«…Зоология архипелагов будет достойна изучения…»

– Чарльз Дарвин [Keynes, 2001], [MacArthur и Wilson, 1967, стр. 3]

Биогеография – это наука о видообразовании, вымирании и географическом распределении биологических видов. Как и предсказывал Чарльз Дарвин в приведенной выше цитате, биогеография действительно остается плодотворной областью исследований. Свежий поиск биологических рефератов, индекс биологических исследований, показывает, что в 2010 году было написано 37 847 статей по теме биогеографии, и этой теме посвящено несколько журналов. А популяризатор науки писатель Дэвид Квамэн (David Quammen) даже написал увлекательный рассказ о биогеографии в своей книге The Song of the Dodo («Песня Додо») [Quammen, 1997].

Точно так же, как и поведение биологических муравьев, которое породило алгоритм оптимизации на основе муравьиной кучи, наука генетика, которая породила генетические алгоритмы, и изучение животных стай, которое породило оптимизацию на основе роя частиц, наука биогеография породила биогеографическую оптимизацию. Биогеографическая оптимизация (biogeography-based optimization, BBO) является относительно недавним дополнением к стабильным эволюционным алгоритмам, но в данной книге мы посвящаем ей целую главу по следующим причинам.

- Популярность биогеографической оптимизации быстро растет. Запрос в поисковой системе Академия Google показывает следующее:

- 1 публикация в 2008 году;
- 37 публикаций в 2009 году;
- 81 публикация в 2010 году;
- 145 публикаций в 2011 году.

В 2012 году (на момент написания этой главы) мы находились в шаге от того, чтобы увидеть более 200 публикаций по теме биогеографической оптимизации. Пока не ясно, будет ли данный рост продолжаться, но эти цифры указывают на то, что биогеографическая оптимизация быстро набирает популярность.

- Несмотря на недавнее внедрение, биогеографическая оптимизация добилась больших успехов в реальных приложениях, включая биомедицинские задачи, оптимизацию уровня мощности, конструирование антенн, проектирование механических систем, робототехнику, планирование, навигацию, военные задачи и другие. За дополнительными сведениями обратитесь к веб-сайту по биогеографической оптимизации [Simon, 2012].
- В отличие от многих других недавних эволюционных алгоритмов, за короткое время с момента создания теории биогеографической оптимизации было написано относительно большое количество материалов, ей посвященных, включая работы по марковским моделям [Simon и соавт., 2011a], системно-динамическим моделям [Simon, 2011a] и статистическо-механическим моделям [Ma и соавт., 2013].
- Автор этой книги также является изобретателем биогеографической оптимизации и, следовательно, имеет естественную в ней заинтересованность.

Краткий обзор главы

В этой главе дается краткий обзор естественной биогеографии в разделе 14.1 и ее интерпретация как процесса оптимизации в разделе 14.2. Затем в разделе 14.3 мы покажем, как биогеография может быть адаптирована для получения алгоритма биогеографической оптимизации. В разделе 14.4 мы предлагаем некоторые полезные модификации и расширения алгоритма биогеографической оптимизации.

14.1. Биогеография

Биогеографию как науку можно проследить начиная с работ натуралистов XIX века, в первую очередь Альфреда Уоллеса [Wallace, 2006] и

Чарльза Дарвина [Keynes, 2001]. Уоллес обычно считается родоначальником биогеографии, хотя Дарвин гораздо более известен благодаря своей теории эволюции.

До 1960-х годов биогеография была в основном описательной и исторической, за исключением квантифицирующей докторской диссертации Юджина Манро [Munroe, 1948]. В начале 1960-х годов Роберт Макартур и Эдвард Уилсон начали работу над математическими моделями биогеографии, кульминацией которой стала их классическая книга 1967 года «The Theory of Island Biogeography» («Теория островной биогеографии») [MacArthur и Wilson, 1967]. Их больше всего интересовало распределение видов между соседними островами и математические модели вымирания и миграции видов. Со времени работы Макартура и Уилсона биогеография стала профилирующим подмножеством биологии [Hanski и Gilpin, 1997].

Математические модели биогеографии описывают видообразование (эволюцию новых видов), миграцию видов между островами и вымирание видов.

Термин остров здесь описательный, а не буквальный. Островом считается любая среда обитания, географически изолированная от других сред обитания. В классическом смысле этого слова остров изолирован от других мест обитания водой. Но острова также могут быть средой обитания, изолированной участками пустыни, рек, горных хребтов, хищников, искусственных артефактов или других препятствий. Остров может состоять из речного берега, который поддерживает травы; пруда, который поддерживает амфибий; скалистого выхода, который поддерживает улиток; или ствола мертвого дерева, который поддерживает насекомых [Hanski и Gilpin, 1997].

Утверждается, что дружественные к жизни географические районы имеют высокий индекс пригодности естественной среды обитания (habitat suitability index, HSI) [Wesche и соавт., 1987]. Признаки, которые коррелируют с индексом пригодности естественной среды обитания, включают такие факторы, как осадки, растительное многообразие, топографическое многообразие, площадь суши и температура. Эти характеризующие обитаемость переменные называются переменными индекса пригодности (suitability index variable, SIV). С точки зрения обитаемости переменные индекса пригодности являются независимыми переменными среды обитания, а индекс пригодности среды обитания – зависимой переменной.

Острова с высоким индексом пригодности среды обитания, как правило, поддерживают многие виды, а острова с низким индексом могут

поддерживать только несколько видов. Острова с высоким индексом имеют много видов, которые эмигрируют в близлежащие места обитания, просто в силу большого числа видов, которые они содержат. Эмиграция с острова с высоким индексом пригодности естественной среды обитания не происходит, потому что виды *хотят* покинуть свой дом; в конце концов, родной остров является привлекательным местом для жизни. Причиной эмиграции с этих островов является накопление случайных эффектов на большом числе видов с большой популяцией. Эмиграция происходит по мере того, как животные добираются на соседние острова на плавучих остатках растительности, вплавь, по воздуху или по ветру. Когда вид эмигрирует с острова, это не означает, что вид полностью с острова исчезает; эмигрирует только несколько представителей, поэтому эмигрирующий вид остается на своем родном острове, в то же время мигрируя на соседний остров. Тем не менее в большинстве наших обсуждений мы будем считать, что эмиграция с острова приводит к вымиранию вида на этом острове. Это допущение будет необходимым в нашем применении биогеографии для разработки алгоритма биогеографической оптимизации.

Острова с высоким индексом пригодности естественной среды обитания не только имеют высокую скорость эмиграции, но они еще имеют низкую скорость иммиграции, потому что они уже поддерживают многие виды. Виды, которые прибывают на такие острова, как правило, не выживают, несмотря на высокий индекс, потому что существует слишком большая конкуренция за ресурсы.

Острова с низким индексом пригодности естественной среды обитания имеют высокую скорость иммиграции из-за низкой популяции. Опять же, это не потому, что виды хотят иммигрировать на такие острова; в конце концов, эти острова являются нежелательными местами для жизни. Причина, почему на эти острова происходит иммиграция, заключается в том, что существует много географических возможностей для дополнительных видов. Сможет ли иммигрирующий вид выжить в своем новом доме и как долго, это уже другой вопрос. Тем не менее многообразие видов коррелирует с индексом пригодности естественной среды обитания, поэтому чем больше видов, которые прибывают на остров с низким индексом пригодности естественной среды обитания, тем больше шансов, что индекс острова увеличится [Wesche и соавт., 1987].

На рис. 14.1 показана модель численности видов на одном острове [MacArthur и Wilson, 1967]. Скорость иммиграции λ и скорость эмиграции μ являются функциями от числа видов на острове. Мы изобразили

миграционные кривые как прямые линии, но в целом они вполне могут иметь более сложный вид, о чем мы поговорим позже.

Рассмотрим иммиграционную кривую. Максимально возможная иммиграция в местообитание равняется I, которая происходит, когда на острове нет ни одного вида. По мере увеличения числа видов остров становится более заполненным, все меньше видов успешно выживают после иммиграции, а скорость иммиграции снижается. Самое большое возможное число видов, которое может поддерживать среда обитания, равняется S_{max}, при которой скорость иммиграции равна нулю.

Теперь рассмотрим эмиграционную кривую. Если на острове видов нет, то скорость эмиграции равна нулю. По мере увеличения числа видов на острове увеличивается его заселенность, все больше видов могут покидать остров, увеличивается скорость эмиграции. Максимальная скорость эмиграции равна E, когда на острове содержится наибольшее число видов, которые он может поддерживать.

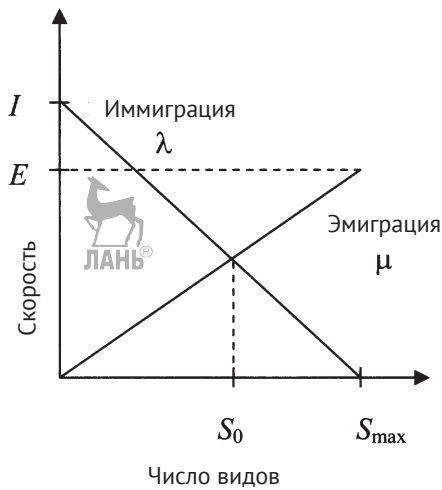


Рис. 14.1. Модель миграции видов на острове, основанная на работе [MacArthur и Wilson, 1967]. S_0 – это равновесная численность видов

Математическая модель биогеографии

В оставшейся части этого раздела представлена математическая модель численностей видов в биогеографии. Этот материал не является необходимым для понимания алгоритма биогеографической оптимизации, и поэтому читатель, если хочет, может спокойно перейти к следующему разделу.

Равновесное число видов на рис. 14.1 равняется S_0, при котором скорости иммиграции и эмиграции равны. Однако существуют случайные экскурсы от S_0 в силу временных, случайных причин. Положительные экскурсы от S_0 могут быть вызваны необычайно большим количеством плавучих остатков растительности, прибывающих с соседнего острова, либо статистически маловероятным большим числом рождений. Отрицательные экскурсы от S_0 могут быть вызваны заболеванием, временным появлением нового хищника либо природной катастрофой. Для того чтобы число видов достигло равновесия после большой пертурбации, может потребоваться много лет [Hanski и Gilpin, 1997], [Hastings и Higgins, 1994].

Теперь рассмотрим вероятность P_s, что остров содержит S видов. P_s изменяется со времени t до $(t + \Delta t)$ следующим образом:

$$P_s(t + \Delta t) = P_s(t)(1 - \lambda_s \Delta t - \mu_s \Delta t) + P_{s-1}(t)\lambda_{s-1}\Delta t + P_{s+1}(t)\mu_{s+1}\Delta t, \qquad (14.1)$$

где λ_S и μ_S – это скорости иммиграции и эмиграции, когда на острове имеется S видов. Это уравнение справедливо, если предположить, что Δt достаточно мало, чтобы можно было игнорировать вероятность более чем одной миграции между временем t и $(t + \Delta t)$. Следовательно, для того чтобы иметь S видов во время $(t + \Delta t)$, должно выполняться одно из следующих условий:

1) в момент времени t существовало S видов, и между t и $(t + \Delta t)$ не происходило иммиграции или эмиграции; либо
2) в момент времени t существовало $(S - 1)$ видов, и один вид иммигрировал; либо
3) в момент времени t существовало $(S + 1)$ видов, и один вид эмигрировал.

Взяв предел уравнения (14.1) как $\Delta t \to 0$, получим

$$\dot{P}_s \leftarrow \begin{cases} -(\lambda_s + \mu_s)P_s + \mu_{s+1}P_{s+1} & S = 0 \\ -(\lambda_s + \mu_s)P_s + \lambda_{s-1}P_{s-1} + \mu_{s+1}P_{s+1} & 1 \le S \le S_{\max} - 1 \\ -(\lambda_s + \mu_s)P_s + \lambda_{s-1}P_{s-1} & S = S_{\max} \end{cases} \qquad (14.2)$$

Определим $n = S_{\max}$ и $P = [P_0 \ldots P_n]^T$. Теперь мы можем расположить $(n + 1)$ уравнений уравнения (14.2) в одно матричное уравнение

$$\dot{P} = AP, \qquad (14.3)$$

где матрица A определяется как

$$A=\begin{bmatrix} -(\lambda_0+\mu_0) & \mu_1 & 0 & \cdots & 0 \\ \lambda_0 & -(\lambda_1+\mu_1) & \mu_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \lambda_{n-2} & -(\lambda_{n-1}+\mu_{n-1}) & \mu_n \\ 0 & \cdots & 0 & \lambda_{n-1} & -(\lambda_n+\mu_n) \end{bmatrix}. \quad (14.4)$$

Для прямолинейных скоростей миграции на рис. 14.1 мы имеем

$$\begin{aligned} \mu_k &= Ek/n \\ \lambda_k &= I(1-k/P) \end{aligned} \quad (14.5)$$

Для частного случая $E = I$ мы имеем

$$\begin{aligned} \lambda_k+\mu_k &= E=I \text{ для всех } k\in[0,n] \\ A &= E\begin{bmatrix} -1 & 1/n & 0 & \cdots & 0 \\ n/n & -1 & 2/n & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & 2/n & -1 & n/n \\ 0 & \cdots & 0 & 1/n & -1 \end{bmatrix} \\ &= EA' \end{aligned} \quad (14.6)$$

где A' определяется приведенным выше уравнением.

Теорема 14.1. (n + 1) собственных значений A' для любого натурального n равняются

$$\begin{aligned} x(A') &= \{0,-2/n,-4/n,\ldots,-n\} \\ &= -2k/n, \quad k\in[0,n] \end{aligned} \quad (14.7)$$

Более того, собственный вектор, соответствующий нулевому собственному значению, равняется

$$v(0) = [v_0(0)\ \ldots\ v_n(0)]^T$$

$$\text{где} \quad v_k(0) = \binom{n}{k} = \frac{n!}{k!(n-k)!}, \quad k\in[0,n] \quad 14.8)$$

Первая часть теоремы была высказана в публикации [Simon, 2008] и доказана в публикации [Igelnik и Simon, 2011]. Вторая часть теоремы была доказана в обеих публикациях, но двумя разными способами.

После публикации [Igelnik и Simon, 2011] мы обнаружили, что основная идея первой части теоремы 14.1 была ранее изложена без доказательств в публикациях [Clement, 1959] и [Gregory и Karney, 1969, пример 7.10].

В стационарном состоянии мы имеем $t \to \infty$, поэтому $\dot{P}(\infty) = AP(\infty) = EA'P(\infty) = 0$; то есть $P(\infty)$ является собственным вектором, который соответствует нулевому собственному значению. Напомним, что собственные векторы не определяются однозначно, а определяются только с точностью до ненулевого коэффициента шкалирования. Элементы $P(\infty)$ должны в сумме составлять 1, так как они являются вероятностями. Эти факты дают нам следующее [Simon, 2008], [Igelnik и Simon, 2011].

Теорема 14.2. Стационарное значение для вероятности количества каждого вида задается формулой

$$P(\infty) = \frac{v(0)}{\sum_{k=0}^{n} v_k(0)} = 2^{-n} v(0) \tag{14.9}$$

■ Пример 14.1

Рассмотрим остров, который может поддерживать максимум четыре вида. Максимальные скорости иммиграции и эмиграции составляют два вида в единицу времени. Следовательно, $n = 4$ и $E = I = 2$. Матрица A' уравнения (14.6) равняется

$$A' = \begin{bmatrix} -1 & 1/4 & 0 & 0 & 0 \\ 1 & -1 & 2/4 & 0 & 0 \\ 0 & 3/4 & -1 & 3/4 & 0 \\ 0 & 0 & 2/4 & -1 & 1 \\ 0 & 0 & 0 & 1/4 & -1 \end{bmatrix}. \tag{14.10}$$

Теорема 14.1 говорит нам, что собственные значения равняются $x = \{0, -1/2, -1, -3/2, -2\}$, и собственный вектор, соответствующий $x = 0$, равняется $n(0) = [1\ 4\ 6\ 4\ 1]^T$.

Теорема 14.2 говорит нам, что стационарная вероятность для количества каждого вида равняется

$$\begin{aligned} \Pr(S=0) &= \Pr(S=4) = 1/16 \\ \Pr(S=1) &= \Pr(S=3) = 4/16 \\ &\Pr(S=2) = 6/16 \end{aligned} \tag{14.11}$$

Если выполнить симуляцию миграции для 5000 временных шагов, то мы получим следующие ниже вероятности по каждому виду:

$$\begin{aligned} \Pr(S = 0) &= 0.0714 \\ \Pr(S = 1) &= 0.2605 \\ \Pr(S = 2) &= 0.3734 \\ \Pr(S = 3) &= 0.2358 \\ \Pr(S = 4) &= 0.0544 \end{aligned} \tag{14.12}$$

Они достаточно близки к аналитическим вероятностям, показанным в уравнении (14.11).

■

14.2. Биогеография как процесс оптимизации

Мы знаем, что природа содержит большое число процессов, которые оптимизируются [Alexander, 1996]. По сути дела, эта предпосылка является основополагающим принципом большинства эволюционных алгоритмов. Между тем является ли биогеография процессом оптимизации? На первый взгляд кажется, что биогеография просто поддерживает равновесие численности видов между островами и что она не обязательно оптимальна. В данном разделе рассматривается биогеография с точки зрения оптимальности.

Как обсуждалось ранее, биогеография является естественным способом распространения видов, и она часто изучается как процесс, который поддерживает равновесие в средах обитания. Равновесие, или эквилибриум, можно увидеть в точке S_0 на рис. 14.1, где пересекаются иммиграционные и эмиграционные кривые. Одна из причин, по которой биогеография рассматривалась с точки зрения равновесия, заключается в том, что эта точка зрения была первой, которая поставила биогеографию на прочную математическую основу [MacArthur и Wilson, 1963], [MacArthur и Wilson, 1967]. Однако с тех пор биогеографы все чаще ставят под сомнение или, скорее, расширяют перспективы равновесия.

В инженерном деле мы нередко рассматриваем стабильность и оптимальность как конкурирующие цели; например, простая система обычно легче стабилизируется, чем сложная система, в то время как оптимальная система обычно сложна и менее стабильна, чем более простая [Keel и Bhattacharyya, 1997]. Однако в биогеографии стабильность и оптимальность – это две стороны одной медали. Оптимальность в биогеографии предусматривает многообразные, сложные сообщества,

которые легко адаптируются к окружающей среде. Стабильность в биогеографии предусматривает сохранение существующих популяций. Полевые наблюдения показывают, что сложные сообщества более адаптируемы и стабильны, чем простые [Harding, 2006, стр. 82], и это наблюдение также было поддержано моделированием [Elton, 1958], [MacArthur, 1955].

Хотя комплементарный характер оптимальности и стабильности в биогеографии был поставлен под сомнение [May, 1973], на эти вызовы был дан адекватный ответ, и эта идея в целом принята сегодня [McCann, 2000], [Kondoh, 2006]. Таким образом, дебаты о равновесии и оптимальности в биогеографии становятся вопросом теории, поскольку равновесие и оптимальность – это просто две разные точки зрения на одно и то же явление в биогеографии.

Ярким примером оптимальности биогеографии является Кракатау, вулканический остров в Индийском океане, извергшийся в августе 1883 года [Winchester, 2008]. Извержение было слышно за тысячи миль и привело к смерти более 36 000 человек, в основном от приливных волн, остатки которых были зафиксированы вплоть до Англии. Извержение выбросило частицы пыли высотой 30 миль, которые оставались в воздухе в течение нескольких месяцев и были видны по всему миру. Рогир Вербек (Rogier Verbeek), геолог и горный инженер, был первым посетителем Кракатау через шесть недель после извержения, но поверхность острова была слишком горячей на ощупь и не показывала никаких признаков жизни. Остров был полностью стерилизован [Whittaker и Bush, 1993]. Жизнь первых животных (паук) была обнаружена на Кракатау в мае 1884 года, через девять месяцев после извержения. К 1887 году на острове были обнаружены густые поля травы. К 1906 году растительный и животный мир был в изобилии. Хотя вулканическая активность на Кракатау продолжается и сегодня, к 1983 году (через столетие после его опустения) здесь насчитывалось 88 видов деревьев и 53 вида кустарников [Whittaker и Bush, 1993], и численность видов продолжает линейно увеличиваться со временем. Жизнь иммигрирует в Кракатау, а иммиграция делает остров более пригодным для жизни, что, в свою очередь, делает остров более дружественным к дополнительной иммиграции.

Биогеография – это феномен с положительной обратной связью, по крайней мере до определенного момента. Она аналогична естественному отбору, так называемому выживанию наиболее приспособленных. По мере того как виды становятся все более приспособленными, они имеют больше шансов выжить. По мере того как они выживают

дольше, они расходятся и становятся более способными к адаптации в окружающей среде. Естественный отбор, как и биогеография, влечет за собой положительную обратную связь. Однако временной масштаб биогеографии значительно короче, чем у естественного отбора.

Еще одним хорошим примером биогеографии как процесса оптимизации является тропический лес Амазонки, который представляет типичный пример взаимно оптимизирующей системы жизнь/окружающая среда [Harding, 2006]. Тропический лес имеет огромную способность рециркулировать влагу, что ведет к уменьшенной засушливости и увеличенному испарению. А это приводит к более прохладным и влажным поверхностям, которые более пригодны для жизни. Это свидетельствует о том, что взгляд на биогеографию как «опирающуюся на *оптимизирующие* условия окружающей среды для биотической деятельности представляется более уместным, чем определение, основанное на гомеостазе» [Kleidon, 2004] (выделено курсивом автором). Такой взгляд на окружающую среду как на систему, оптимизирующую жизнь, был предложен еще в 1997 году [Volk, 1997]. Существует ряд других примеров оптимальности биогеографии, таких как температура Земли [Harding, 2006], состав атмосферы Земли [Lenton, 1998] и содержание минеральных веществ в океане [Lovelock, 1990].

Это отнюдь не означает, что биогеография оптимальна для любого определенного вида. Например, исследования атолла Бикини показывают, что высокий уровень радиоактивности в результате ядерных испытаний мало повлиял на его природную экологию, но серьезно пострадали млекопитающие [Lovelock, 1995, стр. 37]. Это и подобные исследования показывают, что Земля «позаботится о себе [и] экологические эксцессы будут сглажены, но вполне вероятно, что такое восстановление окружающей среды произойдет в мире, лишенном людей» [Margulis, 1996]. Интересно, что на фоне всех нынешних предупреждений об истощении озонового слоя мы упускаем из виду тот факт, что в течение первых двух миллиардов лет жизни на Земле вообще не было озона [Lovelock, 1995, стр. 109]. Жизнь процветает и развивается без озона, но не в человеческом направлении. Хотя глобальное потепление или ледниковый период вполне может быть губительным для человека и многих других млекопитающих, это было бы незначительным событием в общей истории биогеографии на нашей планете.

Постулат, что биогеография является процессом оптимизации, обусловливает развитие алгоритма биогеографической оптимизации как алгоритма эволюционной оптимизации, что мы и обсудим далее.

14.3. Биогеографическая оптимизация

Биогеография является природным способом распределения видов и оптимизации среды для жизни и аналогична математической оптимизации. Предположим, что у нас есть оптимизационная задача и несколько кандидатных решений, которые мы называем особями. Хорошие особи хорошо справляются с этой задачей, а слабые особи – плохо. Хорошая особь аналогична острову с высоким индексом пригодности естественной среды обитания (HSI), а слабая особь аналогична острову с низким индексом. Хорошие особи сопротивляются переменам больше, чем слабые, точно так же, как на высоко заселенных островах скорость иммиграции ниже, чем на менее заселенных островах. В то же время хорошие особи, как правило, склонны делиться своими признаками (то есть своими независимыми переменными) со слабыми особями, так же, как и высоко заселенные острова имеют высокие скорости эмиграции. Слабые особи, вероятно, примут новые признаки от хороших особей, так же как менее заселенные острова, вероятно, примут много иммигрантов с высоко заселенных островов. Добавление новых признаков слабым особям может повысить качество этих особей. Эволюционный алгоритм, опирающийся на этот подход, называется биогеографической оптимизацией (biogeography-based optimization, BBO).

Мы исходим из допущения, что каждая особь биогеографической оптимизации для простоты представлена идентичной кривой численности видов $E = I$. Рисунок 14.2 иллюстрирует скорости миграции для алгоритма биогеографической оптимизации с учетом этих допущений. S_1 на рис. 14.2 представляет слабую особь, а S_2 – сильную. Скорость иммиграции для S_1 будет относительно высокой, и это означает, что она, вероятно, получит новые признаки от других кандидатных решений. Скорость эмиграции для S_2 будет относительно высокой, и это означает, что она, вероятно, будет делиться своими признаками с другими особями. Рисунок 14.2 называется линейной моделью миграции, поскольку μ и λ – это значения линейных функций приспособленности.

Мы используем скорости миграции каждой особи с целью вероятностного обмена информацией между особями. Существует несколько разных способов реализации деталей алгоритма биогеографической оптимизации, но в этой главе мы сосредоточимся на оригинальной формулировке данного алгоритма [Simon, 2008], которая называется биогеографической оптимизацией на основе частичной иммиграции [Simon, 2011b]. Используя нашу стандартную запись, мы исходим из

того, что у нас есть популяция размером N, что x_k – это k-я особь в популяции, что размерность нашей оптимизационной задачи равна n и что $x_k(s)$ – это s-я независимая переменная в x_k, где $k \in [1, \mathrm{N}]$ и $s \in [1, n]$. В каждом поколении и для каждого признака решения в k-й особи существует вероятность λ_k (вероятность иммиграции), что данная переменная будет заменена:

$$\lambda_k = \text{Вероятность того, что } s\text{-я независимая переменная в } x_k \text{ будет заменена} \qquad (14.13)$$

для $k \in [1, \mathrm{N}]$ и $s \in [1, n]$. Если признак решения отобран для замены, то мы отбираем эмигрирующее решение с вероятностью, пропорциональной вероятностям эмиграции $\{\mu_i\}$. Для этого шага мы можем использовать любой метод отбора на основе приспособленности (см. раздел 8.7). Если мы используем рулеточный отбор, то

$$\Pr(x_j) \text{ отобрана для эмиграции} = \frac{\mu_j}{\sum_{i=1}^{N} \mu_j}. \qquad (14.14)$$

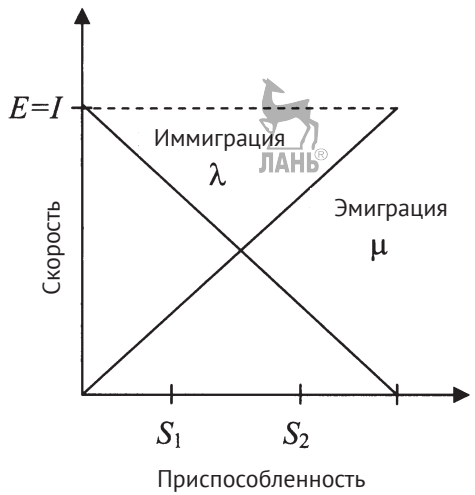


Рис. 14.2. Характеристика обмена признаками в биогеографической оптимизации. S_1 представляет слабую особь с низкой вероятностью предоставления признаков другим, но высокой вероятностью их получения от других особей. S_2 представляет хорошую особь с высокой вероятностью предоставления признаков другим, но низкой вероятностью их получения от других особей

В результате мы получаем алгоритм рис. 14.3. Миграция и мутация каждой особи в текущем поколении происходят до того, как какая-либо из особей будет заменена в популяции, что требует использования

временной популяции z на рис. 14.3. Заимствуя терминологию генетических алгоритмов [Vavak и Fogarty, 1996], мы говорим, что рис. 14.3 изображает поколенческий алгоритм биогеографической оптимизации, в отличие от стационарного алгоритма. Как и в случае с другими эволюционными алгоритмами, в алгоритме биогеографической оптимизации мы обычно реализуем элитарность (см. раздел 8.4), хотя это не показано на рис. 14.3.

Инициализировать популяцию кандидатных решений $\{x_k\}$ для $k \in [1, N]$.
While not (критерий останова)
  For each x_k, назначить вероятность эмиграции $\mu_k \propto$ приспособленность x_k,
    при $\mu_k \in [0, 1]$
  For each особь x_k, назначить вероятность иммиграции $\lambda_k = 1 - \mu_k$
  $\{z_k\} \leftarrow \{x_k\}$
  For each особь z_k
    For each признак s решения
      Применить λ_k для вероятностного принятия решения об иммиграции в z_k
        (см. уравнение (14.13)).
      If иммигрировать **then**
        Применить $\{\mu_i\}_{i=1}^N$ для вероятностного отбора эмигрирующей особи x_j
          (см. уравнение (14.14)).
        $z_k(s) \leftarrow x_j(s)$
      End if
    Next признак решения
    Вероятностно мутировать $\{z_k\}$.
  Next особь
  $\{x_k\} \leftarrow \{z_k\}$
Next поколение

Рис. 14.3. Схематичное представление алгоритма биогеографической оптимизации (BBO) с популяцией размером N. Данный алгоритм также называется алгоритмом BBO на основе частичной иммиграции. $\{x_k\}$ – это вся популяция особей в целом, x_k – это k-я особь и $x_k(s)$ – s-й признак x_k. Аналогичным образом, $\{z_k\}$ – это временная популяция особей, z_k – k-я временная особь и $z_k(s)$ – s-й признак z_k

Рисунок 14.4 иллюстрирует миграцию в алгоритме биогеографической оптимизации. Рисунок показывает, что особь z_k «иммигрирует» признаки. Мы используем уравнение (14.13), для того чтобы решить, следует или нет заменять каждый признак в z_k. В примере на рис. 14.4 мы видим следующие миграционные решения.

1. Для первого признака иммиграция не отобрана; именно поэтому первый признак в z_k остается неизменным.
2. Для второго признака иммиграция отобрана, и уравнение (14.14) выбирает x_1 в качестве эмигрирующей особи; именно поэтому второй признак в z_k заменяется вторым признаком из x_1.
3. Для третьего признака иммиграция отобрана, и уравнение (14.14) выбирает x_3 в качестве эмигрирующей особи; именно поэтому третий признак в z_k заменяется третьим признаком из x_3.
4. Для четвертого признака иммиграция отобрана, и уравнение (14.14) выбирает x_2 в качестве эмигрирующей особи; именно поэтому четвертый признак в z_k заменяется четвертым признаком из x_2.
5. Наконец, для пятого признака иммиграция отобрана, и уравнение (14.14) выбирает x_N в качестве эмигрирующей особи; именно поэтому пятый признак в z_k заменяется пятым признаком из x_N.

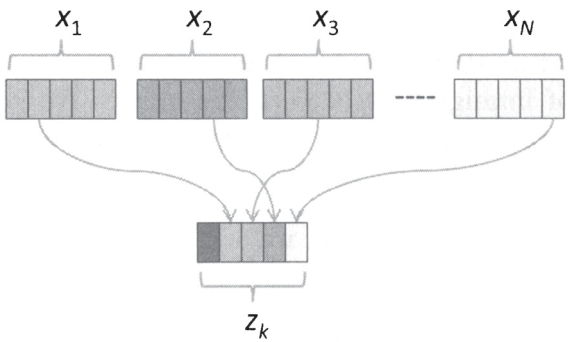


Рис. 14.4. Иллюстрация миграции в алгоритме биогеографической оптимизации (BBO) для пятимерной задачи. Признак 1 для иммиграции не отобран, но признаки 2–5 для иммиграции отобраны. Уравнение (14.14) используется для отбора эмигрирующих особей

■ Пример 14.2

Данный простой эксперимент с использованием алгоритма BBO мотивирован публикацией «GA simulation by hand» («Симуляция генетического алгоритма вручную») Дэвида Голдберга [Goldberg, 1989a]. Предположим, что мы хотим максимизировать x^2, где x кодируется как пятибитное целое число. Мы должны решить, сколько особей хотим иметь в нашей популяции и какую скорость мутации хотим ис-

пользовать. Мы начинаем со случайно сгенерированной популяции из четырех особей и скорости мутации 1 % на бит. Для каждой особи мы вычисляем значение приспособленности x^2, а затем линейно назначаем скорости миграции, как показано на рис. 14.2. Скорость миграции должна быть между 0 и 1, но мы часто принимаем наименьшее значение немного большим 0, а наибольшее значение чуть меньшим 1. Это допускает некоторую случайность (недетерминизм) даже для лучших и худших особей в популяции. В данном примере в качестве минимальных значений для λ и μ мы произвольно решаем использовать $1/N$ и в качестве максимальных значений – $(N-1)/N$, где $N = 4$ – это размер популяции. Предположим, что наша случайная исходная популяция создана, как показано в табл. 14.1.

Таблица 14.1. Пример 14.2: исходная популяция для простой задачи с использованием алгоритма BBO

Номер битовой строки	x (двоичная)	x (десятичная)	$f(x) = x^2$	μ	λ
1	01101	13	169	2/5	3/5
2	11000	24	576	4/5	1/5
3	01000	8	64	1/5	4/5
4	10011	19	361	3/5	2/5

Первое, что мы сделаем, – это скопируем популяцию x во временную популяцию z. Затем мы рассмотрим возможность иммиграции в каждый бит первой особи во временной популяции, z_1, которая равна x_1 (01101). Мы упорядочиваем битовые числа слева направо, начиная с индекса 1. Поэтому мы видим, что

$$z_1(1) = 0, \quad z_1(2) = 1, \quad z_1(3) = 1, \quad z_1(4) = 0, \quad z_1(5) = 1. \tag{14.15}$$

Поскольку z_1 является третьей наиболее приспособленной особью, скорость иммиграции $\lambda_1 = 3/5$, поэтому есть 60%-ный шанс иммигрировать в каждый бит в z_1. Мы генерируем случайное число $r \sim U[0, 1]$ для каждого бита в z_1, для того чтобы определить, следует или нет иммигрировать в этот бит.

1. Предположим, $r = 0.7$. Поскольку $r > \lambda_1$, мы не будем иммигрировать в $z_1(1)$, поэтому $z_1(1)$ остается равным 0.
2. Предположим, что следующее случайное число, которое мы генерируем, равно $r = 0.3$. Поскольку $r < \lambda_1$, мы иммигрируем в $z_1(2)$.

Для того чтобы отобрать имигрирующий бит, мы используем рулеточный отбор. $x_3(2)$ имеет наибольшую вероятность иммигрировать в $z_1(2)$, $x_1(2)$ имеет вторую наибольшую вероятность, $x_4(2)$ имеет третью наибольшую вероятность, и $x_2(2)$ имеет наименьшую вероятность. Мы можем исключить $x_1(2)$ из рассмотрения, поскольку z_1 является копией x_1, но это деталь реализации, которая зависит от предпочтения инженера. Предположим, что этот процесс рулеточного отбора приводит к отбору $x_3(2)$ для иммиграции. Тогда $z_1(2) \leftarrow x_3(2) = 1$. Несмотря на то что мы иммигрировали в $z_1(2)$, он не изменил свое исходное значение.

3. Мы продолжаем этот процесс для $z_1(3)$, $z_1(4)$ и $z_1(5)$. Предположим, что генерируемые случайные числа приводят к следующему:
   - $z_1(3) = 1$ (без иммиграции),
   - $z_1(4) \leftarrow x_4(4) = 1$ (иммиграция) и
   - $z_1(5) = 1$ (без иммиграции).

   Теперь мы завершили процесс миграции для z_1 и получили $z_1 = 01111$.
4. Повторяем шаги 1–3 для z_2, z_3 и z_4.
5. Далее мы рассмотрим возможность мутации для каждого бита в каждой временной особи z_1, z_2, z_3 и z_4. Мутация может быть реализована, как и в любом другом эволюционном алгоритме (см. раздел 8.9).
6. Теперь, когда у нас есть модифицированная популяция $\{z_k\}$ особей, мы копируем z_k в x_k для $k \in [1, 4]$, и первое поколение алгоритма BBO завершено.

Вышеуказанный процесс продолжается до тех пор, пока не будет выполнен некий критерий сходимости. Например, мы можем продолжать в течение определенного числа поколений, либо до тех пор, пока не достигнем удовлетворительного значения приспособленности, либо пока значение приспособленности не перестанет изменяться (см. раздел 8.2).

■

14.4. Расширения алгоритма биогеографической оптимизации

В этом разделе рассматривается несколько расширений, которые могут быть внесены в алгоритм биогеографической оптимизации (BBO) для

улучшения его результативности. Мы обсудим формы миграционной кривой, смешанную миграцию и альтернативные подходы к реализации алгоритма биогеографической оптимизации. Мы завершим это обсуждение разделом 14.4.4, в котором рассматривается вопрос, следует или нет считать алгоритм BBO разновидностью генетического алгоритма, а не отдельным эволюционным алгоритмом.

14.4.1. *Миграционные кривые*

До этого момента мы исходили из того, что миграционные кривые алгоритма биогеографической оптимизации являются линейными, как показано на рис. 14.2. Это допущение удобно, и оно соответствует линейному ранговому отбору (см. раздел 8.7.4); но в биогеографии миграционные кривые нелинейны. Точная форма миграционных кривых биогеографии трудно поддается квантификации и изменяется от одного острова к другому. Вместе с тем многие кривые в природе имеют S-образную форму. В оригинальной статье по алгоритму биогеографической оптимизации [Simon, 2008] предполагалось, что нелинейные миграционные кривые вполне могут давать более высокую результативность, чем линейные кривые. Это утверждение привело к исследованию нескольких разных миграционных кривых в публикациях [Ma и соавт., 2009], [Ma, 2010]. Здесь мы обсудим наиболее перспективную кривую: S-образные миграционные кривые.

На рис. 14.2 моделируются ненормализованные скорости миграции как

$$\begin{aligned} \mu_k &= r_k \\ \lambda_k &= 1 - r_k \end{aligned} \tag{14.16}$$

где r_k – это ранг приспособленности k-й особи популяции, $r_k = 1$ для наименее приспособленной особи, и $r_k = N$ (где N – это размер популяции) для наиболее приспособленной особи. Синусоидальное моделирование скорости миграции назначает скорости миграции как

$$\begin{aligned} \mu_k &= 1/2(1 - \cos(\pi r_k/N)) \\ \lambda_k &= 1 - \mu_k \end{aligned} \tag{14.17}$$

Эти уравнения приводят к S-образным кривым, показанным на рис. 14.5.

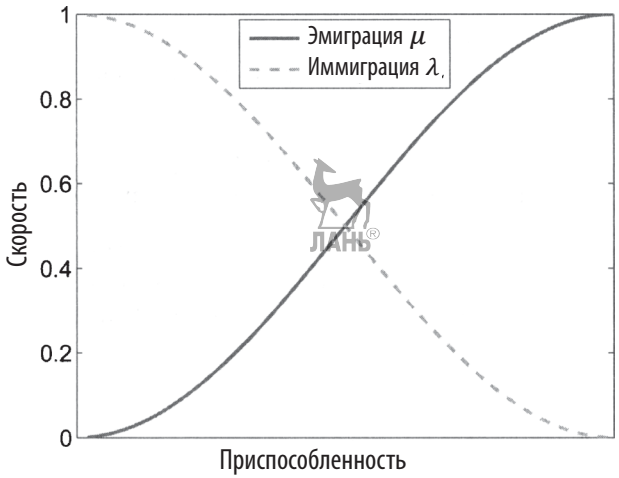


Рис. 14.5. Синусоидальная модель миграции в алгоритме BBO. Сравните с рис. 14.2

■ Пример 14.3

Если естественная биогеография действительно является процессом оптимизации, то логично предположить, что более тесное моделирование алгоритма BBO по образцу естественной биогеографии смогло бы привести к более высокой результативности оптимизации. Имея эту идею в виду, мы просимулируем линейный алгоритм BBO и синусоидальный алгоритм BBO на множестве 20-мерных эталонных задач, получая результаты, показанные в табл. 14.2. Для каждого прогона алгоритма BBO мы используем популяцию размером 50, лимит поколений 50 и скорость мутации 1 % на каждый признак решения. Мы реализуем мутацию, создавая признак решения, равномерно распределенный между минимальным и максимальным значениями области с вероятностью 1 % на особь на поколение. Мы также используем параметр элитарности 2, что означает, что мы оставляем двух лучших особей из поколения в поколение.

Из табл. 14.2 видно, что синусоидальная миграция явно превосходит линейную для стандартных эталонов, приведенных в таблице. Средняя эффективность синусоидальной миграции на 43 % выше линейной миграции. Это показывает, что модели миграции, которые ближе к природе, превосходят более простые модели миграции, и поддерживает гипотезу о том, что естественная биогеография сама по себе является процессом оптимизации.

Таблица 14.2. Результаты примера 14.3: относительная результативность алгоритма BBO с линейной и синусоидальной моделями миграции. В таблице показан нормализованный минимум, найденный двумя версиями алгоритма BBO, усредненно по 50 симуляциям Монте-Карло. См. определения эталонных функций в приложении *C*

Эталон	Линейная миграция	Синусоидальная миграция
Ackley	1.0373	1
Fletcher	1.2015	1
Griewank	1.2367	1
Penalty #1	1.4249	1
Penalty #2	4.3265	1
Quartic	1.6876	1
Rastrigin	1.0665	1
Rosenbrock	1.0759	1
Schwefel 1.2	1.0980	1
Schwefel 2.21	1.0468	1
Schwefel 2.22	1.0721	1
Schwefel 2.26	1.2471	1
Sphere	1.2582	1
Step	1.2683	1
Среднее	**1.4319**	**1**

■

14.4.2. *Смешанная миграция*

Смешанное скрещивание, как было показано, повышает результативность генетических и других эволюционных алгоритмов [McTavish и Restrepo, 2008], [Mezura-Montes и Palomeque-Oritiz, 2009], [Muhlenbein и Schlierkamp-Voosen, 1993] (см. раздел 8.8.9). В смешанном скрещивании, используемом в генетическом алгоритме, вместо копирования гена одиночного родителя в ген ребенка последний получает в виде выпуклой комбинации 2 гена родителя. Это обусловливает использование оператора смешанной миграции для алгоритма BBO [Ma и Simon, 2010], [Ma и Simon, 2011b]. В стандартном алгоритме BBO на рис. 14.3 признак s особи z_k полностью заменяется признаком из особи x_j:

$$z_k(s) \leftarrow x_j(s). \tag{14.18}$$

В смешанной миграции в алгоритме BBO признак особи z_k не просто заменяется признаком из особи x_j; вместо этого признак особи z_k принимается равным выпуклой комбинации $z_k(s)$ и $x_j(s)$:

$$z_k(s) \leftarrow \alpha z_k(s) + (1 - \alpha)\, x_j(s), \tag{14.19}$$

где $\alpha \in (0, 1)$. Если $\alpha = 0$, то смешанный алгоритм BBO сводится к стандартному алгоритму BBO; поэтому смешанный алгоритм BBO является обобщением стандартного алгоритма BBO. Параметр смешивания α может быть случайным, детерминированным или пропорциональным относительной приспособленности z_k и x_j.

Смешанная миграция подходит для задач с непрерывными признаками решения. Вполне возможно, что данный алгоритм может быть адаптирован для задач с дискретными признаками решения, но мы эту идею здесь не исследуем. Существует несколько обоснований для применения смешанной миграции по сравнению со стандартной миграцией. Во-первых, хорошие особи будут менее подвержены деградации из-за миграции, поскольку в процессе миграции они сохранят определенную долю своих первоначальных признаков. Во-вторых, во время миграции слабые особи будут по-прежнему принимать, по крайней мере, часть признаков решения от хороших особей.

■ Пример 14.4

Для того чтобы изучить влияние смешанной миграции на результативность работы алгоритма BBO, мы симулируем стандартный алгоритм BBO и смешанный алгорим BBO при $\alpha = 0.5$ на множестве 20-мерных эталонных задач. Мы используем те же параметры алгоритма BBO, что и в примере 14.3, и получаем результаты, показанные в табл. 14.3.

Таблица 14.3. Результаты примера 14.4: относительная результативность стандартного алгоритма BBO и смешанного алгоритма BBO при $\alpha = 0.5$. В таблице показан нормализованный оптимум, найденный двумя версиями алгоритма BBO, усредненно по 50 симуляциям Монте-Карло. См. определения эталонных функций в приложении *C*

Эталон	Стандартный алгоритм BBO ($\alpha = 0$)	Смешанный алгоритм BBO ($\alpha = 0.5$)
Ackley	1.6559	1.0
Fletcher	1.0	2.388
Griewank	3.4536	1.0

Эталон	Стандартный алгоритм BBO (α = 0)	Смешанный алгоритм BBO (α = 0.5)
Penalty #1	701.47	1.0
Penalty #2	8817.7	1.0
Quartic	49.663	1.0
Rastrigin	1.0	1.6892
Rosenbrock	3.9009	1.0
Schwefel 1.2	12.63	1.0
Schwefel 2.21	4.0846	1.0
Schwefel 2.21	1.3280	1.0
Schwefel 2.26	1.0	4.8213
Sphere	5.4359	1.0
Step	4.5007	1.0
Среднее	**686.34**	**1.4213**

В табл. 14.3 показано, что смешанный алгоритм BBO работает лучше, чем стандартный алгоритм BBO на 11 из 14 эталонов. Величина улучшения весьма впечатляет. Стандартный алгоритм BBO работает лучше на трех эталонах со средним коэффициентом улучшения около 3. Но смешанный алгоритм BBO работает лучше на 11 эталонах, с фактором улучшения до 8818 (функция Penalty #2).

■

14.4.3. *Другие подходы к биогеографической оптимизации*

Алгоритм, представленный на рис. 14.3, называется алгоритмом биогеографической оптимизации (BBO) на основе частичной иммиграции [Simon, 2011b]. Слово «частичный» в названии означает, что для иммиграции одновременно рассматривается только один признак решения. То есть для особи z_k скорость иммиграции λ_k тестируется против случайного числа один раз для каждого признака решения, для того чтобы решить, следует или нет заменить этот признак решения. Термин «иммиграция» в названии означает, что λ_k сначала используется для принятия решения о том, иммигрировать или нет в z_k, и только после принятия решения об иммиграции переменные $\{\mu_i\}$ используются для выбора эмигрирующего решения, используя некую процедуру, такую как рулеточный отбор.

Однако есть и другие способы реализации идеи алгоритма BBO. Вместо того чтобы тестировать λ_k против случайного числа один раз для каждого признака решения, мы можем протестировать λ_k против случайного числа только один раз для каждой особи, а затем, если будет принято решение об иммиграции, мы заменим все признаки решения в z_k. Этот подход можно назвать алгоритмом BBO на основе полной иммиграции.

Кроме того, мы можем сначала использовать μ_k, чтобы решить, следует или нет эмигрировать признак от данной особи. Затем, только если будет принято решение об эмиграции, мы будем использовать $\{\lambda_i\}$ переменных в рулеточном процессе, для того чтобы отобрать, куда иммигрировать выбранный признак решения. Эта идея порождает алгоритм BBO на основе эмиграции.

Комбинация упомянутых выше идей приводит к четырем разным реализациям алгоритма BBO. Первая, алгоритм BBO на основе частичной иммиграции, является реализацией по умолчанию и представлена на рис. 14.3. Остальные три представлены на рис. 14.6–14.8. Кроме того, каждый из этих подходов может быть объединен с синусоидальными миграционными кривыми, как описано в разделе 14.4.1, и/или смешанной миграцией, как описано в разделе 14.4.2. Как и в любом другом эволюционном алгоритме, мы также должны реализовать мутацию и элитарность, хотя эти процедуры не показаны на рис. 14.6–14.8. Теоретические и прикладные исследования этих вариаций алгоритма BBO представлены в публикации [Ma и Simon, 2013].

Инициализировать популяцию кандидатных решений $\{x_k\}$ для $k \in [1, N]$.
While not (критерий останова)
  For each x_k, назначить вероятность эмиграции $\mu_k \propto$ приспособленность x_k, при $\mu_k \in [0, 1]$
  For each особь x_k, назначить вероятность иммиграции $\lambda_k = 1 - \mu_k$
  $\{z_k\} \leftarrow \{x_k\}$
  For each особь x_k
    For each признак s решения
    Применить μ_k для вероятностного принятия решения об эмиграции $x_k(s)$.
    If эмигрировать **then**
      Применить $\{\lambda_i\}$ для вероятностного отбора эмигрирующего решения z_j
      $z_j(s) \leftarrow x_k(s)$
    End if
  Next признак решения
  Next особь
  Вероятностно мутировать $\{z_k\}$

$\{x_k\} \leftarrow \{z_k\}$
Next поколение

Рис. 14.6. Приведенный выше алгоритм описывает алгоритм BBO на основе частичной эмиграции с популяцией размером N. $\{x_k\}$ – это вся популяция особей в целом, x_k – это k-я особь и $x_k(s)$ – s-й признак x_k. Аналогичным образом, $\{z_k\}$ – это временная популяция особей, z_k – k-я временная особь и $z_k(s)$ – s-й признак z_k

Инициализировать популяцию кандидатных решений $\{x_k\}$ для $k \in [1, N]$.
While not (критерий останова)
  For each x_k, назначить вероятность эмиграции $\mu_k \propto$ приспособленность x_k, при $\mu_k \in [0, 1]$
  For each особь x_k, назначить вероятность иммиграции $\lambda_k = 1 - \mu_k$
  $\{z_k\} \leftarrow \{x_k\}$
  For each особь z_k
    Применить λ_k для вероятностного принятия решения об иммиграции в z_k.
    If иммигрировать **then**
      For each признак s решения
        Применить $\{\mu_i\}$ для вероятностного отбора эмигрирующего решения x_j
        $z_k(s) \leftarrow x_j(s)$
      Next признак решения
    End if
  Next особь
  $\{x_k\} \leftarrow \{z_k\}$
Next поколение

Рис. 14.7. Приведенный выше алгоритм описывает алгоритм BBO на основе полной иммиграции с популяцией размером N. $\{x_k\}$ – это вся популяция особей в целом, x_k – k-я особь, а $x_k(s)$ – s-й признак x_k. Аналогичным образом, $\{z_k\}$ – это временная популяция особей, z_k – k-я временная особь и $z_k(s)$ – s-й признак z_k

Инициализировать популяцию кандидатных решений $\{x_k\}$ для $k \in [1, N]$.
While not (критерий останова)
  For each x_k, назначить вероятность эмиграции $\mu_k \propto$ приспособленность x_k, при $\mu_k \in [0, 1]$
  For each особь x_k, назначить вероятность иммиграции $\lambda_k = 1 - \mu_k$
  $\{z_k\} \leftarrow \{x_k\}$
  For each особь x_k
    Применить μ_k для вероятностного принятия решения об эмиграции x_k.
    If эмигрировать **then**
      For each признак s решения

```
            Применить {λ_i} для вероятностного отбора эмигрирующего решения z_j
            z_j(s) ← x_k(s)
        Next признак решения
    End if
Next особь
{x_k} ← {z_k}
Next поколение
```

Рис. 14.8. Приведенный выше алгоритм схематично представляет алгоритм BBO на основе полной эмиграции с популяцией размером N. $\{x_k\}$ – это вся популяция в целом, x_k – это k-я особь и $x_k(s)$ – s-й признак x_k. Аналогичным образом, $\{z_k\}$ – это временная популяция особей, z_k – k-я временная особь и $z_k(s)$ – s-й признак z_k

14.4.4. *Алгоритм биогеографической оптимизации и генетические алгоритмы*

В этом разделе рассматривается взаимосвязь между генетическими алгоритмами и алгоритмом BBO. В генетических алгоритмах с равномерным скрещиванием мы случайно выбираем каждый детский ген у одного из двух его родителей (см. раздел 8.8.4). В генофондовой рекомбинации, так называемой многородительской рекомбинации или сканирующем скрещивании, мы случайно отбираем каждый детский ген у одного из родителей, где число родителей больше двух (см. раздел 8.8.5). При реализации генофондовой рекомбинации в генетических алгоритмах нам нужно принять несколько решений. Например, сколько особей должно быть в фонде потенциальных родителей? Как выбирать особей для фонда? Каких родителей отбирать из фонда, после того как фонд определен? Один из способов реализации генофондовой рекомбинации можно было бы назвать глобальной равномерной рекомбинацией, при которой мы случайно выбираем каждый детский ген у одного из родителей, где родительская популяция эквивалентна всей популяции генетического алгоритма в целом и случайный отбор основан на значениях приспособленности (например, рулеточном отборе).

Если мы используем глобальную равномерную рекомбинацию и если мы также используем отбор на основе приспособленности для каждого признака решения в каждом ребенке, то мы получаем алгоритм, показанный на рис. 14.9, который называем генетическим алгоритмом с глобальной равномерной рекомбинацией (genetic algorithm with global uniform recombination, GA/GUR). Сравнивая рис. 14.3 и 14.9, мы видим, что алгоритм BBO является обобщением особого вида алгоритма GA/GUR.

Причина в том, что если, вместо того чтобы назначить $\lambda_k = 1 - \mu_k$ в алгоритме BBO рис. 14.3, мы назначаем $\lambda_k = 1$ для всех k, то алгоритм BBO рис. 14.3 будет эквивалентным алгоритму GA/GUR на рис. 14.9.

Инициализировать популяцию кандидатных решений $\{x_k\}$ для $k \in [1, N]$.
While not (критерий останова)
  For $k = 1$ **to** N
    Ребенок$_k \leftarrow [\,0\ 0\ \ldots\ 0\,] \in R^n$
    For each признак $s = 1$ **to** n решения
      Применить значения приспособленности для вероятностного отбора особи x_j.
      Ребенок$_k \leftarrow x_j(s)$
    Next признак решения
    Вероятностно мутировать Ребенка$_k$
  Next ребенок
  $\{x_k\} \leftarrow$ {Ребенок$_k$}
Next поколение

Рис. 14.9. Схематичное представление генетического алгоритма с глобальной равномерной рекомбинацией (GA/GUR) для задачи n-мерной оптимизации. N – это размер популяции, $\{x_k\}$ – это вся популяция особей в целом, x_k – k-я особь и $x_k(s)$ – s-й признак x^*

Данное обсуждение аналогично обсуждению в разделе 12.4, где мы показали, что алгоритм дифференциальной эволюции (DE) является особым видом непрерывного генетического алгоритма. В том разделе мы увидели, что, несмотря на то что алгоритм дифференциальной эволюции и генетические алгоритмы имеют много общего, алгоритм дифференциальной эволюции достаточно особенный, чтобы считать его отдельным эволюционным алгоритмом, а не особым типом генетического алгоритма. В этом разделе мы делаем аналогичный вывод. Несмотря на то что алгоритм биогеографической оптимизации (BBO) и генетические алгоритмы имеют много общего, алгоритм BBO достаточно особенный, чтобы считать его отдельным эволюционным алгоритмом, а не особым типом генетического алгоритма. Существует еще одна более важная причина рассматривать алгоритм BBO в качестве отдельного эволюционного алгоритма, и эта причина в том, что биогеографические корни алгоритма BBO открывают множество возможностей для расширений и модификаций, которые в противном случае исследователю были бы недоступны. Мы обсудили одно из таких расширений в разделе 14.4.1, и некоторые другие обсудим в заключении этой главы.

14.5. Заключение

Мы увидели, как биогеография, наука о географическом распределении биологических видов, может использоваться для получения алгоритма биогеографической оптимизации (BBO). Алгоритм биогеографической оптимизации моделировался с использованием теории Маркова [Simon и соавт., 2011a], динамических систем [Simon, 2011a] и статистической механики [Ma и соавт., 2013]. Некоторые из этих моделей аналогичны тем, которые мы получили для генетических алгоритмов (см. главу 4). Марковские модели генетических алгоритмов и алгоритма BBO сравниваются в публикации [Simon и соавт., 2011b]. Марковская модель алгоритма BBO была распространена на алгоритм BBO с элитарностью в публикации [Simon и соавт., 2009]. Как и многие другие эволюционные алгоритмы, алгоритм BBO применяется ко многим реальным задачам. Веб-сайт, посвященный алгортму биогеографической оптимизации, доступен по адресу [Simon, 2012].

Одним из недостатков представленного здесь алгоритма биогеографической оптимизации является то, что он между решениями мигрирует только одну независимую переменную за раз. Это отлично работает для разделимых задач, то есть задач, чья функция приспособленности $f(x)$ может быть записана как

$$f(x) = \sum_{i=1}^{n} f_i(x(s_i)), \tag{14.20}$$

где $x(s_i)$ – это i-я независимая переменная x и n – размерность задачи. Однако большинство оптимизационных задач не разделимо. Это означает, что если кандидатное решение содержит группу независимых переменных, которая делает его высоко приспособленным, то простой способ переноса этой группы в другое кандидатное решение отсутствует. Одним из способов устранения данного недостатка могло бы быть изменение алгоритма BBO таким образом, чтобы за раз мигрировали случайные группы независимых переменных, а не только одна независимая переменная. Нечто подобное, хотя и не идентичное, этой идее было предложено в публикации [Omran и соавт., 2013].

Алгоритм биогеографической оптимизации фактически представляет собой семейство алгоритмов, и поэтому его можно назвать метаэвристикой. Он включает варианты реализации, которые указаны в табл. 14.4. Систематическое изучение комбинаций этих вариантов, указанных в табл. 14.4, как теоретических, так и прикладных, остается задачей будущих исследований.

Таблица 14.4. Варианты реализации алгоритма BBO. Алгоритм BBO может быть реализован как комбинация любого столбца 1, любого столбца 2 и любого столбца 3 на выбор

Подходы к миграции	Миграционные кривые	Спешивание миграций
На основе частичной иммиграции	Линейные	None ($\alpha = 0$)
На основе полной иммиграции	Синусоидальные	$\alpha = 0.5$
На основе частичной эмиграции	Другие	α = некая другая константа
На основе полной эмиграции		$\alpha \propto$ приспособленность

Для более тесного согласования алгоритма BBO с биогеографией существует ряд других интересных возможностей, в том числе следующие.

- **Сходство местообитаний**: в островной биогеографии скорость иммиграции коррелирует с изолированностью островов [Adler и Nuernberger, 1994]. Изолированные острова относительно хорошо защищены от иммиграции. Эта интуитивно понятная идея называется эффектом расстояния [Wu и Vankat, 1995]. Вполне естественно, что скорости эмиграции коррелируют с изолированностью островов. В островной биогеографии экологическая уникальность острова связана с его изолированностью, поскольку условия окружающей среды предсказуемо меняются в зависимости от географического расстояния [Lomolino, 2000a]. В алгоритме BBO изоляция кандидатного решения будет связана с уникальностью кандидатного решения. Схожие острова можно рассматривать как сгруппированные в пространстве решений, а несхожие решения – как изолированные в пространстве решений. В языке биогеографии схожие решения будут принадлежать одному и тому же архипелагу (островной группе). Это будет способствовать увеличению иммиграции и эмиграции между схожими решениями и снижению этих скоростей между несхожими решениями. Это может быть реализовано в алгоритме BBO путем вероятностного увеличения обмена признаками решений между схожими решениями. Это аналогично скрещиванию на основе видов, также называемому нишированием, в генетических алгоритмах [Stanley и Mukkulainen, 2002], а также аналогично видообразую-

щей островной модели [Gustafson и Burke, 2006] (см. раздел 8.6.2). Это еще аналогично идее окрестностей в оптимизации на основе роя частиц [Kennedy и Eberhart, 2001] (см. главу 11). Однако мотивация и механизм совершенно разные. Ниширование в генетических алгоритмах основано на вероятности, что особи будут спариваться с похожими на них особями. Окрестности в оптимизации на основе роя частиц (PSO) опираются на вероятности, что особи будут собираться вместе в пространстве решений. Архипелаги в алгоритме BBO формируются на вероятности скопления схожих островов. Количественный способ определения влияния островной изоляции на скорость иммиграции приведен в публикации [Hanski, 1999].

- **Первоначальная иммиграция**: классическая теория островной биогеографии показывает, что скорость иммиграции уменьшается по мере увеличения числа видов, как показано на рис. 14.1 и 14.5. В алгоритме BBO это соответствует монотонному снижению скорости иммиграции по мере увеличения приспособленности особей. Это означает, что, по мере того как особь становится более приспособленной, вероятность встраивания признаков от других особей уменьшается. Однако более поздние достижения в области биогеографии указывают на то, что в случае некоторых видов-первопроходцев (например, растений) первоначальное увеличение численности видов приводит к первоначальному увеличению скорости иммиграции [Wu и Vankat, 1995]. Причина в том, что эти ранние иммигранты модифицируют остров, делая его более гостеприимным для других видов. То есть положительный эффект увеличения многообразия за счет первоначальной иммиграции преодолевает отрицательный эффект увеличения численности популяции. В алгоритме BBO это соответствовало бы первоначальному увеличению скорости иммиграции, по мере того как очень плохое кандидатное решение первоначально улучшает свою приспособленность. В алгоритме BBO это можно рассматривать как временный механизм положительной обратной связи. Очень слабая особь принимает признаки других особей, увеличивая свою приспособленность, что впоследствии увеличивает ее вероятность принятия еще большего числа признаков от других особей. Это показано на рис. 14.10. Данная идея может быть встроена и в другие эволюционные алгоритмы, но ее первоначальная мотивация исходит из биогеографии [Ma и Simon, 2011a].

Рис. 14.10. Предлагаемая модель показывает, что первоначально иммиграция возрастает вместе с приспособленностью. Это дает слабым, но улучшающимся особям импульс к дальнейшему улучшению. По мере того как особь продолжает становиться более приспособленной после первоначального увеличения иммиграции, иммиграция начинает уменьшаться, давая менее приспособленным особям относительно более высокие возможности для иммиграции хороших признаков решения

- **Минимальное требование к пригодности**: можно предположить, что среда обитания должна иметь некоторый минимальный ранг пригодности, для того чтобы иметь ненулевой уровень эмиграции. Это похоже на предположение о том, что остров должен иметь ненулевой индекс пригодности естественной среды обитания (HSI) для поддержки любого вида [Hanski и Gilpin, 1997].
- **Возрастной критерий**: репродуктивная ценность особи (то есть ожидаемое число детей в единицу времени) является треугольной функцией от ее возраста. Репродуктивная ценность низка в молодом возрасте из-за незрелости, высока в детородном возрасте и снова низка в старости из-за потери фертильности. То же самое можно сказать и о видах. Молодой вид вполне может быть плохо адаптирован к окружающей среде и поэтому имеет лишь небольшую вероятность видообразования, средневозрастной вид достаточно зрелый и достаточно динамичный для видообразования, а старый вид слишком застойный для этого. Эта идея может привести к введению в алгоритм BBO возрастного критерия, ана-

логичного тому, который использовался в генетических алгоритмах [Zhu и соавт. 2006].

- **Мобильность видов**: классическая теория островной биогеографии исходит из того, что все виды равны в своей миграционной способности. В действительности некоторые виды более подвижны, чем другие, некоторые виды – более качественные рассеиватели, чем иные. В биогеографии предпринимаются усилия по встраиванию видоспецифических характеристик в теорию островной биогеографии [Lomolino, 2000b]. Алгоритм BBO в настоящее время исходит из того, что все виды одинаково мобильны. Алгоритм BBO будет в большей степени соответствовать своей мотивационной структуре, если мобильность видов будет пропорциональна вкладу вида в приспособленность решения. То есть, учитывая популяцию особей, статистические методы могут использоваться для нахождения корреляции каждого признака решения с приспособленностью. В этом случае мобильность будет определяться как признак решения или множество признаков, которые положительно коррелируют с приспособленностью. Подвижность видов алгоритма BBO будет следовать теории биогеографии путем присвоения значений подвижности с помощью гауссова распределения [Lomolino, 2000b]. Те признаки решения, которые способствуют росту приспособленности, будут эмигрировать с большей вероятностью. Этот подход должен улучшить среднюю приспособленность популяции.
- **Отношения хищник/жертва**: в биологии некоторые виды имеют состязательные отношения. Эти отношения не обязательно наносят вред кормовым видам. Например, жертва может реагировать на хищников, сокращая эксплуатацию своих ресурсов, тем самым принося пользу себе в долгосрочной перспективе [Hanski и Gilpin, 1997]. Вместе с тем более распространенным сценарием является тот, в котором хищники уменьшают жертву до такой степени, что одна или обе популяции сталкиваются с вымиранием. Отношения хищник/жертва могут быть установлены из популяции алгоритма BBO путем изучения особей и отмечая, какие пары признаков решения имеют низкую вероятность сосуществования. Затем эти признаки решения будут смоделированы как пара хищник/жертва. Объединение данной информации со вкладом каждого вида в приспособленность приведет к определению признака решения у хищника как соперника, который положительно кор-

релирует с приспособленностью, и признака решения у жертвы как соперника, который отрицательно коррелирует с приспособленностью. Отношение хищник/жертва вполне может привести к ненулевой равновесной популяции либо к вымиранию одной или обеих популяций [Gotelli, 2008], [Hanski и Gilpin, 1997]. Эта информация может быть использована во всей популяции особей, для того чтобы увеличить вероятность наличия признаков хищника и уменьшить вероятность наличия признаков жертвы. Большинство моделей хищник/жертва в биологии предназначено для двух видов. Эти модели можно было бы использовать в алгоритме BBO, однако более полное описание можно было бы получить, если бы существующие модели хищник/жертва можно было распространить на многовидовые системы.

- **Ресурсная конкуренция**: в отличие от отношения хищник/жертва, описанного выше, мы отмечаем, что аналогичные виды конкурируют за аналогичные ресурсы. Поэтому маловероятно, что многие схожие виды занимают один и тот же остров, в особенности если они имеют большие популяции [Tilman и соавт., 1994]. В алгоритме BBO это означает, что вряд ли признаки решения будут эмигрировать на острова, на которых уже есть особи, похожие друг на друга. С другой стороны, это может означать, что скорость эмиграции не затрагивается, но вероятность выживания ниже. Ресурсная конкуренция в алгоритме BBO также означает, что если два признака решения имеют равную вероятность исчезновения, то признак, наиболее похожий на другие признаки решения, с большей вероятностью исчезнет. Это другой тип взаимодействия, чем описанное выше отношение хищник/жертва. Однако обе модели правдоподобны, и конкуренция, как правило, рассматривается в биологии как более значительный фактор формирования сообщества, чем взаимодействия хищника и жертвы.
- **Временная корреляция**: если в островной биогеографии вид мигрирует на остров в определенном географическом направлении, то он, скорее всего, продолжит движение в этом направлении к следующему острову. Это происходит потому, что на миграцию влияют преобладающие ветры и течения, и эти ветры и течения имеют положительную временную корреляцию. Это описывается теорией биодиффузии, телеграфным уравнением и уравнением диффузии [Okubo и Levin, 2001]. Если вид мигрирует с острова A на остров B, то на следующем временном шаге он, скорее всего,

продолжит движение в том же направлении к следующему острову в цепи. В алгоритме BBO это означает, что если признак решения мигрирует от одной особи к другой, то он, скорее всего, продолжит мигрировать в этом направлении в следующем эволюционном поколении. Понятие «направление» в алгоритме BBO может быть определено в терминах местоположения решения, где местоположение определяется как точка в пространстве признаков решения.

Другие аспекты биогеографии могут вдохновить на создание иных вариаций алгоритма BBO. Литература по биогеографии настолько богата, что существует много возможностей в этом направлении.

Задачи

Письменные упражнения

14.1. Мы написали уравнение (14.1), приняв допущение, что Δt достаточно мало, для того чтобы можно было игнорировать вероятность более чем одной миграции между временем t и $(t + \Delta t)$. Перепишите уравнение с допущением, что между t и $(t + \Delta t)$ может произойти не более двух миграций.

14.2. Мы определили селекционное давление для эволюционных алгоритмов в уравнении (8.13). Как изменить миграционные кривые на рис. 14.2, чтобы увеличить селекционное давление?

14.3. Как изменить алгоритм BBO на рис. 14.3, чтобы он был стационарным и не поколенческим?

14.4. Мы, как правило, устанавливаем $\mu_k = r_k/(N + 1)$ и $\lambda_k = (N + 1 - r_k)/(N + 1)$, где r_k – это ранг k-й особи, лучшая особь имеет ранг N, и худшая особь имеет ранг 1. Это означает, что $\lambda_k \in [1/(N + 1), N/(N + 1)]$. Каков практический результат применения $\lambda_k > 0$ для лучшей особи?

14.5. Данная задача исследует первоначальную модель иммиграции рис. 14.10. Пусть β обозначает нормализованное значение приспособленности при максимальном значении λ.

а) Найдите уравнение скорости иммиграции на рис. 14.10.

б) Найдите равновесную численность видов для рис. 14.10.

14.6. Предположим, что мы используем линейные скорости миграции с размером популяции N, так что $\lambda_1 = 1/(N+1)$ для лучшей особи и $\lambda_N = N/(N+1)$ для худшей особи.

a) Какова вероятность того, что лучшая особь x_b получит иммиграцию хотя бы одного признака решения в алгоритме на основе частичной иммиграции, представленном на рис. 14.3?

b) Какова вероятность того, что лучшая особь x_b получит иммиграцию хотя бы одного признака решения от особи, отличной от нее самой, в алгоритме на основе частичной эмиграции, представленном на рис. 14.6?

14.7. Обратитесь к приложению C.1 и найдите там пример разделимой функции и пример неразделимой функции.

Компьютерные упражнения

14.8. Повторите пример 14.1 для $n = 5$.

14.9. Стандартный алгоритм BBO на рис. 14.3 показывает, что для каждой особи x_k мы должны назначить вероятность эмиграции μ_k $\propto$ приспособленность x_k, с $\mu_k \in [0, 1]$. Мы используем вероятности эмиграции, для того чтобы отобрать эмигрирующую особь, и для этой операции мы можем использовать следующие ниже варианты.

- Ранговый отбор, как описано в разделе 8.7.4. Это стандартный для алгоритма BBO вариант, как указано в примере 14.2.
- Квадратичное ранжирование, которое также обсуждается в разделе 8.7.4.
- Турнирный отбор, как описано в разделе 8.7.6.
- Племенной отбор, как разобрано в разделе 8.7.7.

Реализуйте алгоритм BBO для минимизации 10-мерной функции Экли с $N = 50$, лимитом поколений = 50, скоростью мутации = 1 % и параметром элитарности = 2. Выполните алгоритм BBO для 20 симуляций Монте-Карло, отслеживая самую низкую стоимость в каждом поколении для каждой симуляции Монте-Карло. Постройте график наименьшей стоимости, усредненно по симуляциям Монте-Карло, как функции от номера поколения. Сравните графики алгоритма BBO для каждого из четырех вариантов отбора эмиграции, упомянутых выше. Прокомментируйте свои результаты.

14.10. Постройте графики своих ответов на задачу 14.6 как функции от размера популяции N для $n \in [10, 50]$ и размерности задачи $n = 10$. Прокомментируйте свои результаты.

14.11. Одним из потенциальных способов повышения результативности алгоритма BBO является отбор следующего поколения из старых и новых особей [Du и соавт., 2009]. То есть инструкцию $\{x_k\} \leftarrow \{z_k\}$ в конце рис. 14.3 можно заменить следующей:

$$\{x_k\} \leftarrow \text{Лучшие } N \text{ особей из } \{x_k\} \cup \{z_k\}.$$

Это решение мотивировано эволюционной стратегией, и поэтому мы можем назвать данную модификацию эволюционной стратегией BBO-ES. Реализуйте алгоритм BBO для минимизации 10-мерной функции Экли с популяцией размером $N = 50$, лимитом поколений = 50, скоростью мутации = 1 % и параметром элитарности = 2. Выполните алгоритм BBO для 20 симуляций Монте-Карло, отслеживая самую низкую стоимость в каждом поколении для каждой симуляции Монте-Карло. Постройте график наименьшей стоимости, усредненно по симуляциям Монте-Карло, как функции от номера поколения. Сравните график из стандартного алгоритма BBO с алгоритмом BBO-ES. Прокомментируйте свои результаты.

Глава 15

Культурные алгоритмы

Культура оптимизирует познание.

– Джеймс Кеннеди [Kennedy, 1998]

Суть приведенной выше цитаты заключается в том, что познание (то есть процесс мышления) включает в себя больше, чем активность мозга и нейрональное поведение. Наше мышление находится под влиянием нашей культуры. Кроме того, это влияние выгодно (даже оптимально, согласно приведенной выше цитате). Без культуры наши когнитивные способности были бы ослаблены.

Данная идея иллюстрируется обнаружением одичавших детей [Newton, 2004]. Некоторые из этих детей растут в дикой природе, в то время как другие растут в изоляции в результате жестокости опекунов. Дети, которые растут без какого-либо социального или культурного взаимодействия, как правило, никогда не учатся ассимилировать общество, никогда не учатся разговаривать, никогда не учатся устанавливать связи с другими и никогда не учатся действовать социально приемлемым образом. Их неспособность научиться функционировать в своей новой и цивилизованной среде не является генетической; она не обусловлена отсутствием врожденного интеллекта. Отсутствие социальных и культурных влияний во время их воспитания серьезно подрывает их интеллект. Одичавшие дети являются весомым аргументом в пользу воспитания в дискуссии на тему «природа против воспитания».

Ученые привыкли считать, что человеческая культура возникла на высоком уровне, а затем деградировала до более низких уровней. Это вырождение якобы привело к низким и нецивилизованным культурам, разбросанным по всему миру. Эта вера была в значительной степени основана на религиозных рассказах о сотворении и библейской

истории Вавилонской башни в Книге Бытия. Однако такой взгляд на культурную дегенерацию имеет проверяемые последствия. Например, дегенерационизм должен привести к археологическим данным о повышении культурной изощренности, по мере того как раскопки уходят все глубже и дальше назад, в прошлое. Хотя конкретные религиозные истории не могут быть научно проверены, тестонепригодность дегенерационизма как общего принципа сыграла важную роль в его упадке как социологической теории.

Эдвард Тайлор, антрополог XIX века, показал, что передовые культуры развивались из примитивных культур, а не наоборот [Tylor, 2011]. Он показал, что культура эволюционирует от низших форм к высшим так же, как эволюционируют биологические органы. Тайлор первым использовал слово культура в современном социологическом смысле и определил его как «то сложное целое, которое включает в себя знания, убеждения, искусство, мораль, право, обычаи и любые другие способности и привычки, приобретенные человеком как членом общества» [Tylor, 2009].

Культура общества – это сложное образование, которое взаимодействует с окружающей средой, отдельными народами и другими культурами. Люди могут действовать независимо, но они также взаимодействуют друг с другом, как прямо, так и косвенно; люди влияют друг на друга напрямую, и они влияют друг на друга косвенно через культуру. Большинство людей ограничено культурой, в которой они живут. Некоторые люди плывут против течения, но большинство приспосабливается к обществу.

Краткий обзор главы

В этой главе рассматриваются некоторые способы моделирования культуры в эволюционных алгоритмах с целью повышения их результативности. Раздел 15.1 является предварительным, в нем обсуждаются оптимальные стратегии человеческих отношений на высоком уровне; он дает некоторую мотивацию и основу для остальной части главы. В разделе 15.2 обсуждается конкретная модель культуры, называемая пространствами убеждений, и обсуждается их совместная эволюция с кандидатными решениями в эволюционных алгоритмах. В разделе 15.3 используются пространства убеждений для разработки культурной эволюционной программы (cultural EP) и показывается, что она обеспечивает более высокую результативность, чем стандартная эволюционная программа. В разделе 15.4 принимается другая перспектива культуры,

и в нем она рассматривается как более межличностная и ориентированная на отношения; рассматривается адаптивная модель культуры (ACM) и показывается, как адаптивная модель культуры может решать задачу коммивояжера.

15.1. Сотрудничество и конкуренция

В этом разделе культура рассматривается в смысле межличностных отношений. Современное общество предполагает много межличностного общения, и количество коммуникаций растет быстрыми темпами. Общество также включает в себя много сотрудничества и много конкуренции. Иногда мы общаемся с целью сотрудничества, но иногда – с целью конкуренции. Когда мы пишем технический документ или исследовательское предложение, мы сообщаем о недостатках сопутствующих идей и преимуществах наших собственных идей. Иногда мы преувеличиваем, чтобы выставить себя в лучшем виде или кого-то другого в худшем. Иногда наши преувеличения преднамеренны, иногда они непреднамеренны, а иногда трудно различить наши собственные намерения. В общении с другими мы иногда говорим правду, потому что надеемся, что другие, в свою очередь, тоже скажут нам правду. Но мы часто лжем, когда польза или вознаграждение перевешивает последствия. Рассмотрим типичные ответы на следующие ниже вопросы.

1. Как поживаете?
2. Тебе понравилась блюдо, которое я тебе приготовила?
3. Как часто вы лжете?

Мы рассуждаем так, что тот, кто спрашивает, на самом деле не хочет узнать ответ на свой вопрос. Тот, кто спрашивает, только пытается поговорить или выуживает конкретный ответ, и поэтому мы охотно делаем одолжение, хотя наш ответ технически мог бы быть классифицирован как ложь. Что интересно, все думают, что они лгут реже, чем другие [DePaulo и соавт., 1996]. Более того, все считают, что их собственная ложь более оправдана, чем ложь других.

Мы можем просимулировать межличностную коммуникацию в эволюционных алгоритмах с целью изучения коммуникационных стратегий или с целью поиска решений оптимизационных задач. Дилемма заключенного (см. раздел 5.4) является одним из примеров общения агентов друг с другом. Это интересный пример, потому что он имеет много вариантов, причем лучшая стратегия зависит от стратегии противоположного игрока, и лучшая стратегия не всегда очевидна.

Бар «Эль Фароль»

Еще одним интересным примером межличностного общения является задача из теории игр «Эль Фароль» [Kennedy и Eberhart, 2001, глава 5]. Эта задача связана с человеком по имени Брайан Артур[1], который любит ходить в паб под названием «Эль Фароль» в центре Санте-Фе. Он особенно любит ходить по четвергам, когда в пабе «Эль Фароль» играют ирландскую музыку. Вместе с тем он предпочитает оставаться дома, если паб переполнен. В частности, он хочет пойти в «Эль Фароль», если там будет меньше 60 человек, но он хочет остаться дома, если там будет 60 человек и более. Друзья Брайана в такой же ситуации. Они любят ходить в «Эль Фароль», если там меньше 60 человек, но они не хотят туда идти, если там будет 60 человек или больше.

Брайан и его друзья знают, что за последние 14 недель количество людей в пабе было

44, 78, 56, 15, 23, 67, 84, 34, 45, 76, 40, 56, 22 и 35. (15.1)

Следует ли ему идти в этот четверг? Другими словами, основываясь на данных за последние 14 недель, будет ли на этой неделе в «Эль Фароле» менее 60 человек? Он может применить различные методы распознавания образов и регрессионные тесты, чтобы попытаться предсказать количество людей, которые будут в «Эль Фароле» в этот четверг. Если он найдет хороший предиктор, то его задача будет решена. Однако если он расскажет всем своим друзьям о своем предикторе, то предиктор больше не будет работать. Если все его друзья узнают, что его алгоритм предсказал менее 60 человек, то все они отправятся в Эль Фарол, и там будет более 60 человек. Если его алгоритм предскажет более 60 человек, то все его друзья останутся дома, и их будет меньше 60 человек. В этом парадокс хорошего предсказательного алгоритма, когда учитывается человеческий фактор. Хороший предиктор становится плохим предиктором.

Теперь предположим, что Брайан и его друзья разговаривают друг с другом о том, пойдут они в четверг в «Эль Фароль» или нет. Если Брайан решит отправиться в «Эль Фароль» и расскажет об этом всем своим друзьям, то они с большей вероятностью останутся дома, и Брайан будет вознагражден небольшой толпой в «Эль Фароле». Если же Брайан решит остаться дома и расскажет об этом всем своим друзьям, то они с большей вероятностью пойдут, а его друзья будут наказаны большой толпой.

[1] Брайан Артур (Brian Arthur) – экономист и соучредитель Института Санте Фе.

Вместе с тем Брайан и его друзья могут быть друг с другом не совсем честны. Они все могут сказать друг другу, что собираются в «Эль Фароль» в надежде, что большинство других останется дома. Сказав всем своим друзьям, что он идет, Брайан может решить остаться дома, если он услышит, что все остальные планируют пойти. Вдобавок Брайан вполне может сказать своим друзьям, что он и 10 его друзей идут. Брайан вполне может преувеличить размер своей вечеринки, чтобы побудить друзей остаться дома. Конечно, его друзья могут следовать той же стратегии. То есть они вполне могут солгать ради собственной выгоды.

Какова оптимальная стратегия общения для Брайана? Должен ли он всегда говорить правду? Если он постоянно лжет, то его друзья в конечном итоге узнают его схему лжи и научатся ее игнорировать. Однако если его конечная цель состоит в том, чтобы ходить в «Эль Фароль» в малолюдные вечера и избегать посещений «Эль Фароля», когда там переполнено, то, по-видимому, ему придется лгать от случая к случаю.

Задача паба «Эль Фароль» интересна тем, что она включает в себя истину, ложь, доверие, общение и, возможно, противоречивые цели. Если цель Брайана – понравиться его друзьям, то он, вероятно, будет говорить правду все время. Если его цель состоит в том, чтобы ходить в «Эль Фарол» в малолюдные вечера и избегать посещений паба в переполненные вечера, то он вполне может иногда говорить неправду.

Другие примеры

Другая причина, по которой задача «Эль Фароль» вызывает интерес, заключается в том, что, как упоминалось ранее, хороший предиктор станет плохим предиктором, если все его используют. Эта характеристика возникает во многих реальных ситуациях. Например, рассмотрим романтический интерес, который мальчик проявляет к девушке. Если он проявит слишком большой интерес, то тем самым покажет свое отчаяние, что будет непривлекательно для девушки. Однако если его интерес недостаточно очевиден, то он будет выглядеть незаинтересованным, что не будет способствовать отношениям с девушкой его мечты. Более того, как девушка должна интерпретировать это явное отсутствие заинтересованности? Должна ли она интерпретировать это как истинную незаинтересованность, и в этом случае она обратит свое внимание на других женихов? Или она должна интерпретировать его отсутствие интереса как подавленную страсть, и в этом случае она ответит? Ухаживание – это сложное компромиссное взаимоотношение между двумя

людьми, которые ведут себя в соответствии с их собственными целями, но также и в соответствии с их культурой.

Еще один интересный пример – это бейсбол. Когда счет – три бола и два страйка, питчеру, то есть подающему, нужно бросить страйк, чтобы избежать ухода бэттера, то есть бьющего, из игры, в особенности если базы загружены в конце ничейной игры. Но бэттер знает, что питчеру нужно бросить страйк. Это, похоже, дает преимущество бэттеру, потому что он приблизительно знает, куда мяч будет брошен. Но поскольку питчер знает о мыслительном процессе бэттера, питчер вполне может доставить бросок вне страйк-зоны, пытаясь заманить бэттера взмахнуть битой в сторону плохого броска. Разумеется, бэттер тоже догадывается о мыслительном процессе питчера. Бейсбол становится не только физическим соревнованием, но и психологическим противоборством. Бэттер должен решить, насколько агрессивным он будет в ожидании броска в страйк-зону. Питчер должен решить, может ли он рискнуть броском вне страйк-зоны. При принятии решения о своих стратегиях игроки учитывают не только игровую ситуацию, но и историю предыдущих встреч со своим противником.

Другие примеры возникают в нашем бизнесе и в наших исследовательских программах. В какой области мы должны концентрироваться с нашими исследовательскими предложениями? Должны ли мы писать предложения по высокофинансируемым направлениям? Большие объемы финансирования увеличивают наши шансы, но все остальные тоже пишут предложения по этим направлениям, что снижает наши шансы. Возможно, нам следует писать предложения по направлениям с меньшим финансированием, с тем чтобы у нас было меньше конкуренции. Если наше предложение является единственным в определенной области, то оно, скорее всего, будет профинансировано. Но если наши конкуренты будут следовать той же стратегии, то наша стратегия потерпит неудачу. Принятие решений о том, где сосредоточить наши усилия по подготовке предложений, является сложным и многоаспектным, но оптимальной стратегией, вероятно, является распределение наших усилий между областями с высоким и низким уровнями риска, а также областями с высоким и низким уровнями финансирования [Simon, 2005]. Аналогичная смешанная стратегия может быть использована для инвестирования (вспомним мантру инвестора о диверсификации), маркетинга продукции и других приложений.

Проблемы межличностных отношений, общения, сотрудничества, обмана и многочисленных целей имеют много общего с человеческой культурой. Люди научились почти оптимальным способам установле-

ния взаимоотношений, структурирования общества и развития культуры. Мы не знаем всех особенностей оптимизации, которые присущи человеческой культуре, но рассмотрение возможности таких особенностей, несомненно, было бы увлекательным и полезным исследованием. Имитация и симулирование оптимизационного поведения человеческой культуры – еще одно увлекательное и полезное исследование, и мы обратим наше внимание на это стремление в остальных разделах этой главы.

15.2. Пространства убеждений в культурных алгоритмах

Культурный алгоритм (cultural algorithm, CA) аналогичен другим эволюционным алгоритмам в рассмотрении кандидатных решений оптимизационной задачи как особей. Вместе с тем культурный алгоритм также рассматривает принципы, которые определяют эволюцию особей как их культуру. Культурный алгоритм моделирует влияние между людьми и их культурой с целью получить оптимизационный алгоритм. Культурные нормы виртуального общества культурного алгоритма иногда называются пространством убеждений (belief space). В каждом культурно-алгоритмическом поколении особи рекомбинируют и мутируют, как и в других эволюционных алгоритмах, о которых мы говорили ранее в этой книге. Но в культурном алгоритме на эту рекомбинацию и мутацию влияет пространство убеждений. Пространство убеждений может быть спроектировано программистом для наложения ограничений либо для предпочтения благоприятных признаков в популяции, либо для предотвращения нежелательных признаков.

■ Пример 15.1

В данном примере мы обсудим общую идею относительно того, как пространство убеждений могло бы быть реализовано в эволюционном алгоритме. Вспомните задачу искусственного муравья в разделе 5.5. Искусственный муравей помещается в решетку с пустыми клетками и клетками, заполненными пищей. Единственная сенсорная способность муравья – ощущать присутствие либо отсутствие пищи в клетке непосредственно перед ним. В каждой клетке муравей может совершить одно из трех действий: он может либо двигаться вперед, в этом случае он съест пищу в следующей клетке, если пища присутствует; либо он может остаться в своей текущей клетке и повернуть направо; либо он

может остаться в своей текущей камере и повернуть налево. Мы хотим эволюционно сформировать конечный автомат (FSM), для того чтобы помочь муравью ориентироваться в решетке и потреблять как можно больше пищи. Интуитивно мы знаем, что если муравей ощущает пищу в клетке перед собой, то он, вероятно, должен двигаться вперед и съесть пищу. Однако мы не хотим навязывать это действие как жесткое ограничение. Мы знаем, что эволюция часто требует разведывания неоптимальных решений в поисках оптимума. Поэтому мы хотели бы *стимулировать* каждый конечный автомат в нашей популяции эволюционного алгоритма двигаться вперед всякий раз, когда ощущается пища, но мы не хотим делать это строгой обязанностью. Это именно тот тип поведения, который может быть закодирован в пространстве убеждений, чтобы поощрять, но не требовать определенный признак в эволюционно-алгоритмической популяции.

Уровень стимулирования и поощрения, который мы оказываем эволюционно-алгоритмическим особям двигаться вперед, когда ощущается пища, представляет силу культуры. В человеческом обществе некоторые культуры сильнее других и оказывают большее давление, чтобы ей подчиняться. Если наше стимулирование будет умеренным, то мы позволим многим особям разведывать другие варианты и не продвигаться вперед, когда ощущается пища. Если наше стимулирование будет сильным, то мы позволим лишь очень немногим особям разведывать альтернативные варианты. По мере того как популяция эволюционирует, мы, возможно, захотим изменить наш уровень стимулирования, опираясь на приспособленность особей, которые подчиняются преобладающей культуре, в сопоставлении с приспособленностью тех особей, которые выступают против преобладающей культуры, принимая альтернативное действие при ощущении пищи.

■

Пример 15.1 показывает, что культурные алгоритмы могут реализовывать двойное наследование: признаки решения наследуются детьми от родителей, а пространство убеждений одного поколения наследуется у предыдущего поколения. Эволюция по-прежнему происходит на индивидуальном уровне, но эволюция находится под влиянием пространства убеждений.

Пространство убеждений культурного алгоритма вполне может быть статическим или динамическим в зависимости от того, как мы его реализуем. Если пространство убеждений статично, то оно не изменяется со временем. Если пространство убеждений динамично, то оно

изменяется со временем, то есть культура может эволюционировать. Культурный алгоритм с динамическим пространством убеждений из поколения в поколение эволюционно формирует не только популяцию особей, но и пространство убеждений. В культурном алгоритме с динамическим пространством убеждений пространство убеждений не только влияет на эволюцию популяции, но популяция, в свою очередь, влияет на эволюцию пространства убеждений. Динамические пространства убеждений обусловлены тем, что мы наблюдаем в человеческом обществе. Мы видим, что культура эволюционирует гораздо быстрее, чем биология. Поэтому мы надеемся, что сможем найти оптимальные решения быстрее в эволюционном алгоритме с динамическим пространством убеждений, чем в эволюционном алгоритме без него.

Подобно тому, как существует много теорий о человеческой культуре [Welsch и Endicott, 2005], точно так же существует несколько разных типов культурных алгоритмов. Рисунок 15.1 схематично показывает базовый культурный алгоритм. Как и в случае с любым другим эволюционным алгоритмом, мы инициализируем популяцию кандидатных решений, но также инициализируем пространство убеждений B. Пространство убеждений влияет на эволюцию популяции; можно сказать, что оно направляет эволюционный процесс. Показанный на рис. 15.1 алгоритм работает так же, как и любой другой эволюционный алгоритм, оценивая стоимость каждой эволюционно-алгоритмической особи. Но затем он использует особи для модификации B. Существует много вариантов того, как можно реализовать модификацию B. Например, если популяция указывает на то, что большинство хороших особей имеет определенный признак, то B вполне может обновляться со смещением будущих кандидатных решений в сторону этого признака. Конечно, будущие особи уже смещены в сторону этого признака в силу того факта, что приспособленные особи с большей вероятностью рекомбинируют, чем неприспособленные. Но B также может смещать будущих особей более сложными способами, чем стандартные методы рекомбинации. Например, мы можем встроить в B определенные комбинации признаков или типы поведения. (Вспомните идею в примере 15.1 со смещением конечных автоматов на основе искусственного муравья в сторону решений, которые двигают вперед, если ощущается пища.)

Инициализировать совокупность кандидатных решений $\{x_i\}$, $i \in [1, N]$.
Инициализировать пространство убеждений B.
While not (критерий останова)

Рассчитать стоимость $f(x_i)$ каждой особи в популяции, $i \in [1, N]$.
Использовать популяцию $\{x_i\}$ для обновления B.
Встроить B в рекомбинацию и мутацию популяции $\{x_i\}$.
Next поколение

Рис. 15.1. Схематичное представление базового культурного алгоритма, основанного на публикациях [Reynolds, 1994], [Engelbrecht, 2003, глава 14]

После обновления B на рис. 15.1 мы выполняем рекомбинацию и мутацию популяции $\{x_i\}$. Данный шаг может быть выполнен с помощью любого из эволюционных алгоритмов, которые мы обсуждаем в этой книге. Следовательно, культурный алгоритм нужно рассматривать не как отдельный эволюционный алгоритм, а как способ расширения других эволюционных алгоритмов либо как эволюционный метаалгоритм. Отличительной особенностью культурных алгоритмов является то, что рекомбинация и мутация находятся под влиянием пространства убеждений B; особи в следующем поколении, как правило, согласуются с B.

В культурном алгоритме рис. 15.1 есть много деталей, которые требуют доработки. Например:

- какую информацию мы будем кодировать в пространстве убеждений B?
- как мы будем обновлять B?
- какой тип рекомбинации и мутации мы будем использовать? То есть какой эволюционный алгоритм мы будем использовать в качестве базового для культурного алгоритма?
- как мы будем использовать B для влияния рекомбинации и мутации?

Все эти вопросы предоставляют исследователю возможность найти ответы, которые будут эффективными для конкретных задач или для общих классов задач. Например, рассмотрим первый вопрос из приведенного выше списка. Мы можем частично ответить на этот вопрос, отметив, что пространство убеждений в культурном алгоритме может представлять следующие аспекты оптимизационной задачи.

- Пространство убеждений может представлять ограничения для решения оптимизационных задач с жесткими ограничениями или с мягкими ограничениями [Coello Coello и Becerra, 2002], [Becerra и Coello Coello, 2004].
- Убеждение может представлять специфичные для конкретной задачи знания с целью склонения поиска в предпочтительные

направления, которые основаны на человеческом опыте [Sverdlik и Reynolds, 1993], [Alami и El Imrani, 2008].

- Пространство убеждений может включать важность многообразия с целью оказания помощи в поддержании многообразия в поиске.
- Пространство убеждений может включать важность сотрудничества с целью повышения результативности коэволюционных систем. Коэволюция включает в себя развитие особых, но взаимодействующих эволюционных систем в общей среде [Durham, 1992]. Мы не обсуждаем коэволюцию в этой книге, но искусственная коэволюция может находить оптимальные решения, когда оценивания приспособленности варьируются со временем (см. раздел 21.2) либо когда оценивания приспособленности популяции кандидатных решений зависят от других популяций, которые сами изменяются со временем [Yang и соавт. , 2008].
- Пространство убеждений может включать в себя важность изобретательности, что склоняет поиск эволюционного алгоритма в сторону новых кандидатных решений либо в сторону неразведанных участков поискового пространства. Эти идеи встроены в оппозиционное самообучение (см. главу 16) и поиск новизны [Lehman и Stanley, 2011], но пока еще не встроены в пространства убеждений культурного алгоритма.

15.3. Культурно-эволюционное программирование

В этом разделе показано, как простое пространство убеждений может улучшать результативность алгоритма эволюционной программы (evolutionary program, EP). Базовый алгоритм EP представлен на рис. 5.1. Здесь мы включаем возможность реализации пространства убеждений в алгоритме EP. Пространство убеждений указывает на то, где в поисковом пространстве находятся наиболее приспособленные кандидатные решения. Эволюционная программа в условиях культурной программы (CA-influenced EP, CAEP), которую мы представим в этом разделе, аналогична той, что обсуждалась в публикации [Engelbrecht, 2003, глава 14]. Пространство убеждений B кодируется как $2n$ параметров, где n – это размерность оптимизационной задачи. Интервал $[B_{\min}(k), B_{\max}(k)]$ указывает, где в поисковом пространстве по убеждениям преобладающей культуры лежит k-я размерность хороших решений. Пространство

убеждений воздействует на мутационный процесс алгоритма EP. Если в мутационном методе алгоритма EP, показанном на рис. 5.1, $\beta = 0$, то мы имеем

$$x'_i(k) \leftarrow x_i(k) + r_i(k)\sqrt{\gamma}, \tag{15.2}$$

для $i \in [1, N]$ и $k \in [1, n]$, где N – это размер популяции, n – размерность задачи, $r_i(k) \sim N(0, 1)$ и γ – мутационная дисперсия. В алгоритме CAEP уравнение (15.2) заменяется на

$$\Delta_i(k) \leftarrow \begin{cases} B_{\min}(k) - x_i(k) & \text{если } x_i(k) < B_{\min}(k) \\ 0 & \text{если } B_{\min}(k) \le x_i(k) \le B_{\max}(k) \\ B_{\max}(k) - x_i(k) & \text{если } B_{\max}(k) = x_i(k) \end{cases}$$

$$x'_i(k) \leftarrow x_i(k) + r_i(k)\sqrt{\gamma} + \Delta_i(k) \tag{15.3}$$

Мы видим, что, если k-я размерность особи x_i находится внутри пространства убеждений, то ее мутированная версия $x'_i(k)$ является случайной величиной со средним значением $x_i(k)$. Однако если $x_i(k)$ находится вне пространства убеждений, то ее мутированная версия является случайной величиной со средним значением $B_{\min}(k)$ либо $B_{\max}(k)$, в зависимости от того, какое будет ближе к $x_i(k)$. Рисунок 15.2 иллюстрирует эту идею. Данный рисунок показывает, что когда $x_i(k)$ находится внутри пространства убеждений (левая часть рисунка), то она мутирует стандартным способом. Однако когда $x_i(k)$ находится вне пространства убеждений (правая часть рисунка), ее мутированная версия центрируется на ближайшем краю пространства убеждений. Мутированная версия может в итоге оказаться вне пространства убеждений – на самом деле у нее есть по крайней мере 50%-ный шанс оказаться вне $[B_{\min}(k), B_{\max}(k)]$. Вместе с тем она также имеет почти 50%-ный шанс находиться внутри пространства убеждений, что намного выше, чем шанс, если бы среднее значение мутации не было сдвинуто.

Теперь мы обсудим, как обновлять пространство убеждений в алгоритме CAEP. Есть несколько способов сделать это. Например, для обновления пространства убеждений мы можем использовать лучших M особей. Сначала мы находим минимальное и максимальное значения каждой размерности лучших M особей:

$$\begin{aligned} x_{k,\min} &\leftarrow \min\{x_j(k) : j \in [1, M]\} \\ x_{k,\max} &\leftarrow \max\{x_j(k) : j \in [1, M]\} \end{aligned} \tag{15.4}$$

для $k \in [1, n]$, где особи индексируются от лучших к худшим, так что $\{x_j(k) : j \in [1, M]\}$ включает в себя лучших M особей в популяции. Теперь

мы используем минимальные и максимальные значения областей, для того чтобы оказывать воздействие на пространство убеждений из поколения в поколение:

$$\begin{aligned} B_{\min}(k) &\leftarrow \alpha B_{\min}(k) + (1-\alpha)x_{k,\min} \\ B_{\max}(k) &\leftarrow \alpha B_{\max}(k) + (1-\alpha)x_{k,\max} \end{aligned} \tag{15.5}$$

для $k \in [1, n]$. Параметр $\alpha \in [0, 1]$ является инерцией пространства убеждений, и он определяет степень застойности пространства убеждений из поколения в поколение. Уравнение (15.5) показывает, что если $\alpha = 1$, то пространство убеждений ни разу не меняется. Если $\alpha = 0$, то пространство убеждений целиком определяется текущей популяцией и не зависит от пространства убеждений прошлого поколения.

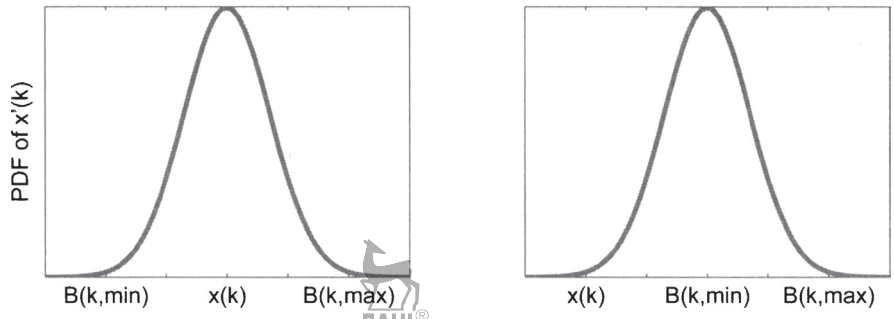


Рис. 15.2. Мутация в культурно-эволюционной программе. На рисунке слева $x(k)$ находится внутри пространства убеждений, поэтому функция плотности вероятности (PDF) ее мутированной версии имеет среднее значение $x(k)$. На рисунке справа $x(k)$ находится вне пространства убеждений, поэтому функция плотности вероятности ее мутированной версии имеет среднее значение, равное ближайшему краю пространства убеждений

■ Пример 15.2

Этот пример показывает, как встраивание культуры может улучшить результативность алгоритма EP. Мы используем $N = 50$, $\beta = 0$ и $\gamma = 1$ в базовом алгоритме EP рис. 5.1. Мы используем алгоритм EP для минимизации 20-мерной функции Экли, описанной в приложении C.1.2, при этом каждая размерность каждой особи случайно инициализируется в области [–30, +30]. Для алгоритма CAEP мы используем $M = 5$ в уравнении (15.4) и $\alpha = 0.5$ в уравнении (15.5). На рис. 15.3 показаны результаты оптимизации, выполненные стандартным и культурным алгоритмом EP, усредненно по 20 симуляциям Монте-Карло. Мы видим,

что алгоритм CAEP намного превосходит стандартный алгоритм EP. На рис. 15.4 показано, как пространство убеждений первой размерности изменяется из поколения в поколение. Мы видим, что пространство убеждений довольно быстро сходится к небольшой области, которая включает оптимальное решение, равное 0. Нет никакой гарантии, что пространство убеждений будет включать оптимальное решение. На самом деле рис. 15.4 показывает, что нижняя граница пространства убеждений иногда немного превышает 0. Но в целом пространство убеждений дает хорошее представление о том, где хорошие кандидатные решения, вероятно, будут находиться в поисковом пространстве. Меньшее значение α в уравнении (15.5) приведет к более быстрому схождению, а большее значение α – к более медленному.

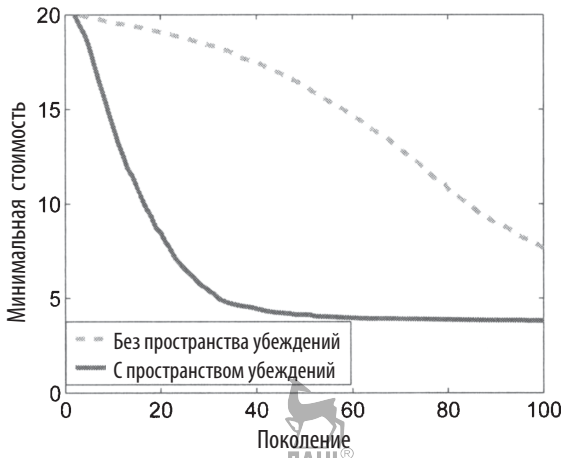


Рис. 15.3. Пример 15.2: эволюционное программирование без пространства убеждений и с пространством убеждений. На рисунке показана лучшая стоимость популяции в каждом поколении, усредненно по 20 симуляциям Монте-Карло. Алгоритм CAEP далеко превосходит стандартный алгоритм EP

■

15.4. Адаптивно-культурная модель

В этом разделе рассматривается культурный алгоритм, который является альтернативным подходу на основе пространства убеждений предыдущих разделов. Алгоритм, который мы обсуждаем в этом разделе, называется адаптивно-культурной моделью (adaptive culture model, ACM) [Axelrod, 1997], [Kennedy, 1998], [Kennedy и Eberhart, 2001, глава 6]. Адаптивно-культурная модель основывается на том, как ин-

дивидуумы в человеческих обществах взаимодействуют друг с другом. Например:

- индивидуумы больше подвержены влиянию тех, кто рядом с ними, географически либо родственно, чем те, кто далек от них [Latané и соавт., 1994]. Это напоминает окрестности роя частиц в главе 11;
- индивидуумы больше подвержены влиянию тех, кто им подобен, чем те, кто от них отличается [Axelrod, 1997], [Kennedy, 1998];
- уравновешивая предыдущий пункт, индивидуумы больше подвержены влиянию тех, кто успешен, чем тех, кто таковым не является [Noel и Jannett, 2005]. Связанный с этим вопрос заключается в том, что индивидуумы больше подвержены влиянию тех, кто подобен их *идеальному* «я», чем тех, кто подобен их *фактическому* «я» [Wetzel и Insko, 1982], [Kennedy, 1998].

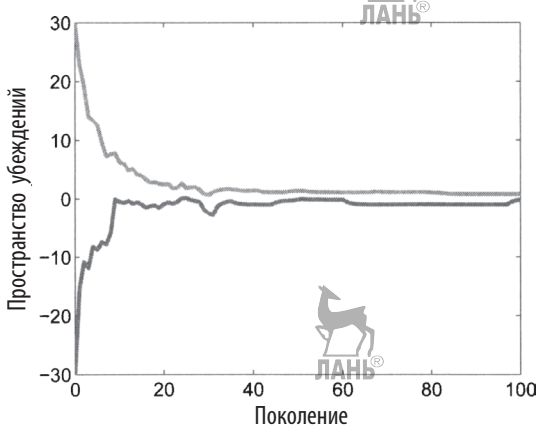


Рис. 15.4. Пример 15.2: пространство убеждений первой размерности алгоритма CAEP для 20-мерной функции Экли. Пространство убеждений быстро сходится к малому участку около 0, который является оптимальным решением

Адаптивно-культурная модель может быть просимулирована, если расположить кандидатные решения некой оптимизационной задачи в виде решетки. Кандидатные решения рассматриваются как отдельные особи в популяции. Близость двух особей можно измерить, по крайней мере, несколькими разными способами. Во-первых, особи имеют географическую близость друг к другу, так как они расположены в решетке. Во-вторых, особи имеют поведенческую близость друг к другу в зависимости от того, насколько они похожи по отношению к их признакам решения.

На рис. 15.5 показан пример решетки эволюционно-алгоритмических особей, в которой каждая особь закодирована восемисимвольной строкой. По мере эволюции популяции особи сохраняют ту же позицию в решетке, но их представления меняются из поколения в поколение. Особи, которые находятся друг к другу ближе, географически или поведенчески, более склонны обмениваться информацией друг с другом и, следовательно, с большей вероятностью станут еще более похожими друг на друга с поведенческой точки зрения. Кроме того, когда особи обмениваются информацией друг с другом, более приспособленная особь с большей вероятностью будет делиться информацией с менее приспособленной особью, а не наоборот.

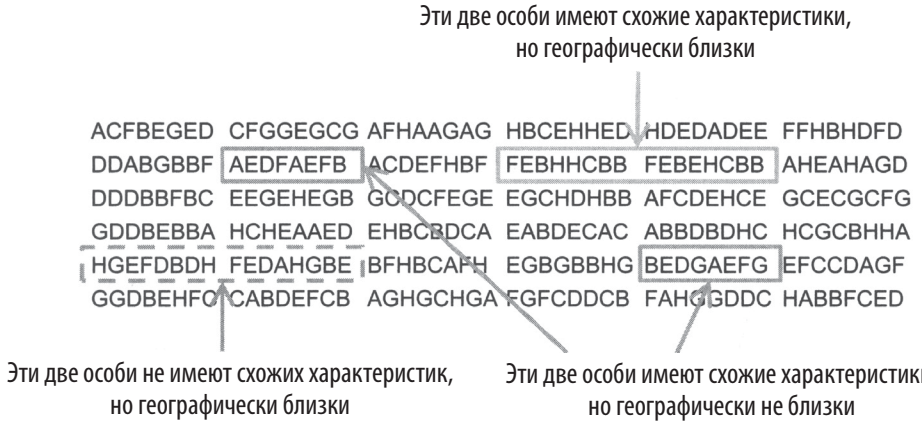


Рис. 15.5. Пример решетки индивидуумов алгоритма ACM. Некоторые особи похожи друг на друга, но географически не близки; они вряд ли будут обмениваться информацией друг с другом. Другие особи, как те две, которые отмечены в левой нижней части решетки, географически близки, но не похожи друг на друга; они также вряд ли смогут обмениваться информацией друг с другом. Вместе с тем некоторые особи, как те, которые отмечены в правой верхней части решетки, географически близки и похожи друг на друга. Они могут обмениваться информацией друг с другом

Мы можем применить все эти идеи для получения алгоритма адаптивно-культурной модели (ACM). На рис. 15.6 приведено схематичное представление базового алгоритма ACM. Популяция инициализируется, и каждому кандидатному решению назначается определенное место в решетке. На каждой итерации алгоритма ACM мы случайно отбираем особь и одного из ее соседей. Мы случайно принимаем решение, делиться или нет информацией между этими двумя соседями, используя

более высокую вероятность, если соседи более похожи. Если мы решим поделиться информацией, то случайно заменяем один из признаков решения у менее приспособленной особи на признак решения у более приспособленной особи.

```
Инициализировать N особей {x_i}, i ∈ [1, N].
Назначить каждой особи случайную позицию в решетке.
While not (критерий останова)
  Случайно отобрать особь x_i, где i ∈ [1, N].
  Случайно отобрать соседа x_k из x_i.
  Рассчитать поведенческое сходство b_{i,k} ∈ [0, 1] между x_i и x_k
  r ← U[0, 1]
  If r < b_{i,k}
    Случайно отобрать индекс признака решения s ∈ [1, n].
    Комментарий: начать делиться информацией
    If x_i более приспособлена, чем x_k then
        x_k(s) ← x_i(s)
    else
        x_i(s) ← x_k(s)
    End if
    Комментарий: прекратить делиться информацией
  End if
Next итерация
```

Рис. 15.6. Схематичное представление базового алгоритма адаптивно-культурной модели (ACM). N – это размер популяции, n – размерность задачи, а $U[0,1]$ – случайное число, равномерно распределенное между 0 и 1

По рис. 15.6 видно, что мы всегда передаем информацию от более приспособленной особи менее приспособленной. В соответствии с духом стохастичности вместо этого мы можем принять вероятностное решение о том, кто с кем делится информацией. Мы принимаем регулировочный параметр $p_1 \in [0.5, 1]$ равным вероятности передачи от лучшей особи к худшей. Мы называем p_1 селекционным давлением и всегда хотим, чтобы $p_1 \geq 0.5$, потому что не имеет интуитивного смысла смещать направление обмена информацией от худшего к лучшему. Предположим, что x_1 и x_2 являются двумя соседними кандидатными особями. Тогда мы заменим псевдокод между «Комментарий: начать делиться информацией» и «Комментарий: прекратить делиться информацией» на рис. 15.6 на логику обмена информацией, показанную на рис. 15.7.

```
ρ ← p1[0, 1]
If ρ < p1 then
  If xi более приспособлена, чем xk then
      xk(s) ← xi(s)
    else
      xi(s) ← xk(s)
  End if
else
  If xi более приспособлена, чем xk then
      xi(s) ← xk(s)
  else
      xk(s) ← xi(s)
  End if
End if
```

Рис. 15.7. Адаптивно-культурная модель со стохастическим обменом информацией. $p_1 \in [0.5, 1]$ – это вероятность обмена информацией от лучшей особи к худшей особи. Приведенный выше фрагмент псевдокода заменяет псевдокод между «Комментарий: начать делиться информацией» и «Комментарий: прекратить делиться информацией» на рис. 15.6. Данный фрагмент кода приводит к p_1 вероятности обмена от лучшей особи к худшей. Если $p_1 = 1$, то этот псевдокод сводится к рис. 15.6

■ Пример 15.3

В данном примере, который мотивирован публикацией [Kennedy и Eberhart, 2001, глава 6], мы решаем задачу коммивояжера с помощью алгоритма ACM. Предположим, что мы хотим проехать восемь мест в таком порядке, который минимизирует общее расстояние пути. Мы предполагаем, что эти места расположены по кругу, как показано на рис. 15.8, и что мы начинаем в точке A. Мы можем легко увидеть, что есть два варианта решения этой задачи: A-B-C-D-E-F-G-H и A-H-G-F-E-D-C-B. Мы случайно инициализируем решетку кандидатных решений размера 18×8. Мы считаем, что решетка тороидальна, благодаря чему особи в крайнем правом углу решетки являются соседями тех, кто находится в крайнем левом углу, а особи в нижней части решетки являются соседями тех, кто наверху. Мы используем логику алгоритма ACM на рис. 15.6 и 15.7 для координации обмена информацией между кандидатными решениями. Реализуем инструкцию «Случайно отобрать соседа x_k из x_i» на рис. 15.6 путем случайного отбора одного из четырех ближайших особей к x_i – то есть особей сразу справа, слева, выше или ниже x_i. Мы используем значение $p_1 = 0.9$ на рис. 15.7.

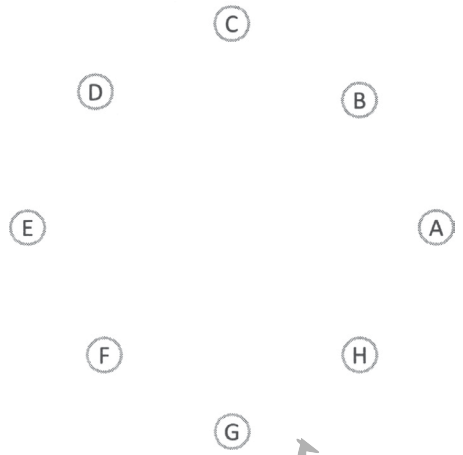


Рис. 15.8. Места в задаче коммивояжера примера 15.3. Цель состоит в том, чтобы посетить все восемь мест, минимизируя общее расстояние пути. Если мы начнем с места A, то существует два оптимальных решения: A-B-C-D-E-F-G-H и A-H-G-F-E-D-C-B

На рис. 15.9 показано схождение типичной симуляции алгоритма ACM. Мы находим первое оптимальное решение примерно после 2500 итераций внешнего цикла на рис. 15.6, и средняя стоимость популяции неуклонно уменьшается вместе с числом взаимодействий.

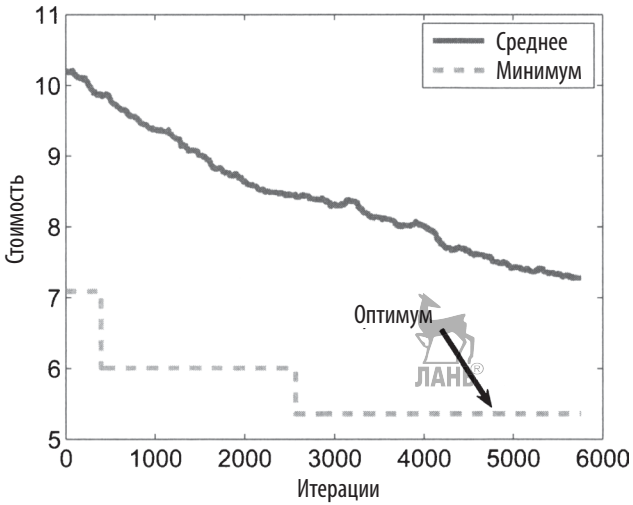


Рис. 15.9. Схождение алгоритма ACM для задачи коммивояжера примера 15.3. Если восемь городов рис. 15.8 расположены в единичной окружности, то глобально минимальная стоимость равна 5.3576

По мере того как особи в популяции продолжают взаимодействовать друг с другом, хорошие особи распространяются, а слабые особи постепенно выпадают из популяции. На рис. 15.10 показано типичное поступательное развитие распространения хороших особей. Первая оптимальная особь найдена примерно после 2500 взаимодействий, после чего превалирование оптимальных решений возрастает примерно линейно.

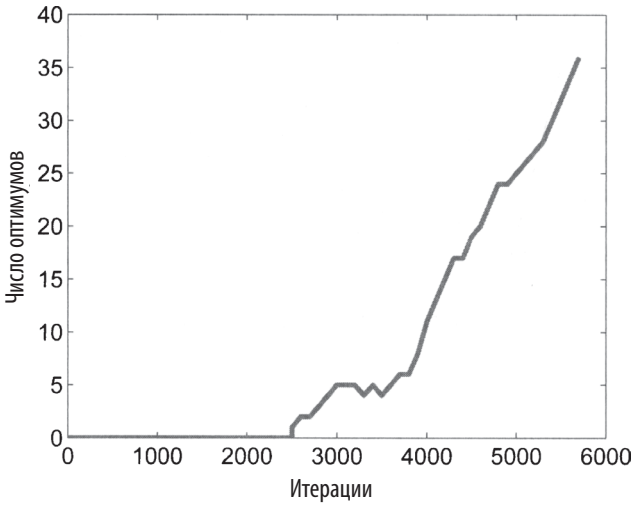


Рис. 15.10. Пример 15.3: распространение оптимальных решений в популяционной решетке задачи коммивояжера. Для появления первого оптимального решения в решетке требуется около 2500 взаимодействий, после чего превалирование оптимальных решений возрастает примерно линейно

На рис. 15.11 показана популяционная решетка размера 18×8 после 5760 взаимодействий, что в среднем составляет 80 взаимодействий на особь. Мы видим, что 31 оптимальное решение A-B-C-D-E-F-G-H сгруппировано в кластер по левому и правому краям решетки (напомним, что данная решетка является тороидом, поэтому правый и левый края смежны). Мы также видим, что существует меньший кластер из пяти оптимальных решений A-H-G-F-E-D-C-B в нижней части решетки. При внимательном рассмотрении рис. 15.11 видно, что существуют и другие кластеры, которые являются неоптимальными решениями задачи коммивояжера. Это поведение отражает то, как информация и поведение распространяются по культуре, как схожие люди склонны группироваться вместе и как мы можем симулировать такое культурное поведение для решения оптимизационных задач.

Рис. 15.11. Популяционная решетка примера 15.3 после 5760 взаимодействий. В решетке выделены два оптимальных кластера (один с 31 особью и один с пятью особями). В модели ACM схожие особи группируются вместе и высоко приспособленные решения, как правило, распространяются по всей популяции

■

Пример 15.3 показывает, как алгоритм ACM может находить несколько решений комбинаторно-оптимизационной задачи. Данный пример может быть расширен для решения непрерывных оптимизационных задач. Мы можем сделать различные обобщения алгоритмов ACM на рис. 15.6 и 15.7.

1. Мы позволили особи быть подверженной влиянию только одного из четырех ближайших соседей. Мы могли бы позволить особям также быть подверженными влиянию более отдаленных соседей. Вероятность или величина взаимодействия может быть убывающей функцией от расстояния.
2. Мы обычно делимся информацией от более приспособленных особей к менее приспособленным особям. Это согласуется с нашим желанием распространять признаки полезного кандидатного решения. Однако в обществе мы видим, что неуспешные люди могут также оказывать влияние на других. Мы склонны избегать

поведения, которое наблюдаем у неудачников. Мы используем эту идею в оптимизации на основе роя частиц с отрицательным подкреплением в разделе 11.6, но она, возможно, еще не была явно использована в культурном алгоритме и поэтому является открытой областью для будущих исследований.

3. На рис. 15.6 мы случайно отбираем особь x_i для взаимодействия. Вместе с тем, возможно, будет иметь больше смысла отбирать особей с низкой приспособленностью, поскольку они больше нуждаются в улучшении. Эта идея напоминает вероятности иммиграции в биогеографической оптимизации (BBO) (см. главу 14).
4. На рис. 15.6 мы случайно отбираем соседа x_k для взаимодействия с x_i. Вместе с тем, возможно, будет иметь больше смысла случайно отбирать группу соседей и принимать решение о выборе стратегии обмена информацией на основе относительных значений приспособленности. Эта идея напоминает вероятности эмиграции в биогеографической оптимизации (BBO) (см. главу 14). Это и предыдущее обобщения намекают на интересные возможности для гибридного культурного алгоритма BBO.
5. Мы могли бы объединить идею пространства убеждений с алгоритмом адаптивно-культурной модели (ACM). Люди в человеческом обществе находятся под влиянием сочетания, состоящего из их соседей и своей культуры. Эта идея называется обобщенной другой моделью (generalized other model, GOM), которая может быть слабо аналогизирована с влиянием средств массовой информации [Shibani et al, 2001]. На рис. 15.6 мы случайно отбираем одного из четырех соседей для обмена информацией с x_i. В алгоритме GOM мы создаем обобщенного соседа, который представляет консенсус всей популяции. Обобщенный сосед – это сосед, который фактически не существует в популяции и является псевдоособью, который формируется путем взятия среднего значения всей популяции. Затем особь x_i может получать информацию от одного из своих четырех соседей или от обобщенного соседа. Эта идея напоминает полноинформированный рой частиц, который предполагает глобальный обмен информацией (см. раздел 11.5). Обобщенный сосед также может быть получен как средневзвешенное приспособленности популяции, хотя это расширение, по-видимому, еще не достаточно изучено.

15.5. Заключение

Культурные алгоритмы – увлекательная область эволюционных вычислений. Они отличаются от типичных эволюционных алгоритмов тем, что не обусловлены непосредственно биологией, а обусловлены социальными науками. Эта обусловленность открывает огромные возможности для социологических исследований, которые могут быть применены к самоорганизующимся вычислительным системам и алгоритмам оптимизации. Поскольку культурные алгоритмы стали впервые изучаться в 1980-х годах, по всей видимости, большинство исследований в этой области было сосредоточено на простых приложениях или модификациях базовых идей культурного алгоритма. Вместе с тем существует большой объем исследований в социальных науках по многим аспектам культуры (включая музыку, экономику, язык), невербальной коммуникации, технологии, семейных отношений, развлечений, образования, спорта, медицины, религии, искусства, литературы, политики, войны и т. д. Любой инженер или исследователь в области информатики, который заинтересован в одном из этих аспектов культуры, имеет практически безграничный пласт идей для применения в культурно-алгоритмических исследованиях. Некоторые другие интересные и потенциально важные области для будущих исследований включают следующее:

- математическое моделирование культурных алгоритмов представляется зрелой областью для будущей работы. Мы видим, что математическое моделирование успешно работает в специализированной литературе для других эволюционных алгоритмов (см. главу 4 и раздел 7.6), но, похоже, не хватает результатов математического моделирования для культурных алгоритмов;
- культуры имеют несколько множеств убеждений, некоторые из которых принадлежат большинству людей, а другие принадлежат меньшинству [Latané et al, 1994]. Иногда эти пространства убеждений конфликтуют в так называемых культурных войнах [Thomson, 2010]. Мы видим это явление в таких спорных областях, как религия, спорт, политика;
- общества включают в себя несколько культур. Эти культуры внутри культур называются субкультурами. Особи внутри субкультур тесно взаимодействуют между собой, а особи из отдельных субкультур взаимодействуют более расплывчато. В одной субкультуре больше выделяются одни системы ценностей, в других субкультурах – другие. Эта идея имеет приложения к многокри-

териальной оптимизации [Coello Coello и Becerra, 2003], [Alami и соавт., 2007].

Как смоделировать эти факторы в культурном алгоритме? Как эти факторы взаимодействуют друг с другом? Какие еще аспекты культуры важны для человеческого самообучения? Как культурные влияния варьируются между людьми? Все эти вопросы остаются открытыми. Дополнительные обзоры и материалы для чтения в области культурных алгоритмов можно найти в публикациях [Reynolds, 1994], [Reynolds и Chung, 1997], [Reynolds, 1999] и [Reynolds и соавт., 2011].

Задачи

Письменные упражнения

15.1. Предложите пару других методов предсказания следующего числа в последовательности уравнения (15.1). Какие значения ваши методы предсказывают?

15.2. Рассмотрие уравнение (15.3) и рис. 15.2.

a) Если $x_i(k)$ находится внутри пространства убеждений, то какова вероятность того, что $x'_i(k)$ будет внутри пространства убеждений?

b) Если $x_i(k)$ находится вне пространства убеждений, то какова вероятность того, что $x'_i(k)$ будет внутри пространства убеждений?

15.3. Рассмотрите уравнение (15.3) и рис. 15.2.

a) Какой будет более агрессивная стратегия использования пространства убеждений при $x_i(k) \in B$? Здесь мы используем термин «более агрессивный», для того чтобы указать на более высокую вероятность того, что $x'_i(k) \in B$.

b) Какой будет более агрессивная стратегия использования пространства убеждений при $x'_i(k) \notin B$?

15.4. Сколько оцениваний функции приспособленности требуется для каждого поколения в алгоритме адаптивно-культурной модели (ACM) на рис. 15.6? Что это значит для справедливого сравнения с другими эволюционными алгоритмами?

15.5. Алгоритм ACM рис. 15.6 делится только одним признаком решения в расчете на взаимодействие. Что это означает для его ре-

зультативности на неразделимых задачах? (Вспомните аналогичное обсуждение в начале раздела 14.5.) Как изменить алгоритм ACM так, чтобы повысить его результативность на неразделимых задачах?

15.6. На рис. 15.9 показано, что алгоритм ACM находит оптимальное решение задачи коммивояжера для 8 городов примерно за 2500 итераций. Проанализируйте качество этой работы.

Компьютерные упражнения

15.7. Повторите пример 15.2 при α = 0, 0.25, 0.5, 0.75 и 1. Постройте график результатов аналогично рис. 15.3 для каждого значения α. Прокомментируйте свои результаты.

15.8. Повторите пример 15.2 при M = 0, 2, 5, 10 и 25. Постройте график результатов аналогично рис. 15.3 для каждого значения M. Прокомментируйте свои результаты.

15.9. Повторите пример 15.3 с селекционным давлением p_1 = 0.5, 0.7, 0.9 и 1. Ограничьте число взаимодействий величиной 2000 для каждой симуляции. Постройте график результатов аналогично рис. 15.9 для каждого значения p_1; при этом, в отличие от рис. 15.9, где показаны результаты типичной симуляции алгоритма ACM, вам необходимо выполнить 20 симуляций Монте-Карло для каждого значения p_1, записать лучшую стоимость в каждом поколении для каждого значения p_1 и построить график среднего значения лучших стоимостей как функции от номера поколения для каждого значения p_1. Прокомментируйте свои результаты.

15.10. Повторите пример 15.3 с размерами окрестности 4 и 8. Ограничьте число взаимодействий величиной 2000 для каждой симуляции. Постройте график результатов аналогично рис. 15.9 для каждого значения размера окрестности; при этом, в отличие от рис. 15.9, где показаны результаты типичной симуляции алгоритма ACM, вам необходимо выполнить 20 симуляций Монте-Карло для каждого размера окрестности, записать лучшие стоимости в каждом поколении для каждого размера окрестности и построить график среднего значения лучших стоимостей как функции номера поколения для каждого размера окрестности. Прокомментируйте свои результаты.

Глава 16

Оппозиционное самообучение

Социальные революции ... чрезвычайно быстрые изменения в человеческом обществе. Они приводят к установлению, проще говоря, противоположных обстоятельств.

– Хамид Тижуш [Tizhoosh, 2005]

Эволюция – это медленный процесс, перемены требуют времени. Однако некоторые изменения происходят быстро. Одним из видов быстрых изменений, который используется почти во всех эволюционных алгоритмах (EA), является мутация. Но есть и такой тип быстрых изменений, который происходит в человеческом обществе, и этот тип мы еще не исследовали: социальные революции. Социальная революция – это сдвиг парадигмы в противоположную сторону от принятой в настоящее время нормы. Иногда социальные революции имеют важные и долгосрочные последствия, например когда Соединенные Штаты сражались против Англии в революционной войне и переходили от колоний к Штатам. Другие революции менее драматичны, например внедрение синтетических материалов в одежду или внедрение микроволновых печей для приготовления пищи. Однако все революции, по определению, приводят к значительным изменениям в образе жизни.

Оппозиционное самообучение (opposition-based learning, OBL) было введено в качестве попытки увеличить скорость самообучения в эволюционных алгоритмах. Поскольку эволюция – процесс медленный, а революция – быстрый, симуляция революции в эволюционных алгоритмах, возможно, ускорит их схождение. Оппозиционное самообучение было первоначально представлено как улучшение самообучения на основе максимизации подкрепления, генетических алгоритмов и тренировки нейронных сетей [Tizhoosh, 2005]. Оно также было реализовано во многих других оптимизационных алгоритмах, включая биогеогра-

фическую оптимизацию (BBO) [Ergezer и соавт., 2009], оптимизацию на основе роя частиц [Omran, 2008], [Rashid и Baig, 2010], дифференциальную эволюцию [Rahnamayan и соавт., 2008], оптимизацию на основе муравьиной кучи [Malisia, 2008] и симулированное закаливание [Ventresca и Tizhoosh 2007].

Краткий обзор главы

В разделе 16.1 представлено несколько определений понятия «оппозиция», так как этот термин относится к численным задачам. В разделе 16.2 описывается, как оппозиционное самообучение может быть встроено в эволюционный алгоритм и, в частности, как оно может быть использовано для улучшения результативности алгоритма биогеографической оптимизации (BBO). В разделе 16.3 математически изучается вероятность улучшения эволюционного алгоритма с использованием разных типов оппозиции. В разделе 16.4 вводится понятие коэффициента перескока, которое используется в оппозиционном самообучении. Хотя оппозиционное самообучение было первоначально определено для задач с непрерывной областью, в разделе 16.5 обсуждается, как оно может быть распространено на комбинаторные задачи и, в частности, на задачу коммивояжера. В разделе 16.6 рассматриваются некоторые идеи дуального самообучения, которое предшествовало оппозиционному самообучению, и показана связь между этими двумя методами.

16.1. Определения и идеи оппозиции

В данном разделе рассматриваются определения и идеи, относящиеся к противоположности скаляра или вектора. Начнем с рассмотрения скаляров и, в частности, примем, что x определен в области $[a, b]$ и центром области является c:

$$\begin{aligned} x &\in [a, b] \quad \text{где } a < b \\ c &= (a + b)/2 \end{aligned} \tag{16.1}$$

16.1.1. *Отраженные противоположности и противоположности по модулю*

Мы можем придумать несколько разных способов определить противоположность скаляра x [Tizhoosh и соавт., 2008]. Например, отраженная противоположность x определяется как

$$x_{o1} = a + b - x. \tag{16.2}$$

Это уравнение означает, что x_{o1} находится на том же расстоянии, что и x, от центра области:

$$c - x = x_{o1} - c. \tag{16.3}$$

Противоположность по модулю от x определяется как

$$x_{o2} = (x - a + c) \bmod (b - a). \tag{16.4}$$

При этом область $[a, b]$ рассматривается как окружность, а противоположность от x определяется как число, лежащее на противоположной стороне окружности. Рисунок 16.1 иллюстрирует отраженную противоположность и противоположность по модулю.

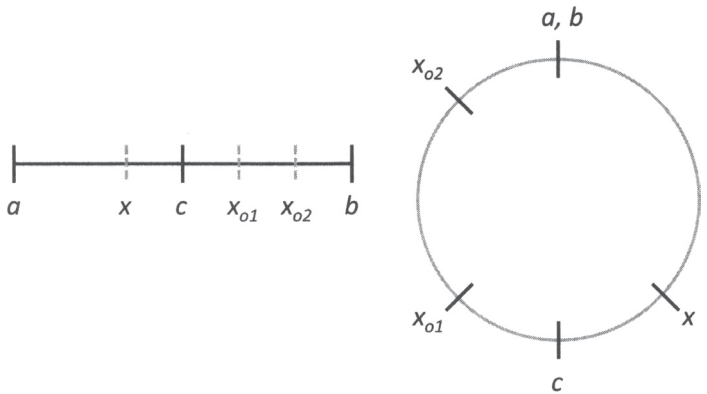


Рис. 16.1. Иллюстрация отраженной противоположности x_{o1} и противоположности по модулю x_{o2} от скаляра x. На рисунке слева показана область x как линейный сегмент, а на рисунке справа – как окружность. Скаляр x определен в области $[a, b]$, и c является центром этой области. Отраженная противоположность x_{o1} находится на том же расстоянии, что и x от c, а противоположность по модулю, x_{o2}, находится на противоположной стороне окружности, которая определяет область x

Определения отраженной противоположности и противоположности по модулю могут быть расширены до векторов простым, но прямолинейным способом. Предположим, что x является n-мерным вектором, определенным на прямоугольной области, то есть x_i определен на области $[a_i, b_i]$ и центром этой области является c_i:

$$\begin{aligned} x &= [x_1 \ldots x_n] \\ \text{где } x_i &\in [a_i, b_i] \text{ и } a_i < b_i \text{ для } i \in [1, n] \\ c_i &= (a_i + b_i)/2 \text{ для } i \in [1, n] \end{aligned} \tag{16.5}$$

Отраженная противоположность от x определяется как

$$\begin{aligned} x_{o1} &= [x_{o1,1} \ldots x_{o1,n}] \\ \text{где } x_{o1,i} &= a_i + b_i - x_i \text{ для } i \in [1, n] \end{aligned} \tag{16.6}$$

Противоположность по модулю от вектора x определяется как

$$\begin{aligned} x_{o2} &= [x_{o2,1} \ldots x_{o2,n}] \\ \text{где } x_{o2,i} &= (x_i - a_i + c_i) \bmod (b_i - a_i) \end{aligned} \tag{16.7}$$

Эти определения применяются только к прямоугольным областям. Распространение данных определений на непрямоугольные области оставлено для будущей работы, но, вероятно, не является слишком сложным.

В этой главе противоположность по модулю использоваться больше не будет. В оставшейся части главы мы будем использовать термин «противоположность от x» как сокращение термина «отраженная противоположность от x» и использовать обозначение x_o как сокращение для x_{o1}.

16.1.2. *Частичные противоположности*

При наличии вектора x мы можем определить x_p, частичную противоположность от x, взяв противоположность нескольких размерностей x, оставив другие элементы x без изменений. Например:

$$\begin{aligned} x &= [x_1 \ldots x_n] \\ \text{Частичная противоположность } x_p &= [x_{p1} \ldots x_{pn}] \\ \text{где } x_{pi} &\leftarrow \begin{cases} x_{oi} & \text{для } i \in S \\ x_i & \text{для } i \in \overline{S} \end{cases} \end{aligned} \tag{16.8}$$

где S – это некое подмножество $\{1, 2, \ldots, n\}$ и $\overline{S}$ – его дополнение, то есть $S \cup \overline{S} = \{1, 2, \ldots, n\}$, $S_j \notin \overline{S}$ для всех $j \in \{1, \ldots, |S|\}$ и $\overline{S}_j \notin S$ для всех $j \in \{1, \ldots, |\overline{S}|\}$. *Степень оппозиции* x_p определяется как

$$\tau(x_p) = |S|/n. \tag{16.9}$$

■ Пример 16.1

Предположим, что $x = [0.5\ 0.5]$, где оба элемента x определены на области $[0, 2]$. Мы можем определить четыре частичные противоположности от x:

$$
\begin{aligned}
x_p^{(1)} &= [\,0.5\;\;0.5\,] \rightarrow \tau\left(x_p^{(1)}\right) = 0 \\
x_p^{(2)} &= [\,1.5\;\;0.5\,] \rightarrow \tau\left(x_p^{(1)}\right) = 1/2 \\
x_p^{(3)} &= [\,0.5\;\;1.5\,] \rightarrow \tau\left(x_p^{(1)}\right) = 1/2 \\
x_p^{(4)} &= [\,1.5\;\;1.5\,] \rightarrow \tau\left(x_p^{(1)}\right) = 1
\end{aligned} \tag{16.10}
$$

Рисунок 16.2 иллюстрирует частичные противоположности от x.

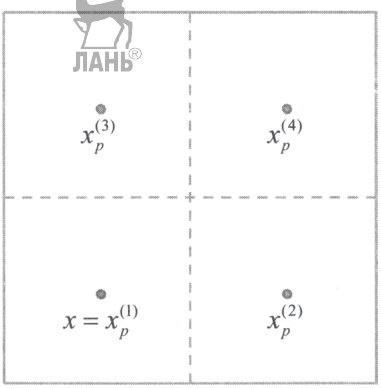


Рис. 16.2. Пример 16.1: степень оппозиции частичных противоположностей двумерного вектора x. Вектор $x_p^{(1)}$ идентичен x, поэтому его степень оппозиции равна 0. Векторы $x_p^{(2)}$ и $x_p^{(3)}$ включают в себя один элемент, который противоположен соответствующему элементу x, и один элемент, который идентичен соответствующему элементу x, поэтому их степень оппозиции равна 0.5. Каждый элемент $x_p^{(4)}$ противоположен соответствующему элементу x, поэтому его степень оппозиции равна 1

■

16.1.3. *Противоположности 1-го рода и противоположности 2-го рода*

До этого момента мы определяли противоположность в терминах области функции. Такое определение называется противоположностью 1-го рода. Мы также можем определить противоположности в терминах диапазона функции, и такое определение называется противоположностью 2-го рода [Tizhoosh и соавт., 2008]. Мы начнем со скалярной функции y(·) скаляра x, где x определен на области $[a, b]$. Диапазон $[y_{\min}, y_{\max}]$ определяется как

$$
\begin{aligned}
y_{\min} &= \min y(x) : x \in [a, b] \\
y_{\max} &= \max y(x) : x \in [a, b]
\end{aligned} \tag{16.11}
$$

Центр диапазона определяется как

$$y_c = (y_{\max} - y_{\min})/2. \tag{16.12}$$

Отраженная противоположность 2-го рода от x определяется как

$$x_o^{(r)} = x' : y(x') = y_{\min} + y_{\max} - y(x). \tag{16.13}$$

Из этого следует, что $y\left(x_o^{(r)}\right)$ – это то же самое, что и расстояние от $y(x)$ до y_c:

$$y_c - y\left(x_o^{(r)}\right) = y_c - y(x). \tag{16.14}$$

Это определение может привести к многочисленным значениям для $x_o^{(r)}$, если $y(\cdot)$ не является взаимно однозначным отображением. Рисунок 16.3 иллюстрирует разницу между противоположностями 1-го и 2-го рода. Обратите внимание, что мы можем расширить определение противоположности 2-го рода, получив противоположность 2-го рода от вектора, противоположность 2-го рода по модулю от вектора и степень оппозиции 2-го рода.

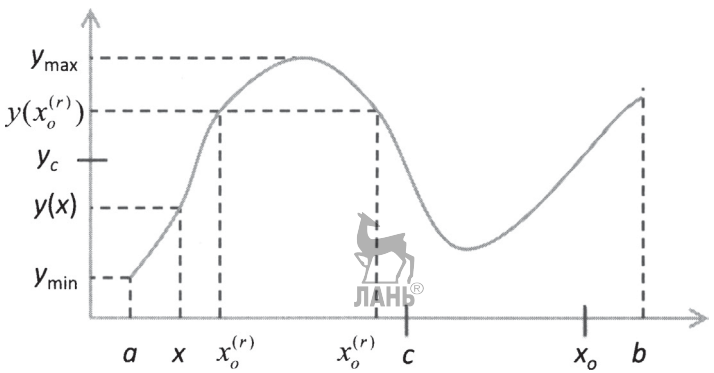


Рис. 16.3. Рассмотрим скаляр x в области $[a, b]$ и функцию $y(x)$. Противоположность 1-го рода от x равняется x_o и получается путем отражения x через центр области c. Противоположность 2-го рода от x получается путем отражения $y(x)$ через центр области y_c, получая $y(x_o^{(r)})$, а затем вычисления инверсии для $y(x_o^{(r)})$, получая $x_o^{(r)}$. В результате для этого примера мы получим два возможных значения $x_o^{(r)}$

В оставшейся части этой главы мы ограничимся обсуждением противоположности 1-го рода. Противоположность 2-го рода заслуживает дальнейшего изучения в контексте эволюционных алгоритмов, но мы оставляем эту тему для дальнейших исследований.

16.1.4. *Квазипротивоположности и сверхпротивоположности*

Теперь мы определим три дополнительных подхода к противоположности. Как и прежде, мы рассмотрим скаляр $x \in [a, b]$ с c в качестве центра его области.

Квазипротивоположность x определяется следующим образом [Tizhoosh et al, 2008]:

$$x_{qo} = \text{rand}(c, x_o), \tag{16.15}$$

где x_o – это стандартная, отраженная противоположность, определенная в уравнении (16.2). То есть x_{qo} является реализацией случайного числа, равномерно распределенного на $[c, x_o]$. Обратите внимание, что мы определяем функцию *rand* таким образом, что ее результат не зависит от порядка ее аргументов; то есть обозначения $\text{rand}(c, x_o)$ и $\text{rand}(x_0, c)$ эквивалентны.

Сверхпротивоположность x определяется следующим образом [Tizhoosh и соавт., 2008]:

$$x_{so} = \begin{cases} \text{rand}(x_o, b) & \text{если } x < c \\ \text{rand}(a, x_o) & \text{если } x > c \end{cases} \tag{16.16}$$

То есть x_{so} является реализацией случайного числа, равномерно распределенного между x_o и границей области, наиболее удаленной от x. Это определение не является полным, поскольку оно не определяет x_{so} для случая $x = c$, но эта особая ситуация может быть решена путем произвольного изменения одного из неравенств в уравнении (16.16), чтобы оно включало как равенство, так и неравенство.

Квазиотраженная противоположность x определяется следующим образом [Ergezer и соавт., 2009]:

$$x_{qr} = \text{rand}(x, c). \tag{16.17}$$

То есть x_{qr} является реализацией случайного числа, равномерно распределенного между x и c. Следует отметить, что использование слова «отраженный» в термине «квазиотраженный» не связано со словом «отраженный» в термине «отраженная противоположность» (см. уравнение (16.2)).

На рис. 16.4 показано четыре разных метода противопоставления. Мы можем распространить эти определения на векторы, противоположности по модулю и противоположности 2-го рода, следуя процедурам, представленным ранее в этом разделе.

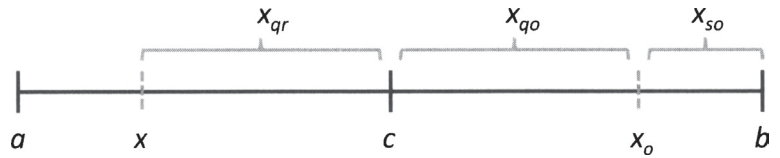


Рис. 16.4. Предположим, что мы имеем скаляр $x \in [a, b]$. Противоположность от x равна x_o и получается путем отражения x через центр области c. Квазипротивоположность от x равна x_{qo} и получается путем генерации случайного числа между c и x_o. Сверхпротивоположность от x равна x_{so} и получается путем генерации случайных чисел между x_o и границы области, наиболее удаленной от x. Квазиотраженная противоположность от x равна x_{qr} и получается путем генерации случайных чисел между x и c

Интересно установить связи между этими определениями противоположности и нечеткой логикой. На рис. 16.4 показано, что противоположность от x_o является четкой; при наличии x его противоположность x_o является четким числом или вектором. Вместе с тем x_{qr}, x_{qo} и x_{so} могут быть определены как нечеткие величины. Связи между этими определениями противоположности и нечеткой логикой в литературе не представлены, и исследование таких связей представляется назревшей областью для дальнейших исследований.

16.2. Алгоритмы оппозиционной эволюции

В данном разделе представлен типичный алгоритм оппозиционного самообучения (OBL) и показано, как он может расширить эволюционный алгоритм. Одним из простых подходов к использованию оппозиционного самообучения с любым эволюционным алгоритмом является выполнение следующих шагов.

1. При инициализации N особей эволюционно-алгоритмической популяции создается N противоположных особей, при этом каждая противоположная особь соответствует одной из N исходных особей. Учитывая наши $2N$ кандидатных решений (N оригинальных особей и N противоположных особей), мы оставляем лучших N в качестве исходной популяции оппозиционного эволюционного алгоритма. Эта общая идея обсуждается в разделе 8.1.
2. Мы выполняем стандартную реализацию эволюционного алгоритма. Как мы уже видели ранее в этой книге, она включает в себя цикл оцениваний функции стоимости, рекомбинации и мутации. По определению, цикл выполняется один раз на поколение.

3. Один раз в несколько поколений мы вычисляем противоположность каждой из N особей. Из этих $2N$ кандидатных решений (N стандартных эволюционно-алгоритмических особей и N противоположных) мы оставляем лучшие N для следующего эволюционно-алгоритмического поколения. В каждом поколении мы выполняем этот шаг с вероятностью $J_r \in [0, 1]$, которая называется скоростью перескока (jumping rate).

В нашем оппозиционном эволюционном алгоритме мы должны принять несколько решений.

1. Какой эволюционный алгоритм использовать? Ответ на этот вопрос также подразумевает, что мы должны выбрать все регулировочные параметры выбранного эволюционного алгоритма.
2. Какую противоположность мы должны использовать?
3. Какое значение мы должны использовать для скорости перескока J_r?

Скорость перескока является регулировочным параметром. У нас не так много рекомендаций для J_r, но мы не хотим делать ее слишком высокой. Причина, почему мы периодически создаем противоположную популяцию, заключается в том, чтобы разведать неизведанные участки поискового пространства. Но мы не хотим создавать противоположную популяцию каждое поколение, потому что тогда мы бы просто многократно скакали по поисковому пространству, в итоге впустую тратя вычислительные ресуры на оценивание функций. Результаты оппозиционно-дифференциальной эволюции показывают, что $J_r \approx 0.3$ обеспечивает хороший баланс [Rahnamayan и соавт., 2008].

Обратите внимание, что неоппозиционный эволюционный алгоритм, который выполняется для G поколений с N особями, требует в общей сложности GN оценивания функций. Оппозиционный эволюционный алгоритм, который выполняется для G' поколений с N' особями и скоростью перескока J_r, в общей сложности требует в среднем $G'N'(1 + J_r)$ оцениваний функций. Для того чтобы провести справедливое сравнение между неоппозиционным эволюционным алгоритмом и его оппозиционной версией, нам нужно выбрать G', N' и J_r так, чтобы

$$GN = G'N'(1 + J_r). \tag{16.18}$$

Мы можем сделать это, либо приняв $N' = N$ и сократив оппозиционноый лимит поколений, так чтобы $G' = G/(1 + J_r)$, либо приняв $G' = G$ и

сократив оппозиционный размер популяции, так чтобы $N' = N(1 + J_r)$, либо одновременно сократив G' и N', для того чтобы удовлетворить уравнению (16.18).

Оппозиционно-биогеографическая оптимизация

Теперь мы покажем, как представленное выше схематичное описание оппозиционного самообучения (OBL) может быть использовано в биогеографической оптимизации (BBO). Мы объединим стандартный алгоритм биогеографической оптимизации на рис. 14.3 с оппозиционным самообучением и получим алгоритм оппозиционно-биогеографической оптимизации (oppositional BBO, OBBO) [Ergezer и соавт., 2009]. Рисунок 16.5 схематично показывает алгоритм OBBO. Обратите внимание, что алгоритм рис. 16.5 идентичен алгоритму рис. 14.3, за исключением псевдокода между строками «Комментарий: начать оппозиционную логику» и «Комментарий: закончить оппозиционную логику».

Инициализировать популяции кандидатных решений $\{x_k\}$ для $k \in [1, N]$.
While not (критерий останова)
  For each x_k, назначить вероятность эмиграции $\mu_k \propto$ приспособленность x_k, при $\mu_k \in [0, 1]$
  For each особь x_k, назначить вероятность иммиграции $\lambda_k = 1 - \mu_k$
  $\{z_k\} \leftarrow \{x_k\}$
    For each особь z_k
      For each признак s решения
        Применить λ_k для вероятностного принятия решения об иммиграции в z_k.
        If иммигрировать **then**
          Применить $\{\mu_i\}_{i=1}^{N}$ для вероятностного отбора мигрирующей особи x_j
          $z_k(s) \leftarrow x_j(s)$
        End if
      Next признак решения
      Вероятностно мутировать $\{z_k\}$.
  Next особь
  Комментарий: начать оппозиционную логику
  $r \leftarrow U[0, 1]$
  If $r < J_r$ **then**
    Применить $\{z_k\}$ для создания противоположной популяции $\{z_k\}$
    $\{z_k\} \leftarrow$ лучшие N особей из $\{z_k\} \cup \{z_k\}$
  End if

Комментарий: закончить оппозиционную логику

$\{x_k\} \leftarrow \{z_k\}$

Next поколение

Рис. 16.5. Алгоритм оппозиционно-биогеографической оптимизации (OBBO) с популяцией размером N. $\{x_k\}$ – это вся популяция особей в целом, x_k – k-я особь и $x_k(s)$ – s-й признак x_k. Аналогичным образом, $\{z_k\}$ – это временная популяция особей, z_k – k-я временная особь и $z_k(s)$ – s-й признак z_k

■ Пример 16.2

В данном примере мы оптимизируем 20-мерную функцию Гриванка. Эта функция определена в приложении С.1.6, а также приведена здесь для удобства:

$$f(x) = 1 + \sum_{i=1}^{n} x_i^2/4000 - \prod_{i=1}^{n} \cos\,(x_i/\sqrt{i}), \tag{16.19}$$

где $x_i \in [-600, +600]$. Минимальное значение x равно $x_i = 0$ для всех $i \in [1, n]$. Мы используем алгоритм BBO с популяцией размером $N = 50$ и лимитом оцениваний функции 2500. В результате получим 50 поколений, если оцениваем каждую особь алгоритма BBO один раз за поколение. Мы используем мутационную вероятность 1 % на размерность на особь и параметр элитарности 2. В алгоритм BBO мы также добавляем оппозиционное самообучение (OBL), как показано на рис. 16.5. При реализации оппозиционного самообучения мы используем скорость перескока $J_r = 0.2$, поэтому число поколений уменьшается примерно до 41–42, в зависимости от последовательности случайных чисел, которая управляет поколением противоположной популяции. После 20 имитаций Монте-Карло среднее самых низких стоимостей, найденных алгоритмами биогеографической оптимизации (BBO) и оппозиционно-биогеографической оптимизации (OBBO), выглядит следующим образом:

BBO	:	8.85
Отраженный OBBO	:	9.69
Квази-OBBO	:	0.05
Супер-OBBO	:	11.82
Квазиотраженный OBBO	:	0.03

Смысл терминов в приведенном выше перечне можно увидеть на рис. 16.4. Отраженый OBBO, то есть алгоритм с отраженной противо-

положностью, относится к x_o, квази-OBBO, то есть алгоритм с квазипротивоположностью, относится к x_{qo}, супер-OBBO, то есть алгоритм со сверхпротивоположностью, относится к x_{so}, а квазиотраженный OBBO, то есть алгоритм с квазиотраженной противоположностью, относится к x_{qr}. Мы видим, что отраженный OBBO и супер-OBBO работают хуже, чем алгоритм BBO. Вместе с тем квази-OBBO и квазиотраженный OBBO работают удивительно лучше, чем алгоритм BBO.

■

Как мы знаем из теоремы об отсутсвии бесплатных обедов (см. приложение B), поразительная результативность алгоритмов OBBO с квазипротивоположностью (квази-OBBO) и с квазиотраженной противоположностью (квази-OBBO) в примере 16.2 не является никаким волшебством. Причина их превосходной результативности состоит в том, что решение задачи с функцией Гриванка лежит ровно в центре своей области. Рисунок 16.4 показывает нам, что оба этих алгоритма, как правило, стремятся перемещать особей ближе к центру поисковой области. Алгоритм OBBO с отраженной противоположностью (отраженный OBBO) удерживает особей на одинаковом расстоянии от центра (но на противоположной стороне поисковой области), поэтому данный алгоритм работает хуже, чем алгоритм BBO. Отраженный OBBO ни ухудшает, ни улучшает особь в задаче с функцией Гриванка; он просто потребляет оценивания функции. Алгоритм OBBO со сверхпротивоположностью (сверх-OBBO) работает даже хуже. Рисунок 16.4 показывает нам, что данный алгоритм всегда перемещает особей дальше от центра поисковой области, что снижает результативность в задаче с функцией Гриванка. Поэтому результаты примера 16.2 представляют собой именно то, что мы предсказали бы из нашего понимания оппозиционного самообучения (OBL) и данной задачи.

Разумеется, если бы мы знали, что решение находится недалеко от центра поисковой области, то нам не нужно было бы использовать оппозиционное самообучение; мы могли бы использовать любой другой метод, который смещает особей алгоритма BBO к центру области. В этом смысле использование оппозиционного самообучения для задачи с функцией Гриванка является «обманом»; он опирается на тот факт, что решение данной задачи находится недалеко от центра области. То есть он неявно опирается на специфичную для конкретной задачи информацию. Это тесно связано с теоремой об отсутствии бесплатных обедов, которая обсуждается в приложении B. Задача, решение которой вполне может находиться *где угодно* в поисковой области, могла бы обеспечить

для оппозиционного самообучения более оптимальный тест, и это приводит нас к следующему примеру.

■ Пример 16.3

В данном примере мы снова оптимизируем функцию Гриванка с $n = 20$ размерностями (см. приложение С.1.6). Мы используем те же параметры, что и в примере 16.2. Однако на этот раз мы случайно сдвинем решение задачи Гриванка:

$$f(x) = 1 + \sum_{i=1}^{n}(x_i - r_i)^2/4000 - \prod_{i=1}^{n}\cos\left((x_i - r_i)/\sqrt{i}\right), \qquad (16.20)$$

где r_i – это случайное число, равномерно распределенное в поисковой области [–600, +600]. Минимальный аргумент функции $f(x)$ равен $x_i^* = r_i$ для $i \in [1, n]$. После 20 симуляций Монте-Карло, где мы используем разное множество значений $\{r_i\}$ для каждой выборки Монте-Карло, среднее значение наименьшей стоимости, найденной алгоритмами BBO и OBBO, выглядит следующим образом:

BBO	:	10.4
Отраженный OBBO	:	14.1
Квази-OBBO	:	13.8
Супер-OBBO	:	13.4
Квазиотраженный OBBO	:	13.9

Все оппозиционные алгоритмы BBO работают значительно хуже, чем стандартный алгоритм BBO. Это связано с тем, что сдвинутое решение функции Гриванка равномерно распределено в поисковом пространстве, поэтому противоположная точка не более вероятна, чем особь алгоритма BBO, близкая к оптимальному решению. Фактически по мере увеличения числа поколений алгоритма BBO противоположная точка будет с меньшей вероятностью близка к оптимальному решению. Причина этого в том, что по мере увеличения числа поколений особи алгоритма BBO перемещаются ближе к оптимальному решению в силу их механизма обмена информацией. Поэтому оппозиционная функция, скорее всего, отдалит их от решения. Использование оппозиции в данном случае не только ведет к пустой трате вычислительных ресурсов на оценивание функций, но и, как представляется, приводит к обратным результатам.

■

Пример 16.3, по-видимому, показывает, что после более тщательного рассмотрения первоначально вселившие надежду результаты примера 16.2 оказываются миражом. Однако не все потеряно. Когда мы пытаемся решить реальную оптимизационную задачу с помощью эволюционного алгоритма, нам нужно определить поисковую область. Обычно мы определяем ее так, чтобы быть уверенными в том, что решение находится в поисковой области. Это означает, что мы часто делаем поисковую область больше, чем необходимо. Нам нужна большая поисковая область, потому что мы не уверены, где находится решение. Мы склонны ошибаться больше на стороне более крупной, чем требуется, поисковой области, чем на стороне слишком малой поисковой области. Но мы, возможно, подозреваем, что решение лежит недалеко от центра области. Следовательно, ситуация, более реалистичная, чем в примере 16.2 или примере 16.3, вполне может заключаться в случайном сдвиге решения функции Гриванка таким образом, чтобы он достиг любой крайности области, но при этом будет держаться ближе к центру, и это приводит нас к следующему далее примеру.

■ Пример 16.4

В данном примере мы еще раз оптимизируем функцию Гриванка с $n = 20$ размерностями. Мы используем те же параметры, что и в примерах 16.2 и 16.3. Однако в этот раз мы случайно сдвигаем решение задачи Гриванка таким образом, что ее решением является реализация нормально распределенного вектора, каждый элемент которого имеет стандартное отклонение, равное 200:

$$\begin{aligned} r_i &\leftarrow 200\,N(0,1) \quad \text{для } i \in [1,n] \\ r_i &\leftarrow \max(\min(r_i, 600), -600) \\ f(x) &= 1 + \sum_{i=1}^{n}(x_i - r_i)^2/4000 - \prod_{i=1}^{n}\cos\left((x_i - r_i)/\sqrt{i}\right) \end{aligned} \tag{16.21}$$

$N(0, 1)$ – это нормально распределенное случайное число с нулевым средним и единичной дисперсией, что означает, что $200N(0, 1)$ имеет стандартное отклонение 200. Операция max/min в уравнении (16.21) гарантирует, что каждый элемент решения сдвинутой функции Гриванка остается в поисковой области [–600, 600]. После 20 симуляций Монте-Карло, где мы используем разное множество значений $\{r_i\}$ для каждой выборки Монте-Карло, среднее наименьшей стоимости, найденной алгоритмами BBO и OBBO, выглядит следующим образом:

```
                BBO :  9.5
Отраженный OBBO :  11.2
      Квази-OBBO :  9.9
      Супер-OBBO :  11.9
Квазиотраженный OBBO :  6.0
```

Результативность алгоритмов BBO и квази-OBBO статистически идентична, в то время как результативность алгоритмов отраженного OBBO и супер-OBBO хуже, чем у стандартного алгоритма BBO. Вместе с тем квазиотраженный OBBO работает заметно лучше, чем стандартный алгоритм BBO. Причина в том, что квазиотраженный OBBO стремится передвигать особей алгоритма BBO к центру области. Вполне можно было бы ожидать, что квази-OBBO тоже должен работать лучше, чем алгоритм BBO, потому что квази-OBBO тоже передвигает особей к центру области. Однако данный алгоритм передвигает особей к центру области, а также отдаляет их от текущей особи x (см. рис. 16.4). Это приводит к ухудшению результативности в более поздних поколениях, когда большинство особей имеет низкую стоимость. Квазиотраженный OBBO работает лучше, потому что он не только передвигает особей к центру, но и стремится удерживать особей вблизи их первоначального местоположения в поисковом пространстве, что полезно после первых нескольких поколений.

■

16.3. Оппозиционные вероятности

В разделе 16.2 показано, как оппозиционное самообучение (OBL) может быть встроено в биогеографическую оптимизацию (BBO) для улучшения ее результативности. В данном разделе исследуется вероятность приближения к решению оптимизационной задачи при использовании разных типов противоположности (то есть оппозиции): отраженной противоположности, квазипротивоположности и квазиотраженной противоположности. Этот раздел чрезвычайно математически сложен, поэтому практико-ориентированный читатель может смело пропустить его или просто прочитать результаты в табл. 16.1 в конце раздела.

В этом разделе мы принимаем следующие допущения.

1. Мы будем считать поисковое пространство одномерным. Это, очевидно, очень ограничительно, но это отправная точка, и до-

полнительная работа должна позволить расширить одномерный случай до более высоких размерностей.

2. Мы будем считать, что решение x^* оптимизационной задачи неизвестно, но что оно является реализацией случайной величины, равномерно распределенной в области x. Это допущение основывается на принципе недостаточного основания, который утверждает, что в отсутствие предварительных знаний надо полагать, что все события в поисковом пространстве равновероятны [Dembski и Marks, 2009b], [Dembski и Marks, 2009a].

Предположим, что у нас есть произвольная эволюционно-алгоритмическая особь x. Мы будем считать без потери общности, что x находится в нижней половине поисковой области. Рассмотрим вероятность того, что ее квазипротивоположность x_{qo} ближе, чем ее противоположноть x_o, к оптимальному решению x^*. На рис. 16.6 показаны произвольная эволюционно-алгоритмическая особь x, ее противоположность x_o и квазипротивоположность x_{qo}. Оптимальное решение x^* может быть в одном из трех участков.

1. Определим случай 1 как ситуацию, в которой $x^* \in [a, c]$.
2. Определим случай 2 как ситуацию, в которой $x^* \in [c, x_o]$.
3. Определим случай 3 как ситуацию, в которой $x^* \in [x_o, b]$.

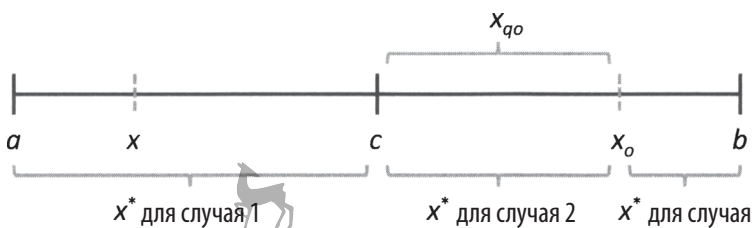


Рис. 16.6. x – это эволюционно-алгоритмическая особь, x_o – ее противоположность, x_{qo} – ее квазипротивоположность (взятые из равномерного распределения между c и x_o). Решение x^* оптимизационной задачи равномерно распределено на $[a, b]$, и поэтому оно может лежать в одном из трех участков, показанных выше

Случай 1

В случае 1 ясно, что x_{qo} ближе, чем x_o к x^*. Следовательно:

$$\Pr\left(|x_{qo} - x^*| < |x_o - x^*|\right) = 1 \quad \text{для случая 1.} \tag{16.22}$$

Случай 2

В случае 2 x^* и x_{qo} независимы и равномерно распределены в $[c, x_o]$. Мы можем применить теорему полной вероятности [Mitzenmacher и Upfal, 2005] и тот факт, что $x_o - x^* > 0$ для случая 2, чтобы записать вероятность того, что x_{qo} ближе, чем x_o к x^*, следующим образом:

$$\begin{aligned}
&\Pr\left(|x_{qo} - x^*| < |x_o - x^*|\right) \\
&= \Pr\left(|x_{qo} - x^*| < x_0 - x^* \mid x_{qo} - x^* < 0\right) \Pr\left(x_{qo} - x^* < 0\right) + \\
&\quad \Pr\left(|x_{qo} - x^*| < x_0 - x^* \mid x_{qo} - x^* > 0\right) \Pr\left(x_{qo} - x^* > 0\right) \\
&= \Pr\left(x_{qo} > 2x^* - x_0 \mid x_{qo} < x^*\right) \Pr\left(x_{qo} < x^*\right) + \\
&\quad \Pr(x_{qo} < x_0 \mid x_{qo} > x^*) \Pr\left(x_{qo} > x^*\right). \qquad (16.23)
\end{aligned}$$

Рассмотрим члены в правой части приведенного выше уравнения. Во-первых, поскольку x_{qo} и x^* равномерно распределены на $[c, x_o]$, мы видим, что

$$\begin{aligned}
\Pr(x_{qo} < x^*) &= 1/2 \\
\Pr(x_{qo} > x^*) &= 1/2 \\
\Pr(x_{qo} < x_0 \mid x_{qo} > x^*) &= 1 \qquad (16.24)
\end{aligned}$$

Мы можем применить теорему Байеса, чтобы написать первое выражение в правой части уравнения (16.23) как

$$\begin{aligned}
&\Pr(x_{qo} > 2x^* - x_0 \mid x_{qo} < x^*) \Pr(x_{qo} < x^*) \\
&= \Pr(x_{qo} > 2x^* - x_0, x_{qo} < x^*) \\
&= \Pr(2x^* - x_o < x_{qo} < x^*) \\
&= \int_c^{x_o} \int_{x_{qo}}^{(x_{qo}+x_0)/2} f(x^*)\, f(x_{qo}) dx^*\, dx_{qo}, \qquad (16.25)
\end{aligned}$$

где мы посчитали, что x^* и x_{qo} независимы от функций плотности вероятности $f(x^*)$ и $f(x_{qo})$. Исходя из равномерных функций плотности вероятности, вышеупомянутое интегрирование может быть выполнено как

$$\begin{aligned}
&\Pr(x_{qo} > 2x^* - x_0 \mid x_{qo} < x^*) \Pr(x_{qo} < x^*) \\
&= \int_c^{x_o} \int_{x_{qo}}^{(x_{qo}+x_0)/2} \frac{1}{(x_o - c)^2}\, dx^*\, dx_{qo} \\
&= \int_c^{x_o} \frac{x_o - x_{qo}}{2(x_o - c)^2}\, dx_{qo} \\
&= 1/4. \qquad (16.26)
\end{aligned}$$

Подстановка уравнений (16.24) и (16.26) в уравнение (16.23) дает

$$\Pr\left(|x_{qo} - x^*| < |x_o - x^*|\right) = 3/4 \quad \text{для случая 2.} \tag{16.27}$$

Случай 3

В случае 3 рис. 16.6 видно, что x_o ближе, чем x_{qo} к x^*. Следовательно,

$$\Pr\left(|x_{qo} - x^*| < |x_o - x^*|\right) = 0 \quad \text{для случая 3.} \tag{16.28}$$

Окончательные результаты

Будем использовать ε как сокращенное обозначение, чтобы указать событие, что x_{qo} ближе, чем x_o, к оптимальному решению x^*:

$$\varepsilon = \left\{|x_{qo} - x^*| < |x_o - x^*|\right\}. \tag{16.29}$$

Тогда мы можем объединить результаты из случаев 1, 2 и 3 и получить

$$\begin{aligned} \Pr(\varepsilon) &= \Pr\left(\varepsilon \mid x^* \in [a, c]\right) \Pr\left(x^* \in [a, c]\right) + \\ &\quad \Pr\left(\varepsilon \mid x^* \in [c, x_o]\right) \Pr\left(x^* \in [c, x_o]\right) + \\ &\quad \Pr\left(\varepsilon \mid x^* \in [x_o, b]\right) \Pr\left(x^* \in [x_o, b]\right) \\ &= (1)\left(\frac{1}{2}\right) + \left(\frac{3}{4}\right)\left(\frac{x_o - c}{b - a}\right) + 0. \end{aligned} \tag{16.30}$$

Если x равномерно распределено в нижней половине поисковой области, то x_o равномерно распределено в верхней половине поисковой области. Следовательно, ожидаемое значение равно x_o:

$$E(x_o) = (c + b)/2. \tag{16.31}$$

Приняв ожидаемое значение уравнения (16.30), получаем:

$$\begin{aligned} E\left[\Pr\left(|x_{qo} - x^*| < |x_o - x^*|\right)\right] &= \frac{1}{2} + \frac{3}{4}\frac{(b - c)/2}{b - a} \\ &= 1/2 + 3/16 = 11/16. \end{aligned} \tag{16.32}$$

Приведенный выше вывод исходит из того, что $x \in [a, c]$, но это не влияет на общность результатов; тот же результат имеет место, если $x \in [c, b]$. Таким образом, мы получили следующую теорему.

Теорема 16.1. Допустим, что эволюционно-алгоритмическая особь x и решение x^* одномерной оптимизационной задачи независимы и

равномерно распределены в поисковом пространстве. Тогда средняя вероятность того, что квазипротивоположность от x ближе, чем противоположность от x к решению x^*, равна 11/16.

Эти результаты были впервые представлены в публикациях [Ergezer и соавт., 2009], [Ergezer, 2011]. В этих работах также содержатся некоторые дополнительные результаты и кратко излагаются в табл. 16.1. В первой строке табл. 16.1 показано, что эволюционный алгоритм и его противоположность одинаково близки к оптимальному решению. Этот результат является ожидаемым, исходя из симметрии между x и x_o.

Таблица 16.1. Одномерные вероятности того, что некоторые противоположные точки находятся ближе к оптимальному решению, чем другие точки

Событие	Вероятность
$\lvert x_o - x^* \rvert < \lvert x - x^* \rvert$	1/2
$\lvert x_{qo} - x^* \rvert < \lvert x - x^* \rvert$	9/16
$\lvert x_{qr} - x^* \rvert < \lvert x - x^* \rvert$	11/16
$\lvert x_{qo} - x^* \rvert < \lvert x_o - x^* \rvert$	11/16
$\lvert x_{qr} - x^* \rvert < \lvert x_o - x^* \rvert$	9/16
$\lvert x_{qo} - x^* \rvert < \lvert x_{qr} - x^* \rvert$	1/2

Хотя табл. 16.1 ограничена одномерными задачами, распространение подхода в этом разделе на более высокие размерности будет концептуально прямолинейным и оставлено для дальнейших исследований. Некоторые экспериментальные результаты из более высоких размерностей показаны в публикации [Ergezer и соавт., 2009], где, по-видимому, вероятности увеличиваются до асимптоты по мере увеличения числа размерностей. См. также задачу 16.12.

Обратите внимание, что в этом разделе мы произвольно определили x^* как решение оптимизационной задачи. Мы могли бы также определить его как худшую особь в поисковом пространстве. Ключом к оппозиционному самообучению является то, что после генерации противоположной популяции лучшие N особей из исходных N особей и противоположные N особей оставляются для следующего поколения. Причина, почему оппозиционное самообучение работает, заключается в том, что квазипротивоположные и квазиотраженные точки имеют высокую вероятность быть ближе, чем произвольная эволю-

ционно-алгоритмическая особь *x*, к произвольной точке в поисковом пространстве.

Наш вывод исходит из того, что эволюционно-алгоритмическая особь *x* равномерно распределена в поисковом пространстве. Мы ожидаем, что по мере поступательного продвижения эволюционного алгоритма к более поздним поколениям большинство особей приблизится к оптимальному решению, а это означает, что *x* больше не будет равномерно распределена. По-видимому, это свидетельствует о том, что оппозиционное самообучение, должно быть, эффективнее на ранней стадии процесса поиска. При реализации оппозиционного самообучения мы можем использовать более высокую скорость перескока в начале процесса поиска, чем позже. Это тот же самый тип логики, который мы часто используем в симулированном охлаждении (см. главу 9). Мы также часто используем подобные рассуждения в эволюционном алгоритме в мутации; мы используем высокие скорости мутации в начале процесса поиска и более низкие скорости позже [Haupt и Haupt, 2004, раздел 5.9].

16.4. Коэффициент перескока

В данном разделе вводится понятие коэффициента перескока, простого расширения, которое может улучшить результативность оппозиционного самообучения (OBL). Эта идея обусловлена осознанием того, что оппозиционное самообучение требует вычислительных ресурсов. Каждая противоположная особь, которую мы генерируем, требует дополнительного оценивания приспособленности, а в реальных задачах оценивания приспособленности оно может быть весьма дорогостоящим (см. главу 21). Мы не хотим произвольно генерировать противоположные решения во время реализации эволюционного алгоритма. Мы предпочли бы генерировать противоположные решения только в том случае, если достаточно уверены, что дополнительные вычислительные усилия окупятся за счет повышения результативности.

Обратите внимание, что противоположность высоко приспособленной эволюционно-алгоритмической особи менее вероятна, чем противоположность низко приспособленной особи. То есть если эволюционный алгоритм близок к оптимальному решению, то не стоит генерировать ее противоположность. И наоборот, если эволюционный алгоритм далек от оптимального решения, то, вероятно, стоит сгенерировать ее противоположность. Разумеется, мы не знаем, находится ли данная особь близко или далеко от оптимального решения. Но мы

знаем относительные значения приспособленности каждой особи в нашей эволюционно-алгоритмической популяции. Возможно, оппозиционное самообучение должно быть реализовано так, чтобы вероятность генерации противоположной особи была функцией приспособленности этой особи. Логику алгоритма оппозиционно-биогеографической оптимизации (OBBO) на рис. 16.5 можно заменить на нечто вроде того, что показано на рис. 16.7.


```
α = оппозиционное давление ∈ [0, 1]
r1 ← U[0, 1]
If r1 < Jr then
  m = 0
  For each особь zk
    r2 ← U[0, 1]
    If r2 > αμk then
      m ← m + 1
      z̄m ← противоположность от zk
    End if
  Next особь
  {zk} ← лучшие N особей из {zk} ∪ {z̄m}
End if
```

Рис. 16.7. Оппозиционная логика на основе приспособленности. Данная логика может заменить стандартную логику алгоритма OBBO на рис. 16.5

Параметр $\alpha > 0$ на рис. 16.7 управляет давлением для генерирования противоположных особей. Вспомнив, что μ_k пропорциональна приспособленности z_k, мы видим, что оппозиционная логика на основе приспособленности делает более вероятным то, что для особи с низкой приспособленностью будет сгенерирована противоположная особь. Малое значение α приведет к созданию множества противоположных особей. В пределе $\alpha \to 0$, оппозиционная логика на основе приспособленности эквивалентна стандартной оппозиционной логике на рис. 16.5, и мы создадим противоположность для каждой эволюционно-алгоритмической особи. По мере того как α становится больше, наоборот, будет создано меньше особей. Когда $\alpha \to \infty$, никаких противоположных особей создано не будет, и алгоритм OBBO сведется к стандартному алгоритму биогеографической оптимизации (BBO). Создание противоположных особей – это риск; оно требует дополнительных вычислительных усилий, потому что должна быть вычислена приспособленность каждой противоположной особи. Стоит ли потенциальный выигрыш от новых, высоко приспособленных противоположных особей дополнительных

вычислений приспособленности? Параметр α обеспечивает нужный баланс.

Еще один способ реализации оппозиционной логики на основе приспособленности – генерировать оппозиционных особей только для наименее приспособленной доли ρ особей в популяции. Данный подход очень похож на идею, изложенную выше, но более детерминирован и может быть реализован, как показано на рис. 16.8, где $\rho \in [0, 1]$.

ρ = коэффициент перескока $\in [0, 1]$
$r \leftarrow U[0, 1]$
If $r < J_r$ **then**
  $m = 0$
  For each особь z_k
    If z_k находится в наиболее приспособленной ρ доли популяции **then**
      $m \leftarrow m + 1$
      $\bar{z}_m \leftarrow$ противоположность от z_k
    End if
  Next особь
  $\{z_k\} \leftarrow$ лучшие N особей из $\{z_k\} \cup \{\bar{z}_m\}$
End if

Рис. 16.8. Пропорциональная оппозиционная логика на основе приспособленности. Данная логика может заменить стандартную логику алгоритма OBBO на рис. 16.5. Если ρ = 1, то эта логика сводится к стандартной оппозиционной логике, показанной на рис. 16.5

Представленные в этом разделе идеи являются попыткой сделать оппозиционное самообучение интеллектуальнее, адаптивнее и эффективнее. Изобретательные исследователи могут развить другие идеи в этом направлении, для того чтобы усовершенствовать оппозиционное самообучение. В целях совершенствования оппозиционного самообучения мы вполне можем также задействовать идеи управляемой мутации из общих эволюционно-алгоритмических исследований [Zhang и соавт., 2005]. Мы продемонстрируем описанную выше логику коэффициента перескока в примере следующего далее раздела.

16.5. Оппозиционная комбинаторная оптимизация

В данном разделе приводится расширение оппозиционного самообучения (OBL) до комбинаторных оптимизационных задач. Если мы хо-

тим распространить оппозиционное самообучение на комбинаторные задачи, то нам явно нужно переосмыслить определения противоположностей из раздела 16.1. Первоначальное исследование в этой области было представлено в публикации [Ergezer и Simon, 2011].

Комбинаторная задача – это задача, для которой мы хотим найти лучший способ упорядочить множество вершин. Задача коммивояжера является хорошим примером комбинаторной задачи (см. раздел 2.5 и раздел 18). Задача коммивояжера может быть задачей с замкнутым путем и задачей с разомкнутым путем. Задача с замкнутым путем – это задача, решение которой формирует замкнутный путь, то есть маршрут начинается и заканчивается в том же городе. Задача с разомкнутым путем – это задача, решение которой посещает каждый город ровно один раз, поэтому начальный и конечный города разные. В этом разделе мы рассмотрим задачи с разомкнутым путем.

Прежде чем мы попытаемся определить противоположность особи в комбинаторном эволюционном алгоритме, мы воспользуемся простым примером, для того чтобы ввести несколько определений. Предположим, что мы пытаемся решить задачу коммивояжера для 4 городов с помощью эволюционного алгоритма. Города помечены метками A, B, C и D. Одно из кандидатных решений имеет следующий вид:

$$A \rightarrow B \rightarrow C \rightarrow D. \tag{16.33}$$

1. Мы определяем один отрезок пути как поездку между двумя смежными городами. Мы видим, что уравнение (16.33) состоит из трех ветвей: $A \rightarrow B$, $B \rightarrow C$ и $C \rightarrow D$.
2. Мы определяем близость между двумя городами как число отрезков, которые требуются для того, чтобы добраться из одного города в другой. В уравнении (16.33) A и B имеют близость, равную единице, A и C имеют близость, равную двум, и A и D имеют близость, равную трем.
3. Мы определяем общую близость маршрута как сумму близости между каждой парой смежных городов. В уравнении (16.33) общая близость равна трем, потому что каждый отрезок $A \rightarrow B$, $B \rightarrow C$ и $C \rightarrow D$ имеет близость, равную единице. Общая близость маршрута всегда равна $N - 1$, где N – это число городов.
4. Мы определяем относительную близость маршрута β как сумму близостей между каждой парой смежных городов в β, где близости получаются из некоторого другого маршрута α. Например, предположим, что у нас есть следующие ниже маршруты:

$$\begin{aligned} \alpha &: D \to C \to A \to B \\ \beta &: B \to D \to A \to C. \end{aligned} \tag{16.34}$$

Близость β относительно α равна шести. Причина этого в том, что β состоит из трех отрезков: первый отрезок $B \to D$, два города, которые имеют близость, равную трем в α; второй отрезок $D \to A$, два города, которые имеют близость, равную двум в α; и третий отрезок $A \to C$, два города, которые имеют близость, равную единице в α.

Один из способов определить противоположность маршрута α – найти маршрут β, относительная близость которого как можно больше. Это интуитивно понятно, потому что относительная близость α относительно α равна $N - 1$, что является минимально возможным значением. Используя это определение, противоположность маршрута уравнения (16.33) равна

$$C \to A \to D \to B. \tag{16.35}$$

Этот маршрут имеет близость 7 относительно уравнения (16.33), что является максимально возможным значением.

Вместе с тем задача нахождения маршрута для максимизации относительной близости сама по себе является комбинаторно-оптимизационной задачей. Это означает, что если мы хотим решить такую задачу, как задача коммивояжера, используя оппозиционное самообучение, мы должны решать комбинаторную задачу, которая состоит из нескольких комбинаторных задач в каждом поколении. Такой подход может быстро стать невозможным с точки зрения вычислений. Поэтому мы определяем жадную противоположность комбинаторной особи. Жадная противоположность оставляет исходный город без изменений, а затем вставляет город с наибольшей относительной близостью в качестве второго города. Мы устанавливаем новый третий город равным городу с наибольшей относительной близостью из нового второго города и повторяем этот процесс до завершения жадного противоположного маршрута. Данный процесс показан на рис. 16.9.

Рисунок 16.9 дает следующую ниже жадную противоположность маршрута уравнения (16.33):

$$A \to D \to B \to C. \tag{16.36}$$

Этот маршрут имеет близость, равную шести, относительно уравнения (16.33), что на единицу меньше, чем противоположный маршрут уравнения (16.35). На рис. 16.10 показан еще один, более простой спо-

соб реализации жадной противоположности. Эти алгоритмы не дают полной противоположности кандидатного решения задачи коммивояжера, но, будем надеяться, они дают почти противоположное с разумно низкими вычислительными затратами.

$\alpha = \{\alpha_1, \alpha_2, \cdots, \alpha_N\}$ = кандидатное решение
$p(\alpha_i, \alpha_j) = |i - j|$ = близость между узлами α_i и α_j
$\beta_1 \leftarrow \alpha_1$
$\beta \leftarrow \{\beta_1\}$
For $k = 2$ **to** N
  $\beta_k \leftarrow \arg\max_{\alpha} p(\alpha_{k-1}, \alpha) : \alpha \notin \beta$
    $\beta \leftarrow \beta \cup \beta_k$
Next k

Рис. 16.9. Приведенный выше псевдокод описывает алгоритм нахождения жадной противоположности β кандидатного решения α комбинаторно-оптимизационной задачи, где N – это число вершин в каждом кандидатном решении

$\alpha = \{\alpha_1, \alpha_2, \cdots, \alpha_N\}$ = кандидатное решение
For $k = 1$ **to** N
  If k является нечетным **then**
    $m \leftarrow (k + 1)/2$
  else
    $m \leftarrow N + 1 - k/2$
  End if
  $\beta_k \leftarrow \alpha_m$
Next k

Рис. 16.10. Приведенный выше псевдокод описывает простой алгоритм нахождения жадной противоположности β кандидатного решения α комбинаторно-оптимизационной задачи, где N – это число вершин в каждом кандидатном решении. Данный алгоритм эквивалентен рис. 16.9, но проще

■ Пример 16.5

В данном примере мы исследуем применение оппозиционного самообучения (OBL) для задачи коммивояжера. Применяем оператор скрещивания с инверсией по опорному элементу inver-over (см. раздел 18.3.1.5) и алгоритм биогеографической оптимизации (BBO) для выбора иммигрирующих и эмигрирующих членов популяции. Мы применяем эталон Ulysses16, который состоит из 16 городов (см. раздел C.6), и используем 10 000 вычислений функций. Напомним, что в задаче коммивояжера для 16 городов существует $16!/2 \approx 10^{13}$ возможных

решений. Таблица 16.2 показывает среднее значение и стандартное отклонение кратчайшего маршрута, найденного различными комбинациями алгоритмов BBO/OBL после 40 симуляций Монте-Карло. Результаты показывают, что результативность улучшается по мере увеличения скорости перескока J_r и коэффициента перескока ρ. (См. рис. 16.8 относительно определения коэффициента перескока ρ.) Если J_r и ρ увеличиваются слишком сильно, то результативность снижается, хотя эти результаты мы здесь не показывали.

Таблица 16.2. Пример 16.5: результаты оппозиционной биогеографической оптимизации для решения эталонной задачи Ulysses16. Результаты показывают среднее значение и стандартное отклонение лучшего решения, найденного по 40 симуляциям Монте-Карло. $J_r = 0$ соответствует стандартному алгоритму биогеографической оптимизации (BBO) без оппозиционного самообучения (OBL). В общем случае результативность улучшается по мере увеличения скорости перескока J_r и коэффициента перескока ρ

	$\rho = 0.1$	$\rho = 0.2$	$\rho = 0.3$	$\rho = 0.4$
$J_r = 0.0$	7266 ± 353	7266 ± 353	7266 ± 353	7266 ± 353
$J_r = 0.1$	7153 ± 289	7284 ± 244	7122 ± 296	7127 ± 270
$J_r = 0.2$	7160 ± 297	7100 ± 324	7047 ± 251	6910 ± 315
$J_r = 0.3$	7180 ± 267	6976 ± 336	6945 ± 270	6869 ± 319
$J_r = 0.4$	7127 ± 201	7005 ± 326	6910 ± 265	6776 ± 207

■

16.6. Дуальное самообучение

Оппозиционное самообучение похоже на дуальное самообучение, которое впервые было предложено в 1990-х годах [Collard и Aurand, 1994], [Collard и Gaspar, 1996] и вновь в начале 2000-х годов [Yang, 2003a], [Yang, 2003b]. Позже в публикации [Yang и Yao, 2005] было предложено брать дублера (dual) только худших особей в популяции. Встраивание идей дуального самообучения в алгоритм оппозиционно-биогеографической оптимизации (OBBO) на рис. 16.5 дает дуальную логику рис. 16.11. Обратите внимание, что рис. 16.5 может заменить раздел «Оппозиционная логика» на рис. 16.5.

$\{w_k\} \leftarrow \{N_d$ худшие особи в популяции$\}$
Применить $\{w_k\}$, чтобы создать популяцию из N_d противоположностей $\{w_k\}$
For $i = 1$ **to** N_d

If $\overline{w}_k$ лучше w_k, **then** заменить w_k на $\overline{w}_k$ в популяции
Next i

Рис. 16.11. Приведенный выше псевдокод описывает дуальную логику. Эта логика может заменить блок «Оппозиционная логика» на рис. 16.5. N_d – это число дублеров, которые мы создаем для каждого поколения и которые могут быть адаптированы, как описано в тексте

Число дублеров, которые мы создаем для каждого поколения, может быть адаптировано для оптимальной работы эволюционного алгоритма. В работе [Yang и Yao, 2005] предлагается следующая ниже адаптационная схема, которую мы выполняем каждое поколение и которую можем добавить в конец дуальной логики рис. 16.11:

$$\begin{aligned} N_v &\leftarrow |\overline{w}_k : f(\overline{w}_k) > f(w_k)| \\ s &\leftarrow (\delta N_d - N_v)/N \\ N_d &\leftarrow \beta^8 N_d \\ N_d &\leftarrow \max(N_d, N_{d,\min}) \\ N_d &\leftarrow \min(N_d, N_{d,\max}) \end{aligned} \tag{16.37}$$

В уравнении (16.37) $f(\cdot)$ является функцией приспособленности, поэтому более крупное значение $f(\cdot)$ обозначает более результативную особь. N_v – это число «допустимых» дублеров из предыдущего поколения, то есть число дублеров $\overline{w}_k$, которые были лучше, чем особи, от которых они были получены. Параметр $\delta \in (0, 1)$ является порогом принятия решения. Если доля допустимых дублеров больше δ, то в следующем поколении мы хотим создать больше дублеров; но если доля допустимых дублеров меньше δ, то в следующем поколении мы хотим создать меньше дублеров. $\beta \in (0,1)$ – это константа, которая управляет скоростью адаптации. $N_{d,\min}$ и $N_{d,\max}$ – это минимально и максимально допустимые значения N_d. Для констант в уравнении (16.37) предлагаются следующие ниже значения [Yang и Yao, 2005]:

$$\begin{aligned} \text{Initial } N_d &= 0.5N \\ \delta &= 0.9 \\ \beta &= 0.5 \\ N_{d,\min} &= 1 \\ N_{d,\max} &= 0.5N \end{aligned} \tag{16.38}$$

где N – размер популяции. Дуальное самообучение также может быть распространено на популяционное интекрементное самообучение

(PBIL) для решения задач динамической оптимизации [Yang и Yao, 2005], [Yang и Yao, 2008b]. В алгоритме PBIL вектор дуальной вероятности p_d симметричен вектору вероятности p по отношению к 50%-му значению вероятности: $p_d = 1 - p$.

16.7. Заключение

Оппозиционное самообучение (OBL) является относительно новоиспеченным инструментом оптимизации, и поэтому существует ряд возможных расширений. Адаптивное оппозиционное самообучение могло бы стать интересным направлением. Адаптация может реализовываться несколькими разными способами. Например, поскольку эволюционно-алгоритмическая популяция имеет тенденцию сходиться к хорошим решениям, по мере того как число поколений увеличивается, пожалуй, оппозиционное самообучение должно реализовываться чаще на ранних стадиях работы эволюционного алгоритма и реже на поздних статиях. Это можно реализовать, сделав скорость перескока J_r и/или коэффициент перескока p убывающей функцией от номера поколения. Кроме того, в этой главе мы ограничили степень оппозиции нулем или единицей (см. уравнение (16.9)). Мы могли бы реализовать алгоритм адаптивного оппозиционного самообучения, вероятностно уменьшая степень оппозиции вместе с числом поколений.

Другие способы реализации адаптации в алгоритме адаптивного оппозиционного самообучения могли бы включать изменение рода противоположности по мере увеличения числа поколений или изменение рода противоположности на основе индивидуальной приспособленности. Низко приспособленные особи должны извлекать выгоду из радикальных оппозиционных операций больше, чем высоко приспособленные особи, поэтому, возможно, сверхпротивоположность должна быть зарезервирована для низко приспособленных особей.

Несмотря на то что на основе оппозиционного самообучения было проделано много математического моделирования с использованием теории вероятностей, оппозиционное самообучение как оптимизационный алгоритм еще не смоделировано математически. Важные области для будущих исследований алгоритма оппозиционного самообучения включают адаптацию математических моделей эволюционных алгоритмов (см. главу 4 и раздел 7.6) относительно встраивания оппозиционного самообучения.

Дополнительные исследования также могут быть направлены на изучение взаимосвязи между оппозиционным самообучением и эво-

люцией через поиск новизны [Lehman и Stanley, 2011]. Кроме того, поскольку алгоритм оппозиционного самообучения основан на социальных революциях, было бы интересно встроить в данный алгоритм большее количество культурных моделей (см. главу 15). Дополнительные учебные материалы по оппозиционному самообучению можно найти в публикациях [Tizhoosh, 2005] и [Tizhoosh и соавт., 2008].

Задачи

Письменные упражнения

16.1. Уравнение (16.4) определяет противоположность по модулю. Дайте эквивалентное определение, которое не использует функцию mod.

16.2. Приведите пример двумерной области, где противоположность точки x в области может быть вне области.

16.3. Рассмотрим точку $(x, y) = (2, 2)$, где область x равна $[1, 5]$ и область y равна $[1, 7]$. Какой будет противоположность, квазипротивоположность, сверхпротивоположность и квазиотраженная противоположность этой точке?

16.4. Рассмотрим точку $(x, y) = (2, 2)$, где область x равна $[1, 5]$ и область y равна $[1, 7]$. Какого рода противоположностью является точка $(2, 5)$?

16.5. Объясните, как можно модифицировать алгоритм оппозиционно-биогеографической оптимизации (OBBO) на рис. 16.5, для того чтобы в нем использовалась адаптивная скорость перескока.

16.6. Как допущение примера 16.4 противоречит второму допущению раздела 16.3? Какое допущение вы считаете более разумным?

16.7. Предположим, что скорость эмиграции μ_k алгоритма биогеографической оптимизации (BBO) на рис. 16.7 является случайной величиной, равномерно распределенной на $[0, 1]$.

a) Какова вероятность того, что противоположная особь будет сгенерирована для случайно отобранной особи z_k?

b) Имеет ли вероятность, которую вы выведете, интуитивный смысл в пределе вида $\alpha \to 0$ и $\alpha \to \infty$?

16.8. На рис. 16.9 и 16.10 мы произвольно определили начальную точку жадной противоположности β маршрута α, которая совпадает с начальной точкой A. Однако близость β относительно α зависит от начальной точки. Рассмотрим маршрут $\alpha = \{A \to B \to C \to D \to E\}$.

a) Какова жадная противоположность β, если начальный город жадной противоположности β является A, и какова его близость по отношению к α?

b) Чему равна жадная противоположность β, если начальный город жадной противоположности β равен B, и какова его близость относительно α?

16.9. Каковы минимальные и максимальные значения s в уравнении (16.37)?

16.10. Предположим, что мы используем дуальную адаптационную логику уравнения (16.37) с рекомендуемыми константами и что $N_v = 0.1N$ после первого поколения. Каково будет значение N_d во время второго поколения?

Компьютерные упражнения

16.11. Напишите программу, которая устанавливает $x^* \sim U[a, b]$ для некоторых произвольно отобранных значений a и b – таких, что $a < b$. Назначьте $x \sim U[a, b]$, примите x_o равным отраженной противоположности x, а также примите x_{qo} равным квазипротивоположности x. Проверьте, какая противоположность ближе к x^*. Выполните программу несколько тысяч раз для подтверждения теоремы 16.1.

16.12. Решите задачу 16.11 для n = от 1 до 20, где n – число размерностей. Постройте график вероятности того, что $||x_{qo} - x^*||_2 < ||x_o - x^*||_2$ как функции от n. Прокомментируйте свои результаты.

16.13. Повторите пример 16.4 с оппозиционной логикой, пропорциональной приспособленности на рис. 16.8. Используйте $\rho = 0.1$, 0.5 и 1.0. Каково среднее значение (по 20 симуляциям Монте-Карло) наименьшей стоимости, найденной алгоритмом оппозиционно-биогеографической оптимизации (ОВВО) для каждого из этих значений ρ? Прокомментируйте свои результаты.

Глава 17

Другие эволюционные алгоритмы

То, что было, будет снова, то, что было сделано, будет сделано снова; нет ничего нового под солнцем.

– Екклесиаст 1:9

В этой главе дается обзор нескольких эволюционных алгоритмов, которые мы не успели обсудить в предыдущих главах. Некоторые из алгоритмов находятся в сумрачной зоне между эволюционными и неэволюционными алгоритмами, и поэтому эта глава, по всей видимости, является хорошим местом для их обобщения. Другие алгоритмы в данной главе явно эволюционны, но также являются новыми, и поэтому пока не ясно, какое влияние они окажут на будущую теорию и практику применения эволюционных алгоритмов. При принятии решения о том, какие эволюционные алгоритмы включить в эту книгу, было ясно, что генетические алгоритмы, эволюционное программирование, эволюционные стратегии и генетические программы должны быть охвачены в части II из-за их основополагающей важности и их истории. Решение о том, какие эволюционные алгоритмы включить в часть III, было менее ясным. Эволюционные алгоритмы, рассмотренные в предыдущих главах, отражают личные предубеждения автора и его мнение о важности каждого алгоритма.

В этой книге несколько оптимизационных алгоритмов не имеет своей главы, но мы должны их обсудить, по крайней мере, в некоторой степени. Такова цель данной главы. Алгоритмы в этой главе не обязательно менее важны, менее эффективны или менее полезны, чем алгоритмы в предыдущих главах. Их помещение сюда просто отражает ограниченный опыт и субъективные интересы автора.

17.1. Табуированный поиск

Табуированный поиск, или поиск с запретами (tabu search, TS), был представлен в публикации [Glover и McMillan, 1986]. Табу означает «запретный», «запрещенный» или «неразрешенный». Запрещенные предметы, речь или поступки могут основываться на культуре, религии, морали или политике. Табуированный поиск не является строго популяционным подходом к оптимизации, но его можно считать эволюционным алгоритмом, потому что он основан на естественном мире и является итеративным процессом поиска. Табуированный поиск опирается на идею, что если определенный участок поискового пространства уже был посещен в процессе поиска, то он табуируется, и поисковому алгоритму не рекомендуется посещать его снова. Аналогичным образом, если определенная поисковая стратегия уже использовалась в процессе поиска, то эта поисковая стратегия табуируется, и поисковому алгоритму не рекомендуется использовать ее снова.

На рис. 17.1 представлен базовый алгоритм табуированного поиска, где T – это список табуируемых признаков и x_0 – текущее лучшее кандидатное решение. Когда мы создаем детей из x_0, то не допускаем, чтобы процесс поиска включал признаки из T. Когда улучшенное кандидатное решение x' найдено, мы добавляем новые признаки из x' в список табу T. Мы периодически удаляем признаки из T, возможно, исходя из того, как долго они были в T. Этим симулируется постепенное изменение табу со временем, как мы видим в человеческом обществе. Обратите внимание, что проверка на принадлежность (признаков x') $\notin T$ на рис. 17.1 намеренно оставлена неоднозначной. Детали этой проверки зависят от задачи, метода, используемого для создания соседей x_0, предпочтений пользователя и других подробностей.

Инициализировать кандидатное решение x_0.
While not (критерий останова)
  Дети $\leftarrow \emptyset$
  While |Дети| $< M$
    Создать соседа x' из x_0
    If (признаки соседа x') $\notin T$
      Дети $\leftarrow$ Дети $\cup\ x'$
    End if
  End while
  $x' \leftarrow \arg\min(f(x) : x \in$ Дети$)$
  If $f(x') < f(x_0)$

$T \leftarrow T \cup$ (признаки из x')
$x_0 \leftarrow x'$
End if
Удалить старые признаки из T
Next поколение

Рис. 17.1. Схематичное описание алгоритма табуированного поиска (TS) для поиска минимума функции $f(x)$. Каждая итерация включает создание M детей, где M – это указываемый пользователем параметр

Мы можем использовать ряд вариаций алгоритма рис. 17.1. Например, у нас могут быть разные степени табу. У нас также может иметься список табу, содержащий не признаки, которые следует избегать, а поисковые стратегии, которых следует избегать. Табуированный поиск часто используется для усиления других эволюционных алгоритмов. Краткий обзор в этом разделе предназначен для того, чтобы дать читателю достаточно информации для реализации простого алгоритма табуированного поиска, познакомиться с основной идеей табуированного поиска и узнать больше деталей из других источников. Дополнительные материалы для чтения по табуированному поиску можно найти в публикациях [Reeves, 1993, глава 3], [Glover и Laguna, 1998], [Gendreau, 2003] и [Gendreau и Potvin, 2010].

17.2. Алгоритм искусственного косяка рыб

Алгоритм искусственного косяка рыб (artificial fish swarm algorithm, AFSA), который был предложен в публикации [Li и соавт., 2003] и иногда называется алгоритмом искусственной стаи рыб, в общих чертах основан на роевом поведении рыб. Положение искусственной рыбы в поисковом пространстве обозначается как x_i, где $i \in [1, N]$ – индекс рыбы, а N – количество рыбы в косяке. Мы обозначаем поисковую область для каждой размерности как $[l_k, u_k]$ для $k \in [1, n]$, где n – это размерность поискового пространства. У рыб есть поле зрения, в пределах которого они могут видеть других рыб и за пределами которого они не могут видеть других рыб. Визуальный диапазон рыбы определяется как

$$v = \delta \max_k (u_k - l_k), \tag{17.1}$$

где δ – это регулировочный параметр, который часто постепенно уменьшается в процессе оптимизации. В публикации [Fernandes и соавт., 2009] было обнаружено, что значения δ от 1 до 10 дают хорошую результативность для задач от двух до четырех размерностей, хотя этот

диапазон, возможно, потребуется скорректировать для задач с бóльшим числом размерностей. Индексы рыб, находящихся в пределах визуального диапазона рыбы x_i, обозначаются следующим образом:

$$V_i = \left\{ j \neq i : ||x_i - x_j||_2 \leq v \right\}. \tag{17.2}$$

Говорят, что рыба находится в заполненной среде, если в ее визуальном диапазоне есть относительно много рыбы:

$$\begin{aligned} \frac{|V_i|}{N} > \theta \quad &\Rightarrow \text{Визуальный диапазон } x_i \text{ заполнен} \\ \frac{|V_i|}{N} \leq \theta \quad &\Rightarrow \text{Визуальный диапазон } x_i \text{ не заполнен} \end{aligned} \tag{17.3}$$

где θ – регулировочный параметр. В публикации [Fernandes и соавт., 2009] обнаружено, что $\theta \approx 1$ дает хорошую результативность для низкоразмерных задач. Рыба алгоритма AFSA имеет пять разных форм поведения, которые мы обсудим далее: случайное, преследование, роение, поиск и резкое изменение.

17.2.1. *Случайное поведение*

Иногда рыбы ведут себя хаотично, то есть двигаются в случайном направлении в поисковом пространстве. На рис. 17.2 показан псевдокод для случайного перемещения. Случайное поведение происходит, если рыба не видит другой рыбы в пределах своего визуального диапазона либо если процесс оптимизации застаивается. Застаивание, или стагнация, определяется как неспособность лучшей особи в популяции значительно улучшиться в течение предыдущих m поколений:

$$\arg\min_x f_{t-m}(x) - \arg\min_x f_t(x) < \eta \Rightarrow \text{Стагнация}, \tag{17.4}$$

где $f_t(x)$ – это значение функции оптимизации особи x в t-м поколении, m – положительный целочисленный регулировочный параметр, а η – неотрицательный регулировочный параметр. В уравнении (17.4) мы исходим из того, что наша оптимизационная задача является минимизационной задачей. В публикации [Fernandes и соавт., 2009] было обнаружено, что $m \approx 10n$ и $\eta \approx 10^{-4}$ дает хорошую результативность для низкоразмерных эталонных задач, где n – это размерность задачи.

For k – 1 **to** n
  $r \leftarrow U[0, 1]$
  If $r < 1/2$ **then**

```
        ρ ← U[0, 1]
        y_i(k) ← x_i(k) + ρmin(v, u_k – x_i(k))
    else
        ρ ← U[0, 1]
        y_i(k) ← x_i(k) – ρmin(v, x_i(k) – l_k)
    End if
Next размерность
```

Рис. 17.2. Случайное поведение в алгоритме искусственного косяка рыб. Данный псевдокод показывает случайное перемещение рыбы x_i в новое положение y_i, где n – число размерностей в оптимизационной задаче, $U[0, 1]$ – случайное число, равномерно распределенное в [0, 1], а v – визуальный диапазон, определенный в уравнении (17.1)

17.2.2. *Поведение преследования*

Иногда рыба движется к рыбе, которая находится в месте наибольшей концентрации пищи в пределах ее визуального диапазона. Поведение преследования для рыбы x_i описывается следующим образом:

$$j^* \leftarrow \arg\min_j \{f(x_j) : j \in V_i\}$$

$$y_i \leftarrow x_i + r(x_{j^*} - x_i) \tag{17.5}$$

где $r \in [0, 1]$ – это равномерно распределенная случайная величина, а y_i – новое положение x_i. Мы снова исходим из того, что наша оптимизационная задача является минимизационной задачей, поэтому j^* – это индекс рыбы в пределах визуального диапазона, который имеет лучшую результативность на нашей оптимизационной задаче. Если рыба не находится в пределах визуального диапазона любой другой рыбы, то она не может участвовать в поведении преследования. Кроме того, x_i преследует другую рыбу только в том случае, если лучшая рыба x_{j^*} в пределах ее визуального диапазона имеет лучшую результативность оптимизации, чем x_i.

17.2.3. *Роевое поведение*

Рыбы – существа социальные, поэтому иногда они собираются вместе. В этом случае рыба x_i движется к центроиду c_i рыб, которые находятся в пределах ее визуального диапазона. Роевое поведение рыбы x_i описывается следующим образом:

$$\begin{aligned} c_i &\leftarrow \frac{1}{|V_i|}\sum_{j \in V_i} x_j \\ y_i &\leftarrow x_i + r(c_i - x_i) \end{aligned} \tag{17.6}$$

где $r \in [0, 1]$ – это равномерно распределенная случайная величина, а y_i – новое положение рыбы x_i. Если рыба не находится в пределах визуального диапазона любой другой рыбы, то она не может участвовать в роевом поведении. Роение происходит, только если визуальный диапазон рыбы не пуст, не заполнен и $f(c_i)$ лучше, чем $f(x_i)$.

17.2.4. *Поисковое поведение*

Когда рыба видит другую рыбу, у которой больше пищи, она движется к этой рыбе. Поисковое поведение рыбы x_i описывается следующим образом:

$$\begin{aligned} j &\leftarrow \text{случайное целое} \in V_i \\ y_i &\leftarrow x_i + r(x_j - x_i) \end{aligned} \tag{17.7}$$

где $r \in [0, 1]$ – это равномерно распределенная случайная величина, а y_i – новое положение рыбы x_i. Поисковое поведение – это движение рыбы в направлении случайно отобранной рыбы, находящейся в пределах ее визуального диапазона. Если рыба не находится в пределах визуального диапазона любой другой рыбы, то она не может участвовать в поисковом поведении. Поиск происходит в том случае, если визуальный диапазон рыбы заполнен, либо если визуальный диапазон рыбы не заполнен и $f(c_i)$ в уравнении (17.6) хуже, чем $f(x_i)$, либо если визуальный диапазон рыбы не заполнен и $f(x_{j^*})$ в уравнении (17.5) хуже, чем $f(x)$.

17.2.5. *Скачущее поведение*

Иногда рыба случайно «скачет» по поисковому пространству. Такое поведение аналогично рыбе, которая выпрыгивает из воды и случайно приводняется в другом месте. Скачки происходят для одной случайно отобранной рыбы, если процесс оптимизации стагнирует, как указано в уравнении (17.4). На рис. 17.3 показан псевдокод скачущего поведения рыбы.

For k – 1 **to** n
  $r \leftarrow U[0, 1]$
  $\rho \leftarrow U[0, 1]$
  If $r < 1/2$ **then**

$x_i(k) \leftarrow x_i(k) + \rho(u_k - x_i(k))$
else
$x_i(k) \leftarrow x_i(k) - \rho(x_i(k) - l_k)$
End if
Next размерность

Рис. 17.3. Скачущее поведение в алгоритме искусственного косяка рыб. В этом псевдокоде показан скачок особи x_i в алгоритме искусственного косяка рыб, где n – это число размерностей в оптимизационной задаче, а $U[0, 1]$ – случайное число, равномерно распределенное в [0, 1]

17.2.6. *Резюме алгоритма искусственного косяка рыб*

В алгоритме искусственного косяка рыб (AFSA) используется жадный метод отбора. То есть после случайного, преследующего, роевого и поискового поведения рыба x_i перемещается на новое место y_i, только если это новое место лучше прежнего. На рис. 17.4 показан псевдокод алгоритма искусственного косяка рыб (AFSA), который, как представляется, имеет поведение, схожее с алгоритмом оптимизациии на основе роя частиц (PSO). Исследователи предложили множество вариаций и гибридов алгоритма искусственного косяка рыб [Neshat и соавт., 2012]. Анализ и моделирование алгоритма AFSA математически со встраиванием дополнительных признаков из биологического поведения рыб и прояснением взаимосвязи между алгоритмами AFSA и PSO могут быть важными и плодотворными областями будущих исследований алгоритма AFSA.

N = размер популяции
Инициализировать случайную популяцию кандидатных решений $\{x_i\}$ для $i \in [1, N]$.
While not (критерий останова)
For each особь x_i
Найти рыбу в визуальном диапазоне рыбы x_i, как показано в уравнении (17.2).
If $V_i = \emptyset$ **then**
$y_i \leftarrow$ случайное движение, как показано на рис. 17.2
else if визуальный диапазон рыбы x_i заполнен (см. уравнение (17.3)) **then**
$y_i \leftarrow$ поисковое движение, как показано в уравнении (17.7)
else
If $f(c_i) < f(x_i)$ (см. уравнение (17.6)) **then**
$y_i \leftarrow$ роевое движение, как показано в уравнении (17.6)
else
$y_i \leftarrow$ поисковое движение, как показано в уравнении (17.7)
End if

```
        If f(x_j*) < f(x_i) (см. уравнение (17.5) then
            y_i ← движение преследования, как показано в уравнении (17.5)
        else
            y_i ← поисковое движение, как показано в уравнении (17.7)
        End if
        y_i ← arg min {f(x_i), f(y_i)}
    End if
  Next особь
  x_i ← arg min {f(x_i), f(y_i)} для i ∈ [1, N]
  If алгоритм стагнирует, как показано в уравнении (17.4) then
      j ← случайное целое ∈ [1, N]
      x_j ← скачущее движение, как показано на рис. 17.3
  End if
Next поколение
```

Рис. 17.4. Алгоритм искусственного косяка рыб (AFSA) для минимизации n-мерной функции $f(x)$, где x_i – это i-е кандидатное решение

17.3. Оптимизатор на основе группового поиска

Оптимизатор, или оптимизация, на основе группового поиска (group search optimizer, GSO) основывается на пищедобывательном поведении животных [He и соавт., 2009]. Данная основа аналогична алгоритму оптимизации на основе косяка рыб (раздел 17.2) и оптимизации на основе бактериальной кормодобычи (раздел 17.6), но оптимизатор на основе группового поиска опирается на наблюдаемое поведение наземных животных.

Некоторые животные сосредоточивают свои усилия на поиске пищи; эти животные называются производителями. Другие животные сосредоточивают свои усилия на том, чтобы следовать за иными животными и эксплуатировать успех иных в поисках пищи; этих животных называют присоединившимися или иждивенцами. Оптимизатор на основе группового поиска включает в себя третий тип животных под названием бродяг, которые выполняют случайное блуждание в поисках ресурсов. У каждой особи есть местоположение в n-мерном поисковом пространстве, обозначаемое как x_i, и курсовой угол, обозначаемый как $\phi_i = [\phi_{i,1} \ldots \phi_{i,n-1}]$.

Производители

Оптимизатор на основе группового поиска исходит из того, что в популяции есть только один производитель. Роль производителя в каждом поколении берет на себя особь с наименьшей стоимостью. Каждое поколение-производитель сканирует 3 точки в его прямой окрестности в поисках более низкой функции стоимости, чем в его настоящем положении в поисковом пространстве. Это соответствует локальному поиску. Если мы обозначим производителя как x_p, то эти три точки следующие:

$$\begin{aligned} x_z &= x_p + r_1 l_{\max} D(\phi_p) \\ x_r &= x_p + r_1 l_{\max} D(\phi_p + r_2 \theta_{\max}/2) \\ x_l &= x_p + r_1 l_{\max} D(\phi_p - r_2 \theta_{\max}/2) \end{aligned} \tag{17.8}$$

где r_1 – это нормально распределенная случайная величина с нулевым средним и единичной дисперсией[1]; $r_2 \in [0, 1]$ – равномерно распределенная случайная величина; ϕ_p – курсовой угол x_p; $l_{\max}$ – регулировочный параметр, который определяет, насколько далеко производитель может видеть; $\theta_{\max}$ – регулировочный параметр, который определяет, насколько далеко производитель может поворачивать голову; $D(\cdot)$ – трансформация полярных координат в прямоугольную (декартову) систему координат, определенная как

$$\begin{aligned} D(\phi_p) &= [d_1 \ldots d_n] \\ d_1 &= \prod_{q=1}^{n-1} \cos \phi_{p,q} \\ d_j &= \sin \phi_{p,j-1} \prod_{q=j}^{n-1} \cos \phi_{p,q} \quad \text{для } j \in [2, n-1] \\ d_n &= \sin \phi_{p,j-1} \end{aligned} \tag{17.9}$$

Если производитель находит более низкое значение функции стоимости в одной из точек, определенных уравнением (17.8), то он немедленно перемещается в эту точку; в противном случае он остается там, где он есть, и случайно перемещает свой курсовой угол ϕ_p в новое значение. Если производитель не может найти более низкую точку после $\alpha_{\max}$ поисков, то он перемещает свой курсовой угол обратно в значение, которое было $\alpha_{\max}$ поколений назад. Вместе с тем не ясно, почему эта

[1] Обратите внимание, что это определение r_1 позволяет производителю смотреть назад и вперед в трех указанных направлениях.

последняя стратегия должна иметь какое-либо влияние на результативность оптимизации, и поэтому мы вполне спокойно можем ею пренебречь.

Иждивенцы

Иждивенцы обычно перемещаются в сторону производителя. Но они не движутся непосредственно к производителю; вместо этого они движутся к производителю в своего рода зигзагообразном направлении, что позволяет им во время своих перемещений искать более дешевые значения функции. Движение иждивенца моделируется следующим образом:

$$x_i \leftarrow x_i + r_3 \circ (x_p - x_i), \tag{17.10}$$

где r_3 – это n-мерный вектор случайных величин, каждая из которых равномерно распределена по [0, 1]; а $\circ$ – поэлементное умножение.

Бродяги

Бродяги случайно перемещаются по поисковому пространству в поисках участков с низкими значениями функции стоимости. Движение бродяги моделируется следующим образом:

$$\begin{aligned} \phi_i &\leftarrow \phi_i + \rho\alpha_{\max} \\ x_i &\leftarrow x_i + \alpha_{\max} l_{\max} r_1 D(\phi_i) \end{aligned} \tag{17.11}$$

где $\alpha_{\max}$ – это регулировочный параметр, который определяет, как далеко бродяга может повернуть голову; $\rho \in [-1, 1]$ – равномерно распределенная случайная величина; где $l_{\max}$ – регулировочный параметр, связанный с максимальным расстоянием, которое бродяга может пропутешествовать в одном поколении, и такой же, что и $l_{\max}$ в уравнении (17.8), и r_1 – нормально распределенная случайная величина с нулевым средним и единичной дисперсией.

Резюме

На рис. 17.5 дано схематичное описание алгоритма оптимизации на основе группового поиска (GSO) и показано, что данный алгоритм имеет несколько регулировочных параметров. Обратите внимание, что на рис. 17.5 указано, что одна особь является производителем, около 80 % особей являются иждивенцами и около 20 % особей – бродягами. В публикации [He и соавт., 2009] изучается влияние этих настроек и других регулировочных параметров и рекомендуются следующие ниже:

$$\begin{aligned} \alpha_{\max} &= \text{round}\sqrt{n+1} \\ \theta_{\max} &= \pi / \alpha^2_{\max} \\ \alpha_{\max} &= \theta_{\max}/2 \\ l_{\max} &= \| U - L \|_2 \end{aligned} \tag{17.12}$$

где n-мерные векторы U и L – это соответственно верхняя и нижняя границы поискового пространства.

Алгоритм оптимизации на основе группового поиска (GSO) похож на алгоритм оптимизации на основе роя частиц (PSO). Одно из различий заключается в том, что в алгоритме PSO каждая особь сохраняет память о своих предыдущих местоположениях в поисковом пространстве. Другое отличие состоит в том, что в алгоритме PSO каждая особь выполняет одну и ту же поисковую стратегию. Одной из отличительных особенностей алгоритма GSO является его блуждающее поведение, хотя такое поведение также наблюдается в алгоритме оптимизации на основе роя частиц с использованием стимулятора-зубатки (catfish PSO) (см. раздел 11.7). Перспективные направления будущих исследований алгоритма GSO включают математическое моделирование и анализ, онлайновую адаптацию регулировочных параметров, а также встраивание дополнительных природно-обусловленных признаков.

N = размер популяции
Инициализировать случайную популяцию кандидатных решений $\{x_i\}$ для $i \in [1, N]$.
Случайно инициализировать курсовой угол φ_i каждого кандидатного решения x_i.
While not (критерий останова)
  Найти производителя: $x_p \leftarrow \arg\min_{x_i}\{f(x_i) : i \in [1, N]\}$
  $\{x_z, x_r, x_l\} \leftarrow$ результат сканирования из уравнения (17.8)
  If $\min\{f(x_z), f(x_r), f(x_l)\} < f(x_p)$ **then**
    $x_p \leftarrow \arg\min\{f(x_z), f(x_r), f(x_l)\}$
  else
    $\rho \leftarrow U[-1, 1]$
    $\phi(x_p) \leftarrow \phi(x_p) + \rho\alpha_{\max}$
  End if
  For each $x_i \neq x_p$
    $r_2 \leftarrow U[0, 1]$
    If $r_2 < 0.8$
      Дать x_i дармоедничать, используя уравнение (17.10).
    else
      Дать x_i бродить, используя уравнение (17.11).
    End if

Next особь
Next поколение

Рис. 17.5. Оптимизатор на основе группового поиска (GSO) для минимизации n-мерной функции $f(x)$, где x_i – это i-е кандидатное решение

17.4 . Алгоритм перемешанных лягушачьих прыжков

Алгоритм перемешанных лягушачьих прыжков (shuffled frog leaping algorithm, SFLA) был описан в публикациях [Eusuff и Lansey, 2003], [Eusuff и соавт., 2006] как гибрид алгоритма оптимизации на основе роя частиц (PSO) и перемешанной комплексной эволюции (shuffled complex evolution, SCE). Алгоритм перемешанной комплексной эволюции основывается на идее разрешить субпопуляциям эволюционировать независимо, при этом периодически допуская взаимодействия между субпопуляциями [Duan и соавт., 1992], [Duan и соавт., 1993]. В алгоритме перемешанной комплексной эволюции используется вероятностный отбор родителей в каждой субпопуляции, а также для предотвращения застоя случайно создаются новые особи. Алгоритм перемешанных лягушачьих прыжков (SFLA) опирается на идеи алгоритмов SCE и PSO.

На рис. 17.6 показана стратегия глобального поиска в алгоритме перемешанных лягушачьих прыжков (SFLA). Начнем с создания множества из N кандидатных решений. Затем мы делим эти N особей на m субпопуляций, называемых также мемплексы. Обычно N кратно m, так что каждая субпопуляция содержит одинаковое число особей. Затем мы формируем алгоритм локального поиска в каждой субпопуляции. В начале следующего поколения мы перемешиваем популяцию таким образом, чтобы каждая особь случайно была отнесена к новой субпопуляции. Общие регулировочные параметры для алгоритма SFLA включают размер популяции N, равный около 200 и приблизительно $M = 20$ субпопуляций [Elbeltagi и соавт., 2005].

Инструкция «Выполнить локальный поиск» на рис. 17.6 обозначает исполнение алгоритма на рис. 17.7. Во время локального поиска каждая субпопуляция независимо выполняет эволюционный поиск в течение $i_{\max}$ итераций. Каждую итерацию мы обновляем только x_w, то есть субпопуляцию с худшей стоимостью:

$$x_w \leftarrow x_w + r(x_b - x_w), \tag{17.13}$$

где $r \in [0, 1]$ – это равномерно распределенное случайное число, а x_b – субпопуляция с лучшей стоимостью. Если уравнение (17.13) не улучшает x_w, то мы обновляем ее снова следующим образом:

$$x_w \leftarrow x_w + r(x_g - x_w), \tag{17.14}$$

где $r \in [0, 1]$ – новое случайное число, а x_g – глобально лучшая особь из всех m субпопуляций. Если уравнение (17.14) не улучшает x_w, то мы заменяем x_w случайной особью. Лимит итераций $i_{\max} = 10$ является общим регулировочным параметром на рис. 17.7 [Elbeltagi и соавт., 2005]. Перспективные направления будущих исследований алгоритма SFLA включают математическое моделирование и анализ, а также встраивание дополнительных природно-обусловленных признаков.

Инициализировать случайную популяцию $\{x_i\}$ для $i \in [1, N]$.
While not (критерий останова)
  Случайно разделить популяцию на m субпопуляций.
  For each субпопуляций $i = 1$ **to** m
    Выполнить локальный поиск в i-й субпопуляции (рис. 17.7).
  Next субпопуляция
Next поколение

Рис. 17.6. Приведенный выше псевдокод схематично описывает стратегию глобального поиска в алгоритме перемешанных лягушачьих прыжков (SFLA)

Найти лучшую особь во всей популяции, x_g.
For $i = 1$ **to** $i_{\max}$
  Найти лучшую и худшую субпопуляционных особей, x_b и x_w.
  Применить уравнение (17.13) для обновления x_w.
  If обновление не улучшило x_w **then**
    Применить уравнение (17.14) для обновления x_w.
    If обновление не улучшило x_w **then**
      $x_w \leftarrow$ случайно сгенерированная особь
    End if
  End if
Next итерация

Рис. 17.7. Приведенный выше псевдокод схематично описывает стратегию локального поиска в алгоритме перемешанных лягушачьих прыжков (SFLA)

17.5. Светлячковый алгоритм

Светлячковый алгоритм (firefly algorithm) был введен в публикациях [Yang, 2008b, глава 8], [Yang, 2010b]. Светлячковый алгоритм основывается на привлечении светлячков друг к другу. Привлечение опирается на воспринимаемую светлячками яркость, которая экспоненциально уменьшается с расстоянием. Светлячка привлекают только те светлячки, которые ярче него.

На рис. 17.8 показан псевдокод светлячкового алгоритма. Кода $\gamma \to 0$, все светлячки притягиваются друг к другу одинаково, что соответствует нулевой дисперсии света в атмосфере. Мы наблюдаем такое поведение в вакууме. Когда $\gamma \to \infty$, светлячки друг к другу совсем не притягиваются, что соответствует случайному полету и случайному поиску. Мы наблюдаем такое поведение в плотном тумане. Параметры β_0 и α определяют компромисс между эксплуатацией (привлечение других светлячков) и разведыванием (случайный поиск). Типичные регулировочные параметры следующие:

$$\begin{aligned} \gamma_i &= \frac{\gamma_0}{\max_j \|x_i - x_j\|_2}, \text{ где } \gamma_0 = 0.8 \\ \alpha &= 0.01 \\ \beta_0 &= 1 \end{aligned} \tag{17.15}$$

Каждый светлячок x_i сравнивает свою яркость с каждым другим светлячком x_j по одному за раз. Если x_j ярче x_i, то x_i сделает ход, который включает оба компонента: случайный и тот, который направлен в сторону x_j. Величина αr на рис. 17.8 – это случайный компонент. Она обычно относительно мала в силу малого значения α (см. уравнение 17.15). Величина $\beta_0 e^{-\gamma_i r_{ij}^2}(x_i - x_j)$ – это направленный компонент; как указывалось ранее, его величина является экспоненциальной функцией расстояния r_{ij} между x_j и x_i. Хотя экспоненциальная функция биологически обусловлена, мы, возможно, захотим попробовать некоторые другие функции, которые уменьшаются с увеличением расстояния.

Одна вещь, которую мы заметили на рис. 17.8, – это то, что лучшая особь в популяции ни разу не обновляется. Мы вполне могли бы улучшить результативность алгоритма, если бы периодически обновляли лучшую особь для поиска более качественной особи. Однако этот подход, возможно, будет сопряжен с высоким риском и низкой отдачей, поскольку он может потребовать проведения многих оцениваний функ-

ции, прежде чем мы найдем место в поисковом пространстве, которое лучше, чем нынешняя лучшая позиция.

Инициализировать случайную популяцию $\{x_i\}$ для $i \in [1, N]$.
While not (критерий останова)
  For each особь x_i
    For each особь $x_j \neq x_i$
      If $f(x_j) < f(x_i)$
        For each размерность $k \in [1, n]$
          $\rho \leftarrow U[0, 1]$
          If $\rho < 1/2$
            $r_k \leftarrow (u_k - x_i(k))\, U[0, 1]$
          else
            $r_k \leftarrow (x_i(k) - l_k)\, U[0, 1]$
          End if
        Next размерность k
        $r_{ij} \leftarrow$ расстояние между x_i и x_j
        $x_i \leftarrow x_i + \beta_0 e^{-\gamma_i r_{ij}^2}(x_j - x_i) + \alpha r$ (это векторная операция)
      End if
    Next x_j
  Next x_i
Next поколение

Рис. 17.8. Светлячковый алгоритм для минимизации n-мерной функции $f(x)$. В данном алгоритме x_i – это i-е кандидатное решение и $x_i(k)$ – k-й элемент x_i. $U[0, 1]$ – это случайное число, равномерно распределенное на [0,1], а l_k и u_k – соответственно нижняя и верхняя границы k-й размерности поискового пространства

Дополнительные вариации светлячкого алгоритма обсуждаются в публикациях [Lukasik и Żak, 2009] , [Yang, 2009b], [Yang, 2010a]. Например, параметр a часто является убывающей функцией времени, которая служит для сокращения разведывания, по мере того как популяция становится более оптимизированной. Версия алгоритма для комбинаторных задач была предложена в публикации [Sayadi и соавт., 2010]. Светлячковый алгоритм наряду с алгоритмом искусственного косяка рыб (AFSA), описанным в разделе 17.2, очень похож на алгоритм оптимизации на основе роя частиц (PSO) (см. главу 11). Мы можем внести некоторые простые изменения в светлячковый алгоритм на рис. 17.8 и сделать его эквивалентным частному случаю алгоритма PSO (см. задачу 17.5).

17.6. Оптимизация на основе бактериальной кормодобычи

Алгоритм оптимизации на основе бактериальной кормодобычи (bacterial foraging optimization algorithm, BFOA) был описан в публикации [Passino, 2002] и основывается на поведении бактерий Escherichia coli, виде грамотрицательных палочковидных бактерий, широко известных как кишечная палочка (E. coli). Алгоритм оптимизации на основе бактериальной кормодобычи опирается на предпосылку, что естественный отбор способствует распространению генетической информации, которая дает возможность успешно осуществлять пищевое поведение. Кормодобыча, конечно же, распространена среди всех видов, а не только среди бактерий. Иногда животные добывают пищу сообща, а иногда в одиночку. Если они добывают пищу в одиночку, то у них есть преимущество в том, что они оставляют пищу, которую находят, целиком для себя. Но если они добывают пищу в командах, то у них есть преимущество в том, что они могут легче бороться с хищниками. Животные должны уравновешивать вероятность успеха кормодобычи с рисками со стороны хищников.

Если животное находит географический район с большим количеством пищи, то оно должно сбалансировать свою эксплуатацию этих известных ресурсов с возможностью разведать более богатые ресурсы в другом месте; это еще один тип поведения по балансированию риска. По мере того как животное истощает свои ресурсы в данном месте, появляется оптимальное время на то, чтобы оставить известные ресурсы ради поиска участков с большим количеством ресурсов.

Кормодобыча также включает в себя другое поведение, помимо поиска пищи. А именно включает в себя преследование и нападение на добычу, а также потребление добычи. Если добыча больше, чем хищник, то хищники должны координировать свои действия друг с другом, для того чтобы атаковать и потреблять добычу. Если добыча меньше хищника, то для хищника может быть оптимальнее кормиться самостоятельно. Некоторые кормодобытчики постоянно перемещаются по окружающей среде в поисках добычи. Другие кормодобытчики остаются скрытыми в неподвижном месте и ждут, когда добыча приблизится на расстояние удара. Иные практикуют сочетание этих двух подходов. Алгоритм оптимизации на основе бактериальной кормодобычи (BFOA) специально моделируется на основе кормодобывающего поведения бактерий, но теория кормодобычи является широко изучаемой дисци-

плиной [Stephens и Krebs, 1986], [Giraldeau и Caraco, 2000], которая имеет ряд потенциальных приложений в теории оптимизации [Quijano и соавт., 2006].

Алгоритм оптимизации на основе бактериальной кормодобычи опирается на три модели поведения бактерий. Во-первых, бактерии продвигаются по окружающей среде; такое поведение называется хемотаксисом. Во-вторых, бактерии размножаются. В-третьих, бактерии исключаются из среды обитания и рассеиваются по ней в результате воздействия окружающей среды.

Хемотаксис

Первое поведение бактерий, самодвижение, или хемотаксис, может быть далее разделено на два поведения. Во-первых, бактерии могут свободно падать в случайных направлениях. Во-вторых, они могут продвигаться в направлении увеличения кормовых ресурсов. На этот второй тип самодвижения оказывают влияние не только кормовые ресурсы, но и присутствие других бактерий. Другие бактерии служат как для притяжения, так и для отталкивания друг друга. Они имеют определенный уровень притяжения, потому что наличие бактерии в определенном месте означает, что в этом месте есть пища. А также определенный уровень отталкивания, поскольку наличие бактерии в определенном месте указывает на то, что в этом месте существует конкуренция за пищу.

Предположим, что мы хотим найти минимум функции $f(x)$. В алгоритме оптимизации на основе бактериальной кормодобычи (BFOA) самодвижущаяся бактерия моделируется как

$$x \leftarrow x + c\Delta, \tag{17.16}$$

где x – это местоположение в поисковом пространстве особи в популяции, c – размер шага, а Δ – единичный вектор в некотором направлении в поисковом пространстве. При свободном падении Δ является случайным единичным вектором. Сочетание притяжения и отталкивания другими бактериями приводит к эффективной функции стоимости $f'(x)$, воспринимаемой особью x следующим образом:

$$f'(x) = f(x) + \sum_{i=1}^{N} \left[h \exp\left(-w_r \| x - x_i \|_2^2\right) - d \exp\left(-w_\alpha \| x - x_i \|_2^2\right) \right], \tag{17.17}$$

где N – это размер популяции и h, w_r, d и w_α – регулировочные параметры, связанные с силами отталкивания и притяжения, которые бактерии оказывают друг на друга. Если особь x свободно падает в направлении,

которое уменьшает $f'(x)$, то она продолжает продвигаться в этом же направлении, хотя верхний предел N_s (еще один регулировочный параметр) помещается на количество движений в данном направлении.

Размножение

Каждая особь повторяет описанное выше самодвижение в течение N_c итераций. То есть она сначала свободно падает в случайном направлении. Если случайное падение уменьшает $f'(x)$, то она продолжает продвижение в этом же направлении. N_c движений определяют время жизни каждой бактерии. После этого здоровье каждой бактерии измеряется как среднее значение $f'(x)$ за предыдущие ЧПУ итераций. Наиболее здоровая половина бактерий размножается путем порождения двух клонов на каждую бактерию, таким образом обеспечивая N новых бактерий для следующего поколения.

Исключение и рассеивание

После размножения происходит этап исключения-рассеивания. Каждая бактерия рассеивается в случайном месте в поисковом пространстве с вероятностью p_e (еще один регулировочный параметр).

Резюме

На рис. 17.9 изображен базовый алгоритм оптимизации на основе бактериальной кормодобычи (BFOA). Типичные регулировочные параметры приведены ниже [Passino, 2002]:

$$\begin{aligned}
\text{размер шага } c &= 0.1\\
\text{размер популяции } N &= 50\\
\text{число шагов хемотаксиса } N_c &= 100\\
\text{число шагов сокращения стоимости } N_s &= 4\\
\text{число шагов размножения } N_r &= 4\\
\text{число шагов исключения-рассеивания } N_e &= 2\\
\text{глубина силы притяжения } d &= 1\\
\text{глубина силы отталкивания } h &= 1\\
\text{ширина силы притяжения } w_\alpha &= 0.2\\
\text{ширина силы отталкивания } w_r &= 10\\
\text{вероятность исключения-рассеивания } p_e &= 0.25
\end{aligned} \qquad (17.18)$$

Число поколений в алгоритме BFOA не так четко сформулировано, как в других эволюционных алгоритмах. Самый внешний цикл рис. 17.9 исполняется N_e раз, то есть, как правило, только два раза. Лучший спо-

соб измерить вычислительные усилия эволюционного алгоритма – использовать не поколения, а оценивания функций.

```
Инициализировать параметры, показанные в уравнении (17.18).
Инициализировать случайную популяцию {x_i} для i ∈ [1, N].
For l = 1 to N_e (шаги исключения-рассеивания)
  For k = 1 to N_r (шаги размножения)
    For j = 1 to N_c (шаги хемотаксиса)
      For each особь x_i, i ∈ [1, N]
        Вычислить эффективную стоимость, как показано в уравнении (17.17).
        Сгенерировать случайный n-мерный единичный вектор Δ.
        For m – 1 to N_s (шаги сокращения стоимости)
          x̂_i ← x_i + cΔ
          If f'(x̂_i) < f'(x_i) then
              x_i ← x̂_i
          else
              m ← N_s (покинуть цикл сокращения стоимости)
          End if
        Next m
      Next особь
    Next j
    For each особь x_i, i ∈ [1, N]
      F_i ← среднее значение функции f'(x_i) во время N_c шагов цикла хемотаксиса
    Next особь
    Исключить худших N/2 особей на основе {F_i}.
    Клонировать лучших N/2 особей на основе {F_i}.
  Next k
  For each особь x_i, i ∈ [1, N]
    Случайное число r ← U[0, 1]
    If r < p_e then
      x_i ← случайная точка в поисковом пространстве
    End if
  Next особь
Next l
```

Рис. 17.9. Алгоритм оптимизации на основе бактериальной кормодобычи (BFOA) для минимизации n-мерной функции $f(x)$, где x_i – это i-е кандидатное решение

По рис. 17.9 видно, что особи размножаются путем клонирования. Мы вполне могли бы улучшить результативность, используя более сложную операцию рекомбинации (см. раздел 8.8), даже если это приведет к отклонению от бактериальных основ алгоритма BFOA. Кроме того, мы

произвольно клонируем лучшую половину популяции на рис. 17.9, для того чтобы создать следующее поколение. Вместо этого мы могли бы клонировать лучших особей B, где B – это регулировочный параметр.

Алгоритм оптимизации на основе бактериальной кормодобычи (BFOA) представляет собой научную область с широким диапазоном возможностей. Существует ряд аспектов бактериальной кормодобычи и кормодобычи животных в целом, которые могут быть смоделированы для получения улучшенной результативности оптимизации. Для алгоритма BFOA автоматическая адаптация регулировочных параметров, возможно, имеет особое значение, поскольку параметров очень много, и она может дать улучшенную результативность [Dasgupta и соавт., 2009]. Алгоритм BFOA может быть гибридизирован с другими эволюционными алгоритмами, как это уже было с алгоритмами оптимизации на основе роя частиц (PSO) [Biswas и соавт., 2007b] и дифференциальной эволюции (DE) [Biswas и соавт., 2007a]. Небольшой математический анализ алгоритма BFOA приведен в публикации [Das и соавт., 2009], хотя многое еще предстоит сделать в этой области. Обратите внимание, что модель бактериального хемотаксиса представляет собой отдельный, но аналогичный эволюционному алгоритм, основанный на поведении бактерий [Muller и соавт., 2002].

17.7. Алгоритм искусственной пчелиной семьи

Алгоритм искусственный пчелиной семьи (artificial bee colony, ABC) основывается на поведении пчел и был впервые описан в публикации [Basturk и Karaboga, 2006], [Karaboga и Basturk, 2007]. Алгоритм искусственный пчелиной семьи опирается на поиск пчелами оптимального источника пищи. Расположение источника пищи аналогично расположению в поисковом пространстве оптимизационной задачи. Количество нектара в том или ином месте аналогично приспособленности кандидатного решения. Данный алгоритм симулирует три разных типа пчел.

Во-первых, пчелы-собиратели, так называемые рабочие пчелы, летают туда и обратно между источником пищи и ульем. Каждый собиратель связан с определенным местоположением и запоминает его при перемещении туда-сюда от улья. Когда собиратель приносит нектар в улей, он возвращается к источнику пищи, а также участвует в локальной разведке местности, так как ищет поблизости более богатый источник.

Во-вторых, пчелы-наблюдатели, которые не связаны с каким-либо конкретным источником пищи, но наблюдают за поведением собира-

телей, когда они возвращаются в улей. Наблюдатели следят за количеством нектара, возвращаемого собирателями (то есть за приспособленностью расположения каждого из них в поисковом пространстве) и используют эту информацию, для того чтобы принять решение, где искать нектар. Место поиска наблюдателей определяется вероятностно на основе их наблюдений за собирателями.

В-третьих, пчелы-разведчики, которые являются исследователями и, как наблюдатели, не связаны с каким-либо конкретным источником пищи. Если разведчик видит, что собиратель застаивается и поступательно не увеличивает количество нектара, которое он возвращает в улей, то разведчик случайно ищет новый источник нектара в поисковом пространстве. Застой появляется, когда собиратель не в состоянии увеличить количество нектара, который он приносит в улей после определенного числа полетов.

Данные идеи приводят к алгоритму искусственной пчелиной семьи (ABC), который обобщен на рис. 17.10. Цифры показывают, что разделение между пчелами-собирателями, наблюдателями и разведчиками – это просто аналогия, и на ней в алгоритме ABC не делается какого-то особого упора. Ключевая идея алгоритма ABC заключается в том, что в поисках глобального оптимума симулируется поведение сбора, наблюдения и разведывания.

N = размер популяции
Инициализировать положительное число L, то есть лимит стагнации.
Инициализировать размер популяции собирателей $P_f < N$.
Инициализировать размер популяции наблюдателей $P_o = N - P_f$.
Инициализировать случайную популяцию собирателей $\{x_i\}$ для $i \in [1, P_f]$.
Инициализировать числа попыток собирателей $T(x_i) = 0$ для $i \in [1, P_f]$.
While not (критерий останова)
  Пчелы-собиратели:
  For each собиратель x_i, $i \in [1, P_f]$
    $k \leftarrow$ случайное целое $\in [1, N]$ такое, что $k \neq i$
    $s \leftarrow$ случайное целое $\in [1, n]$
    $r \leftarrow U[-1, 1]$
    $v_i(s) \leftarrow x_i(s) + r(x_i(s) - x_k(s))$
    If $f(v_i)$ лучше $f(x_i)$ **then**
      $x_i \leftarrow v_i$
      $T(x_i) \leftarrow 0$
    else
      $T(x_i) \leftarrow T(x_i) + 1$

```
    End if
  Next собиратель
  Пчелы-наблюдатели:
  For each наблюдатель v_i, i ∈ [1, P_f]
    Отобрать собирателя x_j, где Pr(x_j) ∝ приспособленность(x_j) для j ∈ [1, P_f].
    k ← случайное целое ∈ [1, P_f] такое, что k ≠ j
    s ← случайное целое ∈ [1, n]
    r ← U[–1, 1]
    v_i(s) ← x_j(s) + r(x_j(s) – x_k(s))
    If f(v_i) лучше f(x_j) then
        x_j ← v_i
        T(x_j) ← 0
    else
        T(x_j) ← T(x_j) + 1
    End if
  Next наблюдатель
  Пчелы-разведчики:
  For each собиратель x_i, i ∈ [1, P_f]
    If T(x_i) > L then
        x_i ← случайно сгенерированная особь
        T(x_i) ← 0
    End if
  Next собиратель
Next поколение
```

Рис. 17.10. Алгоритм искусственной пчелиной семьи (ABC) для оптимизации n-мерной функции $f(x)$, где x_i – это i-е кандидатной решение

На рис. 17.10 показано, что каждый собиратель случайно изменяет свое позицию в поисковом пространстве. Если случайная модификация приводит к улучшению, то собиратель переходит на новую позицию. Пчелы-наблюдатели также случайно изменяют позицию собирателя, где модифицируемый собиратель отбирается случайно с помощью рулеточного отбора. Опять же, если случайная модификация улучшает собирателя, то собиратель переходит на новую позицию. Наконец, разведчик заменяет собирателя, если он не улучшился после заданного числа случайных модификаций. Числа $T(x_i)$ на рис. 17.10 – это числа попыток собирателей, которые отслеживают, сколько неудачных модификаций каждого собирателя было выполнено подряд. Рисунок 17.10 показывает, что алгоритм ABC включает в себя несколько регулировочных параметров. Типичными параметрами алгоритма ABC являются

$$P_f = P_o = N/2, \text{ лимит стагнации } L = Nn/2. \quad (17.19)$$

В литературе обсуждается несколько вариантов алгоритма искусственной пчелиной семьи (ABC) [Karaboga и Basturk, 2007], [Karaboga и Basturk, 2008], [Karaboga и Akay, 2009], [Karaboga и и соавт., 2013], и, изучив рис. 17.10, мы могли бы подумать о внесении в данный алгоритм многочисленных изменений. Например, на рис. 17.10 показано, что каждый собиратель обновляет свою позицию детерминистически, если находит лучшую позицию; вместо этого мы могли бы сделать это обновление стохастическим. На рис. 17.10 показано также, что наблюдатели выбирают, за каким собирателем следить, на основе рулеточного отбора; мы могли бы применить другой, основанный на приспособленности метод отбора (см. раздел 8.7).

Было предложено несколько иных алгоритмов, похожих на алгоритм ABC. Обратитесь к публикациям [Tereshko, 2000], [Teodorović, 2003], [Benatchba и соавт., 2005], [Wedde и соавт., 2004] и ссылкам в публикации [Karaboga и Basturk, 2008]. Одним из алгоритмов, очень похожих на алгоритм ABC, является пчелиный алгоритм (bees algorithm) [Pham и соавт., 2006]. Эволюционный алгоритм в этой книге с искусственной пчелиной семьей, по всей видимости, имеет больше всего общего с дифференциальной эволюцией (DE) (см. главу 12). Будущие исследования алгоритма на основе искусственной пчелиной семьи (ABC) могли бы включать в себя изучение его общности с дифференциальной эволюцией, его общности с другими алгоритмами, ориентированными на пчел, такими как упомянутые выше, и встраивание большего числа биологически обусловленных признаков.

17.8. Алгоритм гравитационного поиска

Алгоритм гравитационного поиска (gravitational search algorithm, GSA) был введен в публикации [Rashedi и соавт. 2009] и основывается на законе гравитации. Алгоритм гравитационного поиска подобен оптимизации на основе центральной силы (central force optimization), детерминированном эволюционном алгоритме оптимизации, который также основывается на гравитации [Formato, 2007], [Formato, 2008]. Другие аналогичные алгоритмы включают космическую гравитационную оптимизацию [Flsiao и соавт., 2005] и интегральную радиационную оптимизацию [Chuang and Jiang, 2007]. Алгоритм гравитационного поиска похож на оптимизацию на основе роя частиц и работает по принципу, что каждая особь в популяции имеет положение и скорость в поисковом

пространстве, но она также включает ускорение. Частицы притягиваются друг к другу на основе их значений массы, которые пропорциональны их значениям приспособленности (то есть обратно пропорциональны их стоимостным значениям). На рис. 17.11 показан алгоритм гравитационного поиска (GSA).

Инициализировать случайную популяцию особей $\{x_i\}$ для $i \in [1, N]$.
Инициализировать скорость каждой особи v_i, $i \in [1, N]$.
Инициализировать гравитационную константу G_0 и скорость затухания α.
Инициализировать номер поколения $t = 0$ лимит поколений t_{max}.
While not (критерий останова)
  Гравитационная константа $G \leftarrow G_0 \exp(-\alpha t/t_{max})$.
  For each особь x_i, $i \in [1, N]$
    $m_i \leftarrow \frac{f(x_i) - \max_k f(x_k)}{\min_k f(x_k) - \max_k f(x_k)} \in [0, 1]$
    Нормализованная приспособленность $M_i \leftarrow \frac{m_i}{\sum_{k=1}^{N} m_k}$.
  Next особь
  For each особь x_i, $i \in [1, N]$
    Расстояние $R_{ik} \leftarrow \|x_k - x_i\|_2$ для $k \in [1, N]$.
    Вектор силы $F_{ik} \leftarrow \frac{GM_iM_k}{R_{ik} + \varepsilon} (x_k - x_i)$ для $k \in [1, N]$.
    Случайное число $r_k \leftarrow U[0, 1]$ для $k \in [1, N]$.
    Вектор ускорения $\alpha_i \leftarrow 1/M_i \sum_{k=1, k \neq i}^{N} r_k F_{ik}$.
    Случайное число $r \leftarrow U[0, 1]$.
    Вектор скорости $v_i \leftarrow rv_i + \alpha_i$.
    Вектор положения $x_i \leftarrow x_i + v_i$.
  Next особь
  Нарастить номер поколения: $t \leftarrow t + 1$.
Next поколение

Рис. 17.11. Алгоритм гравитационного поиска (GSA) для минимизации $f(x)$, где x_i – это i-е кандидатное решение, ϵ – малая положительная постоянная для предотвращения деления на ноль

В цикле поколений на рис. 17.11 мы сначала обновляем значение гравитационной постоянной G. Временна́я инвариантность G в природе является предметом дискуссии, причем некоторые физики утверждают, что она меняется во времени [Jofré и соавт., 2006]. Постепенное сокращение G в алгоритме GSA с течением времени уменьшает разведывательный компонент данного алгоритма. Далее мы устанавливаем значения приспособленности так, чтобы худшая особь имела приспособленность $m_i = 0$ и лучшая особь имела приспособленность $m_i = 1$; значения приспо-

собленности соответствуют гравитационным массам. Затем получаем нормализованные значения приспособленности $\{M_i\}$, которые в сумме составляют 1. Далее по каждой паре особей вычисляется сила притяжения, то есть вектор, величина которого пропорциональна значениям их приспособленности и расстояниям между ними. Потом мы берем случайную комбинацию векторов сил, чтобы получить вектор ускорения каждой особи. Наконец, используем вектор ускорения для обновления скорости и положения каждой особи. Типичными регулировочными значениями на рис. 17.11 являются $G_0 = 100$ и $\alpha = 20$.

Исследователи предлагают различные модификации и расширения алгоритма GSA, включая альтернативные способы регулировки G каждое поколение. Уравнение ускорения может обновляться так, что только лучшие особи притягивают каждую частицу:

$$\alpha_i \leftarrow \frac{1}{M_i} \sum_{k \in B, k \neq i} r_k F_{ik}, \tag{17.20}$$

где B – это множество лучших особей, а размер множества B – регулировочный параметр. Вдобавок мы можем использовать разные типы значений эффективной массы для активной гравитационной силы, пассивной гравитационной силы и инерции [Rashedi и соавт., 2009]. Расширение алгоритма GSA на дискретные поисковые области приведено в публикации [Rashedi и соавт., 2010]. Учитывая сходство между алгоритмом гравитационного поиска (GSA) и алгоритмом оптимизации на основе роя частиц (PSO), представляется, что многие из расширений, предложенных для алгоритма PSO, могут быть также реализованы в алгоритме GSA (см. главу 11).

17.9. Поиск гармонии

Поиск гармонии (harmony search, HS) был представлен в публикации [Geem и соавт., 2001] и более подробно объясняется в публикации [Lee и Geem, 2006]. Поиск гармонии основывается на музыкальных процессах. Каждый музыкант в хоре или группе издает ноту в пределах некоторого допустимого диапазона. Если все ноты согласуются в гармонии, то в коллективной памяти хора сохраняется позитивный опыт, и возможность достижения крепкой гармонии увеличивается. В алгоритме поиска гармонии хор или группа аналогичны кандидатному решению задачи, а музыкант аналогичен независимой переменной или признаку кандидатного решения.

На рис. 17.12 представлен алгоритм поиска гармонии [Omran и Mahdavi, 2008]. В алгоритме поиска гармонии часто используются альтернативные обозначения, а не стандартные обозначения эволюционного алгоритма. Например, *вектор гармонии* используется для обозначения эволюционно-алгоритмической особи или кандидатного решения x, размер *гармонической памяти* используется для обозначения размера N популяции, *скорость анализа гармонической памяти* (harmony memory considering rate) похожа на скорость скрещивания c в генетических алгоритмах, *скорость корректировки высоты звучания* используется для обозначения скорости мутации p_m и *дистанционный интервал* (distance bandwidth) используется для обозначения стандартного отклонения σ гауссовой мутации. Типичные значения для этих параметров: $c = 0.9$; p_m увеличивается линейно с 0.01 в первом поколении до 0.99 в последнем поколении; и σ уменьшается экспоненциально от 5 % поисковой области до 0.01 % поисковой области.

p_m = скорость мутации $\in$ [0, 1]
σ^2 = гаусова мутационная дисперсия
c = скорость мутации $\in$ [0, 1]
Инициализировать популяцию кандидатных решений $\{x_k\}$ для $k \in [1, N]$.
While not (критерий останова)
  Ребенок – [0 0 ... 0] $\in R^n$
  For each признак $s = 1, ..., n$ решения
    $r_c \leftarrow U[0, 1]$
    If $r_c < c$ **then**
      $j \leftarrow$ случайное целое $\in [1, n]$
      Ребенок(s) $\leftarrow x_j(s)$
      $r_m \leftarrow U[0, 1]$
      If $r_m < p_m$ **then**
        Ребенок(s) $\leftarrow$ Ребенок(s) + $N(0, \sigma^2)$
      End if
    else
      Ребенок(s) $\leftarrow U[x_{min}(s), x_{max}(s)]$
    End if
  Next Признак решения
  $m \leftarrow \arg\max_k (f(x_k) : k \in [1, N])$
  If f(Ребенок) < $f(x_m)$ **then**
    $x_m \leftarrow$ Ребенок
  End if
Next поколение

Рис. 17.12. Схематичное описание алгоритма поиска гармонии (HS) с популяцией размером N для минимизации n-мерной функции $f(x)$. $\{x_k\}$ – это вся популяция особей в целом, x_k – значение k-й особи и $x_k(s)$ – s-й признак x_k

По рис. 17.12 видно, что алгоритм поиска гармонии создает по одному ребенку в каждом поколении. Для каждого признака решения генерируется случайное число r_c. Если r_c меньше скорости скрещивания c, то этот признак решения в ребенке принимается равным случайно отобранному признаку решения из популяции; этот шаг аналогичен глобальной равномерной рекомбинации (см. раздел 8.8.6). Однако если r_m больше скорости скрещивания, то этот признак решения в ребенке принимается равным случайному числу в поисковой области; этот шаг аналогичен равномерной мутации, центрированной в поисковой области (см. раздел 8.9.2). Если детский признак решения получается из популяции, а не из случайного числа, то мы выполняем гауссову мутацию, центрированную на признаке решения (см. раздел 8.9.3). Наконец, если ребенок лучше, чем худшая особь в популяции, то ребенок заменяет эту особь в популяции; этот последний шаг является той же стратегией, которую мы используем в эволюционном программировании (см. раздел 5.1).

Таким образом, как представляется, в алгоритме поиска гармонии нет принципиально новых идей. Алгоритм поиска гармонии – это объединение ранее установленных идей эволюционного алгоритма, включая глобальную равномерную рекомбинацию, равномерную мутацию, гауссову мутацию и замену худшей особи каждое поколение. Вклад алгоритма поиска гармонии заключается в двух областях. Во-первых, то, как данный алгоритм сочетает в себе эти идеи, имеет новизну. Во-вторых, музыкальная мотивация алгоритма поиска гармонии является новой. Вместе с тем лишь в очень немногих публикациях по поиску гармонии обсуждаются музыкальные мотивы или расширения алгоритма поиска гармонии. Большинство публикаций посвящено гибридизации алгоритма поиска гармонии с другими эволюционными алгоритмами, регулировке параметров поиска гармонии либо применению алгоритма поиска гармонии к конкретным задачам. Если бы к алгоритму поиска гармонии можно было бы применить более музыкально обусловленные расширения, то это помогло бы выделить его как отдельный эволюционный алгоритм. Такие исследования потребовали бы изучения теории музыки, изучения процесса музыкальной композиции и аранжировки, изучения образовательных теорий музыки и изобретательного применения этих теорий в алгоритме поиска гармонии. Дополнительные материалы для чтения в области алгоритма поиска гармонии можно найти в двух монографиях [Geem, 2010a], [Geem, 2010b] и одной книге [Geem, 2010c].

17.10. Оптимизация по принципу учитель–ученик

Оптимизация по принципу учитель–ученик (teaching-learning-based optimization, TLBO) была введена в публикации [Rao и соавт., 2011] и более подробно объясняется в публикациях [Rao и соавт., 2012], [Rao и Patel, 2012], [Rao и Savsani, 2012, глава 6]. Оптимизация по принципу учитель–ученик основывается на процессе передачи и получения знаний в классе. В каждом поколении лучшее кандидатное решение в популяции рассматривается как учитель, а другие кандидатные решения рассматриваются как ученики. Ученики в основном принимают инструкции от учителя, но также учатся друг у друга. В алгоритме оптимизации по принципу учитель–ученик учебный предмет аналогичен независимой переменной или признаку кандидатного решения. Фаза учителя состоит из модификации каждой независимой переменной $x_i(s)$ в каждом кандидатном решении x_i следующим образом:

$$\begin{aligned} c_i(s) &= x_i(s) + r(x_t(s) - T_f \bar{x}(s)) \\ \bar{x}(s) &= \frac{1}{N}\sum_{k=1}^{N} x_k(s) \end{aligned} \tag{17.21}$$

для $i \in [1, N]$ и $s \in [1, n]$, где N – это размер популяции, n – размерность задачи, x_t – лучшая особь в популяции (то есть учитель), r – случайное число, взятое из равномерного распределения на [0, 1], T_f называется обучающим фактором (teaching factor) и устанавливается равным либо 1, либо 2 с равной вероятностью. Ребенок c_i заменяет родителя x_i, если ребенок лучше родителя. В общем случае уравнение (17.21) корректирует $x_i(s)$ в направлении к лучшей особи $x_t(s)$. Мы можем видеть это, взяв ожидаемое значение (математическое ожидание) уравнения (17.21), которое дает

$$\begin{aligned} \bar{c}_i(s) &= \bar{x}(s) + \frac{1}{2}\left(x_t(s) - \frac{3}{2}\bar{x}(s)\right) \\ &= \frac{x_t(s)}{2} + \frac{\bar{x}(s)}{4}. \end{aligned} \tag{17.22}$$

То есть в среднем $c_i(s)$ ближе к $x_t(s)$, чем к $x_i(s)$. Уравнение (17.22) также показывает, что $c_i(s)$ в среднем ближе к нулю, чем родительская популяция, что может дать алгоритму оптимизации по принципу учитель–ученик (TLBO) несправедливое преимущество на задачах, решение ко-

торых равно нулю. Многие оптимизационные эталоны имеют решения при $x^* = 0$, поэтому в дальнейших исследованиях алгоритма TLBO должна быть тщательно исследована его результативность на задачах с ненулевыми решениями и по мере необходимости сделана корректировка алгоритма, с тем чтобы устранить это присущее ему смещение.

После завершения фазы учителя начинается фаза ученика. Фаза ученика влечет за собой корректировку каждой особи на основе другой случайно отобранной особи:

$$c_i(s) \leftarrow \begin{cases} x_i(s) + r(x_i(s) - x_k(s)) & \text{если } x_i \text{ лучше чем } x_k \\ x_i(s) + r(x_k(s) - x_i(s)) & \text{в противном случае} \end{cases} \tag{17.23}$$

для $i \in [1, N]$ и $s \in [1, n]$, где k – это случайное целое число в $[1, N]$ – такое, что $k \neq i$, и r – случайное число, взятое из равномерного распределения на [0, 1]. Фаза ученика корректирует $x_i(s)$ в сторону прочь от x_k, если x_k хуже (в первом приведенном выше случае), и в сторону к x_k, если x_k лучше (во втором случае).

На рис. 17.13 схематично представлен алгоритм оптимизации по принципу учитель–ученик (TLBO). Осмотр рис. 12.2 показывает, что из всех эволюционных алгоритмов, обсуждаемых в этой книге, алгоритм TLBO имеет больше всего общего с дифференциальной эволюцией (DE). Предположим, что на рис. 12.2 установлена скорость скрещивания $c = 1$. Далее предположим, что вместо константного параметра размера шага F мы используем случайный параметр размера шага, который отличается для каждой независимой переменной, так что одна независимая переменная за раз определяется в мутантном векторе v. Затем предположим, что мы заменим каждую особь x_i ее ребенком сразу после создания ребенка, а не будем ждать до тех пор, пока создадим всех детей. Эти изменения в алгоритме дифференциальной эволюции не обязательно незначительные, но они прямолинейны. С этими изменениями алгоритм дифференциальной эволюции рис. 12.2 становится модифицированным алгоритмом дифференциальной эволюции рис. 17.14.

Инициализировать популяцию кандидатных решений $\{x_k\}$ для $k \in [1, N]$.
While not (критерий останова)
  For each особь x_i где $i \in [1, N]$
    Комментарий: фаза учителя
    $x_t \leftarrow \arg\min_x(f(x) : x \in \{x_k\}_{k=1}^N)$
    $T_f \leftarrow$ случайное целое $\in \{1, 2\}$
    For each признак $s \in [1, n]$ решения

$\bar{x}(s) \leftarrow 1/N \sum_{k=1}^{N} x_k(s)$
$r \leftarrow U[0, 1]$
$c_i(s) \leftarrow x_i(s) + r(x_t(s) - T_f \bar{x}(s))$
Next признак решения
$x_i \leftarrow \arg\min_{x_i, c_i}(f(x_i), f(c_i))$
Комментарий: фаза ученика
$k \leftarrow$ случайное целое $\in [1, N] : k \neq i$
If $f(x_i) < f(x_k)$ **then**
For each признак $s \in [1, n]$ решения
$r \leftarrow U[0, 1]$
$c_i(s) \leftarrow x_i(s) + r(x_i(s) - x_k(s))$
Next признак решения
else
For each признак $s \in [1, n]$ решения
$r \leftarrow U[0, 1]$
$c_i(s) \leftarrow x_i(s) + r(x_k(s) - x_i(s))$
Next признак решения
End if
$x_i \leftarrow \arg\min_{x_i, c_i}(f(x_i), f(c_i))$
Next особь
Next поколение

Рис. 17.13. Схематичное описание алгоритма оптимизации по принципу учитель–ученик (TLBO) с популяцией размером N для минимизации n-мерной функции $f(x)$. x_t – это лучшая особь в популяции, и она называется учителем

Теперь предположим, что вместо случайного генерирования r_1, r_2 и r_3 на рис. 17.14 мы задаем их следующим образом:

$$\begin{aligned} r_1 &= i \\ r_2 &= \arg\min_i \{f(x_i): i \in [1, N]\} \\ x_{r3}(s) &= \text{среднее значение } s\text{-го признака популяции} \end{aligned} \tag{17.24}$$

Инициализировать популяцию кандидатных решений $\{x_i\}$ для $i \in [1, N]$.
While not (критерий останова)
For each особь x_i, $i \in [1, N]$
$r_1 \leftarrow$ случайное целое $\in [1, N] : r_1 \neq i$
$r_2 \leftarrow$ случайное целое $\in [1, N] : r_2 \notin \{i, r_1\}$
$r_3 \leftarrow$ случайное целое $\in [1, N] : r_3 \notin \{i, r_1, r_2\}$
For each признак $s \in [1, n]$ решения
$r \leftarrow U[0, 1]$

$v(s) \leftarrow x_{r1}(s) + r(x_{r2}(s) - x_{r3}(s))$
Next признак решения
$x_i \leftarrow \arg\min_{x_i, v}[f(x_i), f(v)]$
Next особь
Next поколение

Рис. 17.14. Модифицированный алгоритм дифференциальной эволюции (DE) для минимизации $f(x)$

С этими дополнительными изменениями алгоритм DE рис. 17.14 становится алгоритмом рис. 17.15, который эквивалентен фазе учителя алгоритма TLBO на рис. 17.13 с $T_f = 1$.

Аналогичным образом предположим, что вместо случайного генерирования r_1, r_2 и r_3 на рис. 17.14 мы устанавливаем их следующим образом:

$$\begin{aligned} r_1 &= i \\ r_2 &= \arg\min_{i,k}\{f(x_i), f(x_k)\} \\ r_3 &= \arg\max_{i,k}\{f(x_i), f(x_k)\} \end{aligned} \tag{17.25}$$

где k – это случайное целое число $\in [1, N]$ – такое, что $k \neq i$. Тогда мы получаем фазу обучения алгоритма TLBO.

Таким образом, оказывается, что в алгоритме TLBO нет принципиально новых идей. Алгоритм TLBO является модификацией алгоритма DE, который сам по себе является вариацией генетического алгоритма (см. раздел 12.4). Вклад алгоритма TLBO заключается в исполнении алгоритма DE в двух разных фазах, одна называется фазой учителя, другая – фазой ученика. Кроме того, мотивация взаимоотношения «учитель–ученик» алгоритма TLBO является новинкой. Однако публикации по алгоритму TLBO на данный момент не эксплуатируют методики преподавания и познания с целью улучшения алгоритма TLBO. Если бы к алгоритму TLBO можно было применить больше расширений, основанных на взаимоотношении «учитель–ученик», то это помогло бы выделить его в качестве отдельного эволюционного алгоритма, а также обеспечило бы потенциал для повышения его результативности. Такие исследования потребуют изучения теории познания, стилей преподавания и стилей познания, а также изобретательного применения этих теорий к алгоритму TLBO. Еще одной важной темой исследований в алгоритме TLBO, как упоминалось ранее, является изучение его результативности на задачах, решения которых не находятся в центре по-

исковой области, а настройка алгоритма для удаления его склонности находится в центре области. За дополнительными критическими соображениями в отношении алгоритма TLBO обратитесь к публикации [Crepinsek и соавт., 2012] и просмотрите публикацию [Waghmare, 2013] относительно ответов на критику.

Инициализировать популяцию кандидатных решений $\{x_i\}$ для $i \in [1, N]$.
While not (критерий останова)
  For each особь x_i, $i \in [1, N]$
    $x_t \leftarrow \arg\min_x (f(x) : x \in \{x_k\}_{k=1}^N)$
    For each признак $s \in [1, n]$ решения
      $\bar{x}(s) \leftarrow 1/N \sum_{k=1}^N x_k(s)$
      $r \leftarrow U[0, 1]$
      $v(s) \leftarrow x_i(s) + r(x_t(s) - \bar{x}(s))$
    Next признак решения
    $x_i \leftarrow \arg\min [f(x_i), f(v)]$
  Next особь
Next поколение

Рис. 17.15. Еще один модифицированный алгоритм дифференциальной эволюции (DE) для минимизации $f(x)$. Данный алгоритм эквивалентен фазе учителя оптимизации по принципу учитель–ученик (TLBO)

17.11. Заключение

Со времен первого генетического алгоритма Нильса Барричелли (Nils Barricelli) в 1953 году исследователи предложили целый ряд эволюционных алгоритмов [Dyson, 1998, стр. 111]. По всей видимости, практически любой естественный процесс можно интерпретировать как оптимизационный алгоритм [Alexander, 1996]. В данной главе и в других местах этой книги мы увидели, что многие из этих процессов оптимизации имеют сходные алгоритмические признаки. Поэтому трудно понять, где заканчивается один эволюционный алгоритм и начинается другой. Когда новый эволюционный алгоритм принадлежит своему собственному отдельному классу, и когда его следует классифицировать как вариацию существующего? Одна из задач научного сообщества – найти этот баланс и стимулировать новые исследования, сохраняя при этом высокие стандарты внедрения и разработки предположительно новых алгоритмов.

В этой главе мы охватили несколько дополнительных эволюционных алгоритмов. Некоторые из них популярны и полезны, но не подходят

для других мест в этой книге. Другие являются относительно новыми, и уровень их внедрения инженерами и компьютерными учеными еще предстоит определить. Существует ряд иных эволюционных алгоритмов, которые мы не успели обсудить, некоторые приведены ниже:

- алгоритм общества и цивилизации (Society and civilization algorithm) [Ray и Liew, 2003];
- поисковый алгоритм на основе заряженной системы (Charged system search) [Kaveh и Talatahari, 2010];
- оптимизация на основе инвазивных сорняков (Invasive weed optimization) [Mehrabian and Lucas, 2006];
- поисковый алгоритм кукушки (Cuckoo search) [Yang, 2009a];
- интеллектуальные капли воды (Intelligent water drops) [Shah-Hosseini, 2007];
- динамика формирования рек (River formation dynamics) [Rabanal и соавт., 2007];
- поисковый алгоритм на основе стохастической диффузии (Stochastic diffusion) [Bishop, 1989];
- гауссова адаптация (Gaussian adaptation) [Kjellstrom, 1969];
- алгоритм Большого взрыва и Большого схлопывания (Big bang big crunch algorithm) [Erol и Eksin, 2006];
- империалистический конкурентный алгоритм (Imperialist competitive algorithm) [Atashpaz-Gargari и Lucas, 2007];
- оптимизация на основе скрипучего колеса (Squeaky wheel optimization) [Joslin и Clements, 1999];
- грамматическая эволюция (Grammatical evolution) [O'Neill и Ryan, 2003];
- оптимизация на основе роя светлячков (Glowworm swarm optimization) [Krishnanand и Ghose, 2009];
- оптимизация на основе химических реакций (Chemical reaction optimization) [Lam и Li, 2010];
- стадо криля (Krill herd) [Gandomi и Alavi, 2012];
- алгоритм на основе летучей мыши (Bat-inspired algorithm) [Yang, 2010c];
- принятие порога (Threshold accepting) [Dueck и Scheuer, 1990];
- алгоритм Великого потопа и путешествия от записи к записи (Great deluge algorithm and record-to-record travel) [Dueck, 1993];

- модель бактериального хемотаксиса (Bacterial chemotaxis model) [Muller и соавт., 2002], которую мы также кратко упомянули в конце раздела 17.6;
- несколько алгоритмов искусственной пчелы, которые кратко упомянули в разделе 17.7;
- несколько гравитационных и силовых алгоритмов, которые кратко упомянуты в разделе 17.8.

Без всякого сомнения, существуют и другие эволюционные алгоритмы, которые принадлежат к вышеуказанному списку или заслуживают дальнейшего обсуждения и опущены только из-за недостаточной осведомленности автора. В приведенном выше списке алгоритмов, наряду с теми, которые обсуждались в предыдущих разделах этой главы, может хватить заинтересованным студентам и исследователям на целую продуктивную жизнь. Существуют и другие вычислительные методы, которые обычно не классифицируются как эволюционные алгоритмы, однако иногда разделительная линия между машинным самообучением и оптимизацией нечеткая. Вот почему в этой книге мы не обсуждаем алгоритмы, такие как нейронные сети [Fausett, 1994], нечеткая логика [Ross, 2010], искусственные иммунные системы [Hofmeyr и Forrest, 2000], искусственная жизнь [Adami, 1997], мембранные вычисления [Păun, 2003] и многие другие вычислительные парадигмы.

Задачи

Письменные упражнения

17.1. Перепишите определение стагнации в уравнении (17.4) для случая, когда оптимизационная задача является задачей максимизации.

17.2. Напишите алгоритм, который проще, но функционально эквивалентен прыжковому поведению искусственных рыб рис. 17.3.

17.3. Является ли оптимизатор группового поиска элитарным?

17.4. Сколько оцениваний функции выполняет алгоритм перемешанных лягушачьих прыжков в каждом поколении?

17.5. Приведите какие-то конкретные условия, при которых светлячковый алгоритм рис. 17.8 можно считать особым случаем, либо обобщением, алгоритма оптимизации на основе роя частиц на рис. 11.1.

17.6. В разделе 8.7 обсуждается несколько вариантов отбора для эволюционных алгоритмов. Какой тип отбора используется в оптимизации на основе бактериальной кормодобычи?

17.7. Напишите псевдокод, показывающий, как можно создать случайный единичный вектор для оптимизации на основе бактериальной кормодобычи.

17.8. Приведите какие-то конкретные условия, при которых алгоритм искусственной пчелиной семьи рис. 17.10 может считаться особым случаем, либо обобщением, алгоритма дифференциальной эволюции рис. 12.2.

17.9. Приведите какие-то конкретные условия, при которых алгоритм гравитационного поиска рис. 17.11 можно считать особым случаем, либо обобщением, алгоритма оптимизации на основе роя частиц на рис. 11.1.

17.10. Какова вероятность того, что конкретный ребенок в алгоритме поиска гармонии целиком состоит из ранее существовавших, не мутировавших признаков решения из родительской популяции?

Компьютерные упражнения

17.11. Просимулируйте один из эволюционных алгоритмов в этой главе. Проварьируйте несколько параметров, для того чтобы увидеть, какое влияние они оказывают на результативность оптимизации.

17.12. Алгоритм оптимизации по принципу учитель–ученик (TLBO), показанный на рис. 17.13, имеет фазу учителя и фазу ученика. Напишите симуляции для трех вариантов алгоритма TLBO: первый вариант – это оригинальный алгоритм, как показано на рис. 17.13, второй вариант – с использованием только фазы учителя, и третий вариант – с использованием только фазы ученика. Оптимизируйте 10-мерную функцию Экли с популяцией размером 100 и лимитом оценивания функции 10 000. Сообщите о лучшей результативности, полученной тремя вариациями алгоритма TLBO, усредненно по 20 симуляциям Монте-Карло. Повторите для 10-мерной функции Розенброка. Прокомментируйте свои результаты.

Часть IV

Специальные типы оптимизационных задач

Глава 18

Комбинаторная оптимизация

Мы обозначили как задачу посыльного (так как эта задача встречается у каждого разносчика почты, а также у многих путешественников) задачу нахождения конечного числа точек, попарные расстояния которых известны, задачу кратчайшего пути, соединяющего точки.

– Карл Менгер [Gutin и Punnen, 2007, стр. 1]

До сих пор в этой книге подчеркивались непрерывные оптимизационные задачи. В данной главе обсуждаются дискретные оптимизационные задачи: $\min_x f(x)$, где область x дискретна. Дискретная оптимизационная задача, так называемая комбинаторная оптимизационная задача, может рассматриваться как нахождение оптимального объекта из конечного множества кандидатных объектов:

$$\min_x f(x), \quad \text{где } x \in \{x_1, x_2, \ldots, x_{N_x}\}. \tag{18.1}$$

N_x называется мощностью поискового пространства. Теоретически мы могли бы решить уравнение (18.1), вычислив $f(x)$ для всех N_x возможных решений. Подобный подход к комбинаторной оптимизации называется исчерпывающим поиском, или грубой силой. Однако комбинаторные задачи часто имеют такое большое поисковое пространство, что невозможно проверить каждое возможное решение.

Задача о маршруте шахматного коня, или о ходе коня, – это классическая комбинаторно-оптимизационная задача. Данная задача впервые обсуждалась с математической точки зрения Леонардом Эйлером в 1759 году [Ball и Coxeter, 2010]. Как коню пройти по пустой шахматной доске, чтобы посетить каждую клетку ровно один раз? Замкнутый маршрут (так называемый повторяющийся маршрут) – это маршрут, в

котором конь заканчивает свой маршрут в той же клетке, в которой он начал, и, таким образом, требует 64 хода; в противном случае маршрут незамкнут и, следовательно, требует 63 хода.

Задача о незамкнутом маршруте шахматного коня может быть поставлена как $\min_x f(x)$, где x состоит из исходной позиции и последовательности 63 ходов и $f(x)$ является мерой того, сколько клеток конь не смог посетить. Мощность N_x области x (то есть размер поискового пространства) составляет более 3.3×10^{13} [Löbbing и Wegener, 1995]. Нам бы не хотелось пытаться решать эту задачу грубой силой, а вот используя человеческую проницательность и изобретательность, мы можем решить задачу о маршруте шахматного коня без особых затруднений. Мы видим, что мощность комбинаторно-оптимизационной задачи не обязательно указывает на ее сложность. На рис. 18.1 показано решение задачи о незамкнутом маршруте шахматного коня. Задача о маршруте шахматного коня также была изучена с использованием размеров шахматных досок, отличных от 8×8 клеток.

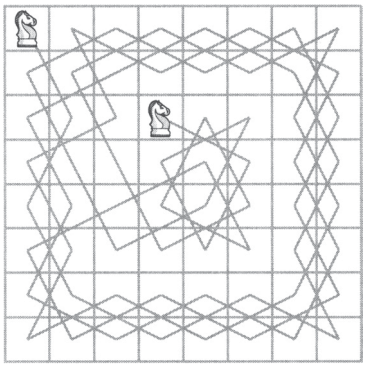


Рис. 18.1. Решение задачи о незамкнутом маршруте шахматного коня [Fealy, 2006, стр. 237]

Краткий обзор главы

Подавляющая часть этой главы посвящена задаче коммивояжера (traveling salesman problem, TSP), которая, пожалуй, является самой известной, применяемой и широко изученной комбинаторно-оптимизационной задачей. В разделе 18.1 дается обзор задачи коммивояжера. В разделе 18.2 обсуждается несколько простых и популярных неэволюционных эвристик для решения задачи коммивояжера; это имеет отношение к эволюционным алгоритмам, потому что мы можем применять обсуждаемые там эвристики для инициализации или улучшения нашей

эволюционно-алгоритмической популяции. В разделе 18.3 обсуждаются разные способы представления кандидатных решений задачи коммивояжера, а также способы комбинирования кандидатных решений в эволюционном алгоритме для получения дочерних решений. В разделе 18.4 обсуждаются некоторые способы мутирования кандидатных решений задачи коммивояжера. Раздел 18.5 связывает воедино материал предыдущих разделов и представляет собой базовый эволюционный алгоритм, который мы можем использовать для решения задачи коммивояжера. В разделе 18.6 обсуждается задача раскраски графов, которая является еще одной популярной комбинаторно-оптимизационной задачей. Обратите внимание, что в приложении С.6 обсуждаются эталонные задачи коммивояжера и другие эталонные комбинаторно-оптимизационные задачи.

18.1. Задача коммивояжера

Маршрут шахматного коня, о котором мы говорили выше, не является сложной задачей, но он подводит нас к задаче коммивояжера, весьма сложной: каков минимальный маршрут, который проходит по каждому из N городов ровно один раз? Как и в случае с маршрутом шахматного коня, замкнутая задача коммивояжера – это задача, в которой маршрут заканчивается в том же городе, в котором он начался; в противном случае задача коммивояжера является незамкнутой. Задача коммивояжера впервые появляется в письменном виде в немецкой брошюре, которая была опубликована в 1832 году под названием «Коммивояжер – каким он должен быть и что он должен делать для того, чтобы получать заказы и быть уверенным в счастливом успехе своего предприятия, – составлено опытным коммивояжером». Австрийский математик Карл Менгер (Karl Menger) назвал задачу коммивояжера «задачей посыльного», и он был первым, кто обсуждал ее в технической литературе в конце 1920-х и начале 1930-х годов [Schrijver, 2005].

Задача коммивояжера имеет применение в робототехнике, сверлении печатных плат, сварочных работах, промышленном производстве, транспортных перевозках и во многих других областях. Как мы обсуждали в разделе 2.5, незамкнутая задача коммивояжера для n городов имеет $(n - 1)!$ возможных решений. Данное число становится невероятно большим даже при умеренных значениях n. Например, число возможных решений задачи коммивояжера для 50 городов равно $49! = 6.1 \times 10^{62}$. А 50 городов – это не очень много для данной задачи. Печатная плата может иметь десятки тысяч отверстий, и сверло должно быть запрог-

раммированно для посещения каждого из этих отверстий, при этом минимизируя некоторую функцию стоимости (к примеру, время или энергию).

В общем случае будем считать, что задача коммивояжера для n городов имеет города, обозначаемые как *город* 1, *город* 2, …, *город* n. Мы будем считать, что существует заданное расстояние $D(i, j)$ между городами i и j для всех $i \in [1, n]$ и $j \in [1, n]$ и что $D(i, j) = D(j, i)$. Такая задача коммивояжера называется симметричной, потому что расстояние (или стоимость) от города i до города j является таким же, что и расстояние от города j до города i. Мы можем представить себе сценариии, где $D(i, j) \neq D(j, i)$ (например, стоимость при подъеме в гору вполне может быть выше, чем при спуске), – подобные задачи называются асимметричными задачами коммивояжера, но в этой главе мы их касаться не будем.

Допустимый маршрут в незамкнутой задаче коммивояжера – это маршрут, в котором все n городов посещаются ровно один раз. Допустимый маршрут в замкнутой задаче коммивояжера – это маршрут, который начинается и заканчивается в том же городе и который проходит через другие (n – 1) городов ровно один раз. Пример допустимого маршрута в незамкнутой задаче коммивояжера для 4 городов будет следующим:

Допустимый незамкнутый маршрут через четыре города: $3 \rightarrow 2 \rightarrow 4 \rightarrow 1$. (18.2)

Ниже приведен пример допустимого маршрута в замкнутой задаче коммивояжера для 4 городов:

Допустимый замкнутый маршрут через четыре города: $3 \rightarrow 2 \rightarrow 4 \rightarrow 1 \rightarrow 3$. (18.2)

Ребро или отрезок – это один сегмент маршрута. Уравнение (18.2) состоит из трех ребер: $3 \rightarrow 2$ – первое ребро, $2 \rightarrow 4$ – второе ребро и $4 \rightarrow 1$ – третье ребро. Уравнение (18.3) состоит из четырех ребер. В общем случае незамкнутый маршрут через n городов содержит (n – 1) ребер и замкнутый маршрут через n городов содержит n ребер.

В задаче коммивояжера мы стараемся минимизировать общее расстояние. Предположим, что n городов в незамкнутой задаче коммивояжера перечислены в порядке $x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_n$. Тогда общее расстояние равняется

$$D_T = \sum_{i=1}^{n-1} D(x_i, x_{i+1}). \quad (18.4)$$

Обратите внимание, что мы используем термин «расстояние» в общеупотребительном смысле. Он вполне может относиться к физическому расстоянию, финансовой стоимости либо любому другому количеству, которое мы хотим минимизировать в комбинаторной задаче.

18.2. Инициализация задачи коммивояжера

В данном разделе представлено несколько популярных и простых неэволюционных эвристик, которые мы можем использовать для решения задач коммивояжера [Nemhauser и Wolsey, 1999]. Мы можем использовать эти эвристики не только для поиска решений данной задачи, но и для инициализации эволюционно-алгоритмической популяции, которая предназначена для решения задачи коммивояжера. Если мы выполним инициализацию эволюционного алгоритма грамотно, не случайно, то мы можем значительно увеличить наши шансы найти хорошее решение. Это относится не только к задаче коммивояжера, но и к любой другой задаче, которую мы хотим решить с помощью эволюционного алгоритма (см. раздел 8.1).

Алгоритмы инициализации в этом разделе называются *жадными*, поскольку все они вносят добавочные изменения в их кандидатные решения, которые обещают немедленное изменение результативности. То есть все они строят кандидатные решения на основе наивысшего немедленного выигрыша. В разделе 18.2.1 строится кандидатное решение путем итеративного добавления города, наиболее близкого к ранее добавленному городу. В разделе 18.2.2 строится кандидатное решение путем итеративного добавления следующего кратчайшего ребра. В разделе 18.2.3 строится кандидатное решение путем итеративного добавления города, ближайшего к любому из ранее добавленных городов. Наконец, в разделе 18.2.4 обсуждается использование случайности в жадных методах инициализации.

18.2.1. *Инициализация на основе ближайшего соседа*

Одним из простых и интуитивно понятных способов инициализации кандидатного решения является стратегия ближайшего соседа. Для задачи коммивояжера для n городов данная стратегия описывается следующим образом.

1. Инициализировать $i = 1$.
2. Случайно отобрать город $s(1) \in [1, n]$ в качестве исходного города.

3. $s(i+1) \leftarrow \arg\min_{\sigma}\{D(s(i), \sigma) : \sigma \notin s(k) \text{ для } k \in [1, i]\}$. То есть найти ближайший к $s(i)$ город, который еще не назначен элементу s, и назначить его $s(i+1)$.
4. Увеличить i на единицу.
5. Если $i = n$, то завершить; в противном случае перейдите к шагу 3.

В конце этого процесса у нас будет незамкнутый маршрут $s(1) \rightarrow s(2) \rightarrow \ldots \rightarrow s(n)$, который дает нам разумную догадку относительно решения задачи коммивояжера. Если мы хотим получить замкнутый маршрут, то просто добавляем $s(1)$ в конец незамкнутого маршрута.

Из-за случайного отбора стартового города мы, как правило, при выполнении этого алгоритма более одного раза получаем разные кандидатные решения. Например, рассмотрим матрицу расстояний

$$D = \begin{bmatrix} \times & 3 & 2 & 9 & 3 \\ 3 & \times & 5 & 8 & 11 \\ 2 & 5 & \times & 4 & 6 \\ 9 & 8 & 4 & \times & 10 \\ 3 & 11 & 6 & 10 & \times \end{bmatrix}, \tag{18.5}$$

где D_{ij}, которое мы также можем записать как $D(i, j)$, представляет расстояние между городом i и городом j. Если мы начнем с города 1, то алгоритм ближайшего соседа даст маршрут $1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 5$, который имеет общую стоимость 25. Если мы начнем с города 2, то алгоритм даст маршрут $2 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 5$, которая имеет общую стоимость 19.

Обратите внимание, что общая $n \times n$-матрица расстояний D является симметричной, поскольку расстояние между городом i и городом j одинаковое, что и между городом j и городом i. Кроме того, $n \times n$-матрица имеет $n(n-1)/2$ членов поверх диагонали. Следовательно, симметричная задача коммивояжера для n городов имеет $n(n-1)/2$ уникальных ребер.

Если мы хотим грамотно инициализировать эволюционный алгоритм для решения задачи коммивояжера, то мы могли бы применить инициализацию на основе ближайшего соседа только для одной особи в популяции, либо для нескольких особей в популяции, либо для всей популяции в целом. Однако если мы будем таким образом инициализировать слишком много особей, то мы, вероятно, получим много дублирующих особей. Мы могли бы также применить алгоритм стохастической инициализации на основе ближайшего соседа – в этом случае вероятность назначения элементу $s(i+1)$ заданного города на каждой итера-

ции была бы обратно пропорциональна его расстоянию от $s(i)$. Наконец, мы могли бы поднять алгоритм ближайшего соседа на другой уровень, выполнив алгоритм «ближайших двух соседей». При таком подходе при наличии $s(i)$ мы могли бы назначать город элементу $s(i + 1)$, который приводит к наименьшему объединенному расстоянию $D(s(i), s(i + 1)) + D(s(i + 1), \sigma)$, где σ разрешено быть равным любому городу $\neq s(k)$ для $k \leq i + 1$.

Нетрудно найти пример, где инициализация на основе ближайшего соседа работает плохо. На рис. 18.2 показана простая задача коммивояжера для 5 городов. На рисунке слева мы начинаем с города 1 и получаем плохой результат при инициализации на основе ближайшего соседа. На рисунке справа мы начинаем с города 3 и получаем глобально оптимальное решение с инициализацией на основе ближайшего соседа. На рис. 18.2 результативность инициализации на основе ближайшего соседа сильно зависит от исходного города. Но в целом инициализация на основе ближайшего соседа часто не срабатывает из-за того, что при планировании маршрута она смотрит не более чем на один город вперед.

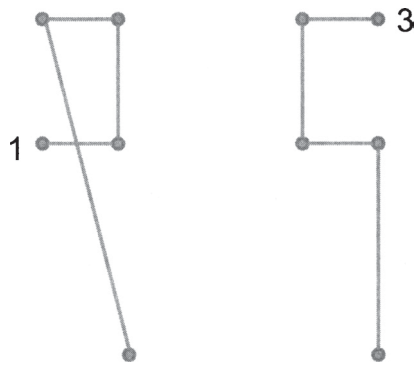


Рис. 18.2. Результаты инициализации на основе ближайшего соседа для незамкнутой задачи коммивояжера для 5 городов. В зависимости от стартового города мы получаем либо плохой результат (слева), либо хороший (справа)

18.2.2. *Инициализация на основе кратчайшего ребра*

Еще один простой способ инициализации кандидатного решения задачи коммивояжера – жадный алгоритм на основе кратчайшего ребра. Предположим, что мы имеем задачу коммивояжера для n городов с матрицей расстояний D, как показано в уравнении (18.5). Мы определяем L_k как ребро, связанное с k-м наименьшим числом в D. То есть $\{L_k\}$ состоит

из $n(n-1)/2$ ребер, отсортированных в порядке возрастания расстояния. Инициализация на основе кратчайшего ребра (shortest-edge initialization) для замкнутой задачи коммивояжера для n городов проходит следующим образом.

1. Определить T как множество ребер в маршруте. Инициализировать T пустым множеством.
2. Найти кратчайшее ребро $\{L_k\}$, которое удовлетворяет следующим ограничениям: (a) оно не находится в T; (b) добавление в T не приведет к замкнутому маршруту с менее чем n ребрами; (c) если оно соединяет города i и j и добавляется в T, то T будет иметь не более двух ребер, связанных с городом i или городом j.
3. Если T имеет n ребер, то работа завершена; в противном случае перейти к шагу (2).

Инициализация на основе кратчайшего ребра включает в маршрут ребра из ближайших друг к другу городов, и этот процесс продолжается до тех пор, пока не будет получен допустимый маршрут. В отличие от инициализации на основе ближайшего соседа, инициализация на основе кратчайшего ребра не является стохастической, поэтому при каждом выполнении она приводит к одному и тому же маршруту. Следовательно, в общем случае инициализация на основе кратчайшего ребра должна применяться для инициализации только одной особи в эволюционном алгоритме. Единственное исключение из этого утверждения: когда несколько пар городов имеют одинаковое расстояние, тогда случайный процесс может быть использован для того, чтобы разбить повторы в приведенном выше шаге (2), и в результате (в общем случае) будет получен другой маршрут, в зависимости от результата случайного процесса.

В качестве примера инициализации на основе кратчайшего ребра рассмотрим матрицу расстояний уравнения (18.5). Инициализация на основе кратчайшего ребра проходит следующим образом.

1. Кратчайшее ребро находится между городами 1 и 3, поэтому мы включаем это ребро в T.
2. Кратчайшее оставшееся ребро находится между городами 1 и 2 и городами 1 и 5. Мы случайно отбираем ребро между городами 1 и 5 для включения в T.
3. Кратчайшее оставшееся ребро находится между городами 1 и 2, но город 1 уже имеет два ребра в T. Поэтому мы смотрим на сле-

дующее кратчайшее ребро, которое находится между городами 3 и 4, и включаем это ребро в T.

4. Кратчайшее оставшееся ребро находится между городами 2 и 3, но город 3 уже имеет два ребра в T. Поэтому мы смотрим на следующее кратчайшее ребро, которое находится между городами 3 и 5, но опять же, город 3 уже имеет два ребра в T. Поэтому мы смотрим на следующее кратчайшее ребро, которое находится между городами 2 и 4, поэтому мы включаем это ребро в T.
5. Единственное оставшееся ребро, которое удовлетворяет ограничениям алгоритма инициализации на основе кратчайшего ребра, – это ребро между городами 2 и 5, поэтому мы включаем это ребро в T для завершения замкнутого маршрута.

Данный алгоритм дает замкнутый маршрут $1 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 1$. Если мы хотим применить инициализацию на основе кратчайшего ребра, для того чтобы найти незамкнутый маршрут, то мы просто остановим приведенный выше алгоритм после получения $(n - 1)$ ребер в T, в результате чего получится маршрут $5 \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow 2$.

18.2.3. *Инициализации на основе вставки*

Инициализация на основе вставки начинается с подмаршрута, а затем добавляет один город за раз в маршрут таким образом, что добавление отобранного города дает наименьшее увеличение расстояния [Rosenkrantz и соавт., 1977]. Исходный подмаршрут часто является единственным ребром, который обычно является кратчайшим. В этом случае мы имеем алгоритм инициализации на основе ближайшей вставки (nearest insertion algorithm), который для незамкнутой задачи коммивояжера задается следующим образом.

1. Определить T как множество ребер в маршруте. Инициализировать T кратчайшим ребром в матрице расстояний.
2. $c \leftarrow \arg\min_c\{D(c, k) : (c \notin T)$ и $(k \in T)\}$. То есть среди всех городов, которые не находятся в Т, отобрать тот, который ближе всего к T.
3. $\{k, j\} \leftarrow \arg\min_{k,j}\{(D(k, c) + D(c, j)) - D(k, j) : D(k, j) \in T\}$. То есть отобрать ребро $D(k, j)$ из T – такое, что разница между расстоянием подмаршрута $k \rightarrow c \rightarrow j$ и расстоянием $k \rightarrow j$ является минимальной.
4. Удалить $D(k, j)$ из T и добавить $D(k, c)$ и $D(c, j)$ в T.
5. Если T содержит $(n - 1)$ ребер, то работа завершена; в противном случае перейти к шагу (2).

Если мы хотим получить замкнутый маршрут, то просто добавим еще одно ребро в *T*, чтобы завершить маршрут. Мы можем модифицировать алгоритм инициализации на основе ближайшей вставки, изменив инициализацию в шаге (1) так, чтобы множество *T* инициализировалось выпуклой оболочкой городов либо инициализировалось случайно, либо инициализировалось другими разнообразными вариантами. Это позволит с помощью алгоритма на основе вставки инициализировать более одной эволюционно-алгоритмической особи.

В качестве примера инициализации на основе ближайшей вставки рассмотрим матрицу расстояний уравнения (18.5). Ближайшая вставка выполняется следующим образом.

1. Кратчайшее ребро находится между городами 1 и 3, поэтому мы включаем это ребро в *T*.
2. Города 2, 4 и 5 еще не находятся в *T*. Среди этих городов наиболее близким к *T* (то есть ближайшим к городу 1 или к городу 3) является город 2 или город 5, оба из которых находятся на расстоянии 3 единиц от города 1. Мы случайно отбираем город 5 для включения в *T*. Затем удаляем ребро 1/3 из *T* и добавляем в *T* ребра 1/5 и 5/3.
3. Города 2 и 4 еще не находятся в *T*. Среди этих городов ближайшим к *T* (то есть ближайшим к городу 1, либо городу 3, либо городу 5) является город 2, который находится на расстоянии 3 единиц от города 1. В результате получаем два варианта.
   a) Мы можем удалить ребро 1/5 из *T* и заменить его ребрами 1/2 и 2/5. В результате расстояние подмаршрута будет увеличено с 3 единиц (расстояние ребра 1/5) до 14 единиц (сумма расстояний ребер 1/2 и 2/5). Суммарное увеличение на 11 единиц.
   b) Мы можем удалить ребро 5/3 из *T* и заменить его ребрами 5/2 и 2/3. В результате расстояние подмаршрута будет увеличено с 6 единиц (расстояние ребра 5/3) до 16 единиц (сумма расстояний ребер 5/2 и 2/3). Суммарное увеличение на 10 единиц.

   Мы выбираем вариант (b), так как он приводит к наименьшему увеличению. *T* теперь включает ребра 1/5, 5/2 и 2/3.
4. Единственный город, который еще не находится в *T*, – это город 4. Следовательно, нам нужно добавить ребро к *T*, которое включает город 4. В результате получаем три варианта.

a) Мы можем удалить ребро 1/5 из T и заменить его ребрами 1/4 и 4/5. В результате расстояние подмаршрута будет увеличено с 3 единиц (расстояние ребра 1/5) на 19 единиц (сумма расстояний ребер 1/4 и 4/5). Суммарное увеличение на 16 единиц.
b) Мы можем удалить ребро 5/2 из T и заменить ее ребрами 5/4 и 4/2. В результате расстояние подмаршрута будет увеличено с 11 единиц (расстояние ребра 5/2) до 18 единиц (сумма расстояний ребер 5/4 и 4/2). Суммарное увеличение на 7 единиц.
c) Мы можем удалить ребро 2/3 из T и заменить его ребрами 2/4 и 4/3. В результате расстояние подмаршрута будет увеличено с 5 единиц (расстояние ребра 2/3) до 12 единиц (сумма расстояний ребер 2/4 и 4/5). Суммарное увеличение на 7 единиц.

Мы можем выбрать либо вариант (b), либо (c), поскольку они приводят к наименьшему увеличению. Мы случайным образом отбираем вариант (c). T теперь включает ребра 1/5, 5/2, 2/4 и 4/3.

Данный алгоритм дает незамкнутый маршрут $1 \rightarrow 5 \rightarrow 2 \rightarrow 4 \rightarrow 3$.

18.2.4. *Стохастическая инициализация*

Инициализация на основе ближайшего соседа – это единственный рассмотренный нами к настоящему моменту метод инициализации, который является стохастическим. Вместе с тем для включения случайной компоненты мы можем модифицировать любые другие параметры инициализации. Кроме того, мы можем изменить инициализацию на основе ближайшего соседа, сделав ее более случайной, чем алгоритм, который мы представили в разделе 18.2.1. Добавление случайности в метод инициализации сохраняет привлекательные особенности метода, при этом позволяя встраивать один из фундаментальных компонентов эволюционных алгоритмов. Это также дает возможность применять методы инициализации для более чем одного эволюционного алгоритма.

В инициализации на основе ближайшего соседа (раздел 18.2.1) мы могли бы заменить шаг (3) «Найти ближайший к $s(i)$ город» на отбор города с вероятностью, обратно пропорциональной расстоянию от $s(i)$). Это даст ближайшему к $s(i)$ городу наибольшую вероятность быть отобранным, но также даст ненулевые вероятности отбора всем другим городам в задаче коммивояжера.

В инициализации на основе кратчайшего ребра (раздел 18.2.2) мы могли бы заменить отбор кратчайшего ребра, удовлетворяющего заданным ограничениям, отбором ребра (среди тех, которые удовлетворяют заданным ограничениям) с вероятностью, обратно пропорциональной длине ребра. Это даст наибольшую вероятность кратчайшему ребру, но также даст ненулевые вероятности отбора всем другим ребрам в задаче коммивояжера.

В инициализации на основе вставки (раздел 18.2.3) мы могли бы добавить случайность в два шага. Во-первых, в шаге (2) вместо отбора ближайшего к T города мы могли бы отобрать город с вероятностью, обратно пропорциональной расстоянию от T. Это даст наибольшую вероятность ближайшему городу, но также даст ненулевые вероятности отбора всем другим городам, которые еще не находятся в T. Во-вторых, в шаге (3) вместо отбора ребра – такого, что разница в расстояниях подмаршрутов минимизируется, – мы могли бы отобрать ребро с вероятностью, обратно пропорциональной разнице в расстояниях подмаршрутов. Это даст наибольшую вероятность ребру с минимальной разностью расстояний, но также даст ненулевые вероятности отбора всем другим ребрам в T.

18.3. Представления задачи коммивояжера и скрещивание

В данном разделе обсуждаются различные способы представления кандидатных решений задачи коммивояжера. Мы рассмотрим представления в виде пути (раздел 18.3.1), смежности (раздел 18.3.2), порядкового числа (раздел 18.3.3) и матрицы (раздел 18.3.4). Мы также обсудим вопрос комбинирования кандидатных решений с помощью скрещивания с использованием разных представлений.

18.3.1. *Представление в виде пути*

Представление в виде пути является наиболее естественным способом представления маршрута в задаче коммивояжера. В представлении в виде пути вектор

$$x = \{x_1 \quad x_2 \quad \ldots \quad x_n\} \tag{18.6}$$

обозначает маршрут через n городов $x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_n$. В следующих далее разделах рассматривается несколько способов комбинирования

представленных таким образом родительских особей для получения детских особей.

18.3.1.1. Скрещивание с частичным совпадением

Скрещивание с частичным совпадением (partially matched crossover, PMX) [Goldberg и Lingle, 1985] опирается на классическое одноточечное скрещивание, которое часто используется в генетических алгоритмах (см. раздел 8.8). В качестве примера рассмотрим два родительских вектора

$$P_1 = [\,2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 1\,]$$
$$P_2 = [\,3\ \ 2\ \ 6\ \ 1\ \ 4\ \ 5\,] \qquad (18.7)$$

Если мы выполним одноточечное скрещивание в срединной точке двух векторов, то получим детей

$$c_1 = [\,2\ \ 3\ \ 4\ \ 1\ \ 4\ \ 5\,]$$
$$c_2 = [\,3\ \ 2\ \ 6\ \ 5\ \ 6\ \ 1\,] \qquad (18.8)$$

Эти дети никуда не годятся, потому что c_1 посещает город 4 дважды и вообще не посещает город 6. c_2 имеет противоположную проблему; он посещает город 6 дважды и вообще не посещает город 4. Мы вполне можем легко исправить поведение детей путем замены одного из 4 элементов в c_1 на 6 и путем замены одного из 6 элементов в c_2 на 4. Например, в результате дети из уравнения (18.8) будут модифицированы в

$$c_1 = [\,2\ \ 3\ \ 6\ \ 1\ \ 4\ \ 5\,]$$
$$c_2 = [\,3\ \ 2\ \ 6\ \ 5\ \ 4\ \ 1\,] \qquad (18.9)$$

где выделенные жирным шрифтом города – это города, которые были случайно изменены для получения допустимых маршрутов.

18.3.1.2. Упорядоченное скрещивание

В упорядоченном скрещивании (order crossover, OX) в ребенка копируется часть маршрута от одного родителя [Davis, 1985]. В результате получается ребенок, который имеет частичный маршрут. Затем упорядоченное скрещивание завершает ребенка, копируя в ребенка оставшиеся требуемые города от второго родителя, сохраняя относительный порядок этих городов от родителя 2. Например, предположим, что у нас есть родители

$$P_1 = [\,9\ \ 2\ \ 3\ \ 8\ \ 4\ \ 5\ \ 6\ \ 1\ \ 7\,]$$
$$P_2 = [\,4\ \ 5\ \ 2\ \ 1\ \ 8\ \ 7\ \ 6\ \ 9\ \ 3\,] \qquad (18.10)$$

Мы случайно отбираем подмаршрут из P_1; предположим, что мы отбираем подмаршрут [8 4 5 6] из P_1. В результате получаем частичного ребенка

$$c_1 = [\,-\ -\ -\ 8\ 4\ 5\ 6\ -\ -\,]. \tag{18.11}$$

Мы видим, что c_1 все еще нуждается в городах 1, 2, 3, 7, и 9. Эти города встречаются в порядке {2, 1, 7, 9, 3} в P_2. Поэтому мы копируем эти города в таком порядке в c_1 и получаем

$$c_1 = [\,2\ 1\ 7\ 8\ 4\ 5\ 6\ 9\ 3\,]. \tag{18.12}$$

В упорядоченном скрещивании мы часто создаем второго ребенка, используя приведенный выше процесс, в котором роли P_1 и P_2 переставлены местами. В нашем примере это даст предварительного второго ребенка с подмаршрутом, скопированным из P_2 в виде

$$c_2 = [\,-\ -\ -\ 1\ 8\ 7\ 6\ -\ -\,]. \tag{18.13}$$

Затем мы скопируем оставшиеся города 9, 2, 3, 4 и 5 в указанном порядке из P_1 и завершим второго ребенка:

$$c_2 = [\,9\ 2\ 3\ 1\ 8\ 7\ 6\ 4\ 5\,]. \tag{18.14}$$

18.3.1.3. Циклическое скрещивание

В циклическом скрещивании (cycle crossover, CX), введенном в публикации [Oliver и соавт., 1987], ребенок создается от двух родителей таким образом, чтобы сохранить как можно больше информации о последовательности от первого родителя, при заполнении ребенка информацией от второго родителя. Циклическое скрещивание лучше всего объяснить на примере. Предположим, что у нас есть родители

$$\begin{aligned} P_1 &= [\,2\ 3\ 4\ 5\ 6\ 1\,] \\ P_2 &= [\,4\ 5\ 2\ 1\ 6\ 3\,] \end{aligned} \tag{18.15}$$

Мы создаем ребенка следующим образом.

1. Выбрать случайный индекс от 1 до n. Предположим, что мы выбираем 4. $P_1(4) = 5$, поэтому ребенок инициализируется как $c = [\,-\ -\ -\ 5\ -\ -\,]$.
2. $P_2(4) = 1$, и город 1 находится в шестой позиции в P_1, поэтому ребенок усиливается, становясь $c = [\,-\ -\ -\ 5\ -\ 1\,]$.
3. $P_2(6) = 3$, и город 3 находится во второй позиции в P_2, поэтому ребенок усиливается, становясь $c = [\,-\ 3\ -\ 5\ -\ 1\,]$.

4. $P_2(2) = 5$, но ребенок уже содержит город 5. Поэтому мы копируем оставшиеся города в ребенка из P_2, в результате получив $c = [\,4\;\;3\;\;2\;\;5\;\;6\;1\,]$.

Мы часто создаем второго ребенка, меняя ролями P_1 и P_2. На рис. 18.3 показан цикл работы циклического скрещивания. Циклическое скрещивание всегда приводит к допустимым детям.

P_1 = Родитель 1, P_2 = Родитель 2
$s \leftarrow$ случайное целое из $[1, n]$
$r \leftarrow P_1(s)$
Инициализировать ребенка пустым маршрутом: $C(i) = 0$ для $i \in [1, n]$.
$C(s) \leftarrow r$
While $C(i) = 0$ для некоторого $i \in [1, n]$
  $r \leftarrow P_2(s)$
  If $C(i) \neq r$ для всех $i \in [1, n]$ **then**
    $s \leftarrow \{\, i : P_1(i) = r \,\}$
    $C(s) \leftarrow r$
  else
    For $i = 1$ **to** n
      If $C(i) = 0$ **then** $C(i) \leftarrow P_2(i)$
    Next i
  End if
Next город

Рис. 18.3. Циклическое скрещивание для задачи коммивояжера для n городов

18.3.1.4. Реорганизующее скрещивание

Реорганизующее скрещивание (order-based crossover, OBX) является модификацией циклического скрещивания [Syswerda, 1991]. Реорганизующее скрещивание случайно отбирает несколько позиций в первом родителе P_1, находит города в соответствующих позициях в P_2, а затем реорганизовывает эти города в P_1 с помощью порядка следования городов из P_2. В результате получается ребенок. Например, предположим, что у нас есть родители

$$P_1 = [\,2\;\;3\;\;4\;\;5\;\;6\;\;1\,]$$
$$P_2 = [\,4\;\;5\;\;2\;\;1\;\;6\;\;3\,]. \qquad (18.16)$$

Реорганизующее скрещивание будет проходить путем случайного отбора определенного числа позиций в P_1. Предположим, что мы отбираем позиции 1, 3 и 4. Городами в этих позициях в P_2 являются города 4, 2

и 1. Ребенок инициализируется всеми городами из P_1, за исключением городов 4, 2 и 1:

$$c_1 = [\ -\ \ 3\ \ -\ \ 5\ \ 6\ \ -\ ]. \tag{18.17}$$

Затем мы копируем города 4, 2 и 1 в ребенка в том же порядке, в котором они встречаются в P_2. В результате получаем ребенка

$$c_1 = [\ 4\ \ 3\ \ 2\ \ 5\ \ 6\ \ 1\ ]. \tag{18.18}$$

Мы часто создаем второго ребенка, меняя ролями P_1 и P_2, что в приведенном выше примере приводит к инициализации c_2 всеми городами из P_2, за исключением городов 2, 4 и 5 (поскольку эти города находятся в позициях 1, 3 и 4 в P_1):

$$c_2 = [\ -\ \ -\ \ -\ \ 1\ \ 6\ \ 3\ ]. \tag{18.19}$$

Затем мы копируем города 2, 4 и 5 в ребенка в том же порядке, в котором они встречаются в P_1, получая

$$c_2 = [\ 2\ \ 4\ \ 5\ \ 1\ \ 6\ \ 3\ ]. \tag{18.20}$$

18.3.1.5. Скрещивание с инверсией по опорному элементу

С учетом двух родителей P_1 и P_2 скрещивание с инверсией по опорному элементу (inver-over crossover) работает следующим образом [Tao и Michalewicz, 1998].

1. Случайно отобрать позицию s из P_1. Предположим, что $P_1(s) = r$.
2. Предположим, что r находится в k-й позиции в P_2, то есть $P_2(k) = r$. Установить конечный город равным $e = P_2(k + 1)$.
3. Инвертировать порядок следования городов между $P_1(s + 1)$ и e в P_1 и получить ребенка.

Например, предположим, что у нас есть родители

$$\begin{aligned} P_1 &= [\ 2\ \ 3\ \ 4\ \ 5\ \ 6\ \ 1\ ] \\ P_2 &= [\ 4\ \ 5\ \ 2\ \ 1\ \ 6\ \ 3\ ] \end{aligned} \tag{18.21}$$

Мы случайно отбираем позицию s из P_1; предположим, что мы отбираем $s = 4$. Мы видим, что $P_1(4) = 5$ и город 5 находится на втором месте в P_2, то есть $P_2(2) = 5$. Поэтому устанавливаем $e = P_2(3) = 2$ в качестве конечного города. Затем мы меняем порядок следования городов между $P_1(5)$ и городом 2 в P_1 на обратный и получаем

$$c = [\ 6\ \ 5\ \ 4\ \ 3\ \ 2\ \ 1\ ]. \tag{18.22}$$

Если в шаге (2) $k = n$, то $P_2(k+1)$ не определено. В этом случае для продолжения процесса скрещивания мы можем применить какой-нибудь ситуативный метод; например, мы могли бы установить $e = P_2(k-1)$ либо вернуться к шагу (1) и отобрать новый случайный s.

18.3.2. *Представление в виде смежности*

В представлении в виде смежности [Grefenstette и соавт., 1985] если маршрут, представленный вектором x, содержит прямой путь из города i в город j, то $x(i) = j$, то есть i-й элемент x равен j. Например, рассмотрим вектор

$$x = [\,2 \quad 4 \quad 8 \quad 3 \quad 9 \quad 7 \quad 1 \quad 5 \quad 6\,]. \tag{18.23}$$

Вектор x интерпретируется следующим образом.

- $x(1) = 2$, поэтому маршрут содержит ребро из города 1 в город 2.
- $x(2) = 4$, поэтому маршрут содержит ребро из города 2 в город 4.
- $x(3) = 8$, поэтому маршрут содержит ребро из города 3 в город 8.
- $x(4) = 3$, поэтому маршрут содержит ребро из города 4 в город 3.
- $x(5) = 9$, поэтому маршрут содержит ребро из города 5 в город 9.
- $x(6) = 7$, поэтому маршрут содержит ребро из города 6 в город 7.
- $x(7) = 1$, поэтому маршрут содержит ребро из города 7 в город 1.
- $x(8) = 5$, поэтому маршрут содержит ребро из города 8 в город 5.
- $x(9) = 6$, поэтому маршрут содержит ребро из города 9 в город 6.

Собрав все это вместе, мы видим, что маршрут, представленный вектором x, равен

$$1 \to 2 \to 4 \to 3 \to 8 \to 5 \to 9 \to 6 \to 7 \to 1. \tag{18.24}$$

Вектор x, использующий для задачи коммивояжера для n городов представление в виде смежности, обладает следующими свойствами.

- $x(i) \neq i$ для всех $i \in [1, n]$.
- Для всех $j \in [1, n]$ существует ровно один $i \in [1, n]$ – такой, что $x(i) = j$.

Вышеуказанные свойства необходимы, но не достаточны для того, чтобы x представлял допустимый маршрут. Например, вектор

$$x = [\,2 \quad 1 \quad 8 \quad 3 \quad 9 \quad 7 \quad 4 \quad 5 \quad 6\,] \tag{18.25}$$

недопустимый. Если мы начнем в городе 1, то повторим последовательность $1 \to 2 \to 1 \to 2 \to \ldots$ бесконечно, и мы никогда не посетим остальные города. В следующих разделах рассматриваются некоторые спо-

собы комбинирования представленных таким образом родительских особей для получения детских особей.

18.3.2.1. Классическое скрещивание

Сначала мы обсудим метод скрещивания, который не работает с представлением в виде смежности, и это одноточечное скрещивание (classic single-point crossover), используемое в генетических алгоритмах (см. раздел 8.8). Например, рассмотрим два родительских вектора

$$P_1 = [\,2 \quad 4 \quad 1 \quad 3\,]$$
$$P_2 = [\,4 \quad 3 \quad 1 \quad 2\,] \tag{18.26}$$

Если мы выполним одноточечное скрещивание в срединной точке двух векторов, то получим детей

$$c_1 = [\,2 \quad 4 \quad 1 \quad 2\,]$$
$$c_2 = [\,4 \quad 3 \quad 1 \quad 3\,] \tag{18.27}$$

c_1 представляет маршрут $1 \to 2 \to 4 \to 2$, который является недопустимым, потому что он никогда не посещает город 3. Обратите внимание, что c_1 содержит две записи «2» и ни одной записи «3». c_2 представляет маршрут $1 \to 4 \to 3 \to 1$, который является недопустимым, потому что он никогда не посещает город 2. Обратите внимание, что c_2 содержит две записи «3» и ни одной записи «2».

18.3.2.2. Скрещивание с чередующимися ребрами

Скрещивание с чередующимися ребрами (alternating edges crossover) начинается с классического скрещивания и исправляет недопустимые маршруты [Grefenstette и соавт., 1985]. Например, мы знаем, что c_1 в уравнении (18.27) недопустим, потому что он имеет две записи «2» и ни одной записи «3». Мы можем попытаться исправить его, заменив одну из записей «2» на «3». Если мы заменим первую запись «2» на «3», то получим

$$c_1' = [\,3 \quad 4 \quad 1 \quad 2\,], \tag{18.28}$$

Это не сработает, потому что это приведет к циклу $1 \to 3 \to 1 \to \ldots$ Поэтому мы можем попытаться исправить c_1, заменив вторую запись «2» на «3», и получить

$$c_1'' = [\,2 \quad 4 \quad 1 \quad 3\,], \tag{18.29}$$

что представляет собой допустимый маршрут $1 \to 2 \to 4 \to 3$. Приведенный выше пример показывает одноточечное скрещивание, но

скрещивание с чередующимися ребрами можно также использовать с двухточечным скрещиванием либо с более большим числом точек скрещивания. Хотя скрещивание с чередующимися ребрами работает в том смысле, что оно создает допустимые маршруты, оно часто разрушает хорошие маршруты, и поэтому оно обычно не очень хорошо работает на практике.

18.3.2.3. Эвристическое скрещивание

Эвристическое скрещивание (heuristic crossover) называется так потому, что в нем для комбинирования двух кандидатных решений используется здравый смысл [Grefenstette и соавт., 1985]. Во время создания ребенка оно работает, комбинируя лучшие ребра от двух родителей. Эвристическое скрещивание описывается следующим образом.

1. Выбрать случайный город r в качестве исходной точки.
2. Сравнить ребра от родителей, которые покидают город r. Отобрать для ребенка более короткое ребро.
3. Город на другой стороне выбранного выше ребра является исходной точкой для выбора следующего города.
4. Если отобранный город уже находится в ребенке, то заменить отобранный город случайным городом, которого нет в ребенке.
5. Продолжить выполнение с шага (2) до завершения дочернего маршрута.

На рис. 18.4 представлен псевдокод эвристического скрещивания. Обратите внимание, что мы можем использовать эвристическое скрещивание с любым векторным представлением задачи коммивояжера. Мы можем легко расширить рис. 18.4 для более чем двух родителей. Кроме того, мы могли бы поэкспериментировать с другими модификациями рис. 18.4. Например, вместо детерминированного выбора $r_{\min}$ из $\{r_1, r_2\}$ для минимизации $[d(r, r_1), d(r, r2)]$ мы могли бы отбирать $r_{\min}$ стохастически. Это может повлечь за собой, например, принятие $r_{\min} = r$ с вероятностью, обратно пропорциональной $d(r, r_1)$ для $i \in [1, 2]$.

P_1 = Родитель 1, P_2 = Родитель 2
$r \leftarrow$ случайное целое из $[1, n]$
Ребенок $C \leftarrow \{r\}$
While $|C| < n$
  Применить r_i для обозначения города, который следует за r в Родителе i,
    для $i \in [1, 2]$.
  $d(r, r_i)$ = расстояние от r до r_i, для $i \in [1, 2]$

$r_{\min} \leftarrow \min_{\{r1, r2\}} [d(r, r_1), d(r, r_2)]$
$r \leftarrow r_{\min}$
If $r \in C$ **then**
$r \leftarrow$ случайный город в $[1, n]$ – такой, что $r \notin C$
End if
$C \leftarrow \{C, r\}$
Next город

Рис. 18.4. Эвристическое скрещивание для задачи коммивояжера для n городов

В качестве иллюстрации эвристического скрещивания предположим, что у нас есть задача коммивояжера для 4 городов с матрицей расстояний

$$D = \begin{bmatrix} - & 13 & 9 & 15 \\ 13 & - & 4 & 7 \\ 9 & 4 & - & 12 \\ 15 & 7 & 12 & - \end{bmatrix}, \tag{18.30}$$

где D_{ij} представляет расстояние между городом i и городом j. Предположим, у нас есть родители

$$P_1 = [\,2 \quad 4 \quad 1 \quad 3\,]$$
$$P_2 = [\,4 \quad 3 \quad 1 \quad 2\,] \tag{18.31}$$

Эвристическое скрещивание проходит следующим образом.

1. Мы случайно отбираем исходный город $r \in [1, 4]$; предположим, что мы выбираем $r = 2$.
2. Мы видим, что P_1 имеет ребро $2 \rightarrow 4$, и $d\{2, 4) = 7$. Видим, что P_2 имеет ребро $2 \rightarrow 3$, и $d(2, 3) = 4$. Следовательно, мы отбираем для ребенка ребро $2 \rightarrow 3$, в результате получая $c = \{2, 3\}$.
3. Мы видим, что у обоих родителей есть ребро $3 \rightarrow 1$, поэтому ребенок увеличивается, становясь $c = \{2, 3, 1\}$.
4. Мы видим, что P_1 имеет ребро $1 \rightarrow 2$, и $d(1, 2) = 13$. Мы видим, что P_2 имеет ребро $1 \rightarrow 4$, и $d(1, 4) = 15$. Следовательно, мы отбираем для ребенка ребро $1 \rightarrow 2$. Но город 2 уже находится в C, поэтому мы отбираем случайный город, который не находится в C. Предположим, что мы отбираем город 4 (который, по сути, единственный город, который еще не находится в C). В результате получаем $C = \{2, 3, 1, 4\}$.

5. C теперь завершен, и его представление в виде смежности равно $C = [\,4\ 3\ 1\ 2\,]$.

18.3.3. *Порядковое представление*

В порядковом представлении (ordinal representation) [Grefenstette и соавт., 1985] маршрут через n городов представляется в виде вектора

$$x = [\,x_1\ \ x_2\ \ \ldots\ \ x_n\,], \tag{18.32}$$

где $x_i \in [1, n-i]$. То есть

$$\begin{aligned} x_1 &\in [1, n] \\ x_2 &\in [1, n-1] \\ x_3 &\in [1, n-2] \\ &\vdots \\ x_n &= 1 \end{aligned} \tag{18.33}$$

Предположим, что у нас есть упорядоченный список городов:

$$L = \{\,1\ \ 2\ \ \ldots\ \ n\,\}. \tag{18.34}$$

То есть $L(i) = i$ для $i \in [1, n]$. В порядковом представлении x_1 представляет первый город маршрута. x_2 дает во множестве $L_2 = L\backslash\{x_1\}$ индекс второго города маршрута[1]. x_3 дает во множестве $L_3 = L\backslash\{x_1, x_3\}$ индекс третьего города маршрута. В общем случае x_k дает во множестве $L_k = L\backslash\bigcup_{i=1}^{k-1} x_i$ индекс k-го города маршрута. Обратите внимание, что любой вектор формы уравнения (18.33) представляет собой допустимый маршрут.

Например, предположим, что маршрут через 6 городов представлен в виде

$$x = [\,5\ \ 2\ \ 4\ \ 1\ \ 2\ \ 1\,]. \tag{18.35}$$

С учетом упорядоченного списка $L = \{1, 2, 3, 4, 5, 6\}$ мы конструируем маршрут, представляемый x следующим образом.

1. $x_1 = 5$ и $L(5) = 5$, поэтому город 5 является первым городом в маршруте. Удаление 5 из L дает $L_2 = \{1, 2, 3, 4, 6\}$.
2. $x_2 = 2$ и $L_2(2) = 2$, поэтому город 2 является вторым городом в маршруте. Удаление 2 из L_2 дает $L_3 = \{1, 3, 4, 6\}$.
3. $x_3 = 4$ и $L_3(4) = 6$, поэтому город 6 является третьим городом в маршруте. Удаление 6 из L_3 дает $L_4 = \{1, 3, 4\}$.

[1] Мы используем запись $A\backslash B$ для обозначения множества $\{x : x \in A \text{ и } x \notin B\}$. То есть $A\backslash B$ означает множество всех элементов в A, которые не находятся во множестве B.

4. $x_4 = 1$ и $L_4(1) = 1$, поэтому город 1 является четвертым городом в маршруте. Удаление 1 из L_4 дает $L_5 = \{3, 4\}$.
5. $x_5 = 2$ и $L_5(2) = 4$, поэтому город 4 является пятым городом в маршруте. Удаление 4 из L_5 дает $L_6 = \{3\}$.
6. $x_6 = 1$ и $L_6(1) = 3$, поэтому город 3 является шестым городом в маршруте.

В результате получается маршрут $5 \to 2 \to 6 \to 1 \to 4 \to 3$.

Предположим, что мы хотим испытать одноточечное скрещивание с порядковым представлением, чтобы скомбинировать двух родителей и получить ребенка. Рассмотрим родителей:

$$\begin{aligned} P_1 &= [\,5 \quad 2 \quad 4 \quad 1 \quad 2 \quad 1\,] \\ P_2 &= [\,1 \quad 5 \quad 3 \quad 3 \quad 1 \quad 1\,] \end{aligned} \tag{18.36}$$

Если в качестве точки скрещивания мы отбираем срединную точку родителей, то получим детей

$$\begin{aligned} c_1 &= [\,5 \quad 2 \quad 4 \quad 3 \quad 1 \quad 1\,] \\ &= 5 \to 2 \to 6 \to 4 \to 1 \to 3 \\ c_2 &= [\,1 \quad 5 \quad 3 \quad 1 \quad 2 \quad 1\,] \\ &= 1 \to 6 \to 4 \to 2 \to 5 \to 3 \end{aligned} \tag{18.37}$$

Оба ребенка представляют допустимые маршруты. Хотя порядковое представление сначала кажется немного неудобным, у него есть преимущество, которое состоит в том, что одноточечное скрещивание всегда приводит к допустимому маршруту.

18.3.4. *Матричное представление*

В матричном представлении незамкнутый маршрут через n городов представляется $n \times n$-матрицей M, содержащей только нули и единицы [Fox и McMahon, 1991]. $M_{ik} = 1$ тогда и только тогда, когда город i предшествует городу k в маршруте. Например, рассмотрим матрицу

$$M = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{18.38}$$

Единицы в первой строке указывают на то, что город 1 находится перед городами 2, 4 и 5. Единицы во второй строке указывают на то,

что город 2 находится перед городами 4 и 5. Единицы в третьей строке указывают на то, что город 3 находится перед городами 1, 2, 4 и 5. Единица в четвертой строке указывает на то, что город 4 находится перед городом 5. Наконец, тот факт, что пятая строка состоит из одних нулей, указывает на то, что город 5 является последним городом в маршруте. Следовательно, уравнение (18.38) представляет маршрут $3 \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 5$.

Еще один способ интерпретировать уравнение (18.38) – отметить, что строка с наибольшим числом единиц является первым городом, строка со вторым наибольшим числом единиц – вторым городом и т. д. Строка с k-м наибольшим числом единиц является k-м городом в маршруте.

Отметим несколько свойств для любой $n \times n$-матрицы M, представляющей допустимый маршрут.

- Ровно одна строка M имеет $(n - 1)$ единиц, ровно одна строка M имеет $(n - 2)$ единиц и т. д.
- Вышеуказанное свойство позволяет нам найти число единиц в M:

$$\text{Число единиц} = \sum_{i=1}^{n}(n - i) = n(n - 1)/2. \tag{18.39}$$

- Ни один город не встречается в маршруте перед самим собой, поэтому $M_{ii} = 0$ для всех $i \in [1, n]$.
- Если город i встречается перед городом j и город j встречается перед городом k, то город i встречается перед городом k. То есть

$$(M_{ij} = 1 \text{ и } M_{jk} = 1) \Rightarrow M_{ik} = 1. \tag{18.40}$$

В следующих разделах обсуждается несколько способов комбинирования родительских матриц для получения детей: мы можем взять пересечение либо объединение двух матриц.

18.3.4.1. Скрещивание на основе пересечения

Проиллюстрируем скрещивание на основе пересечения (intersection crossover) на примере. Предположим, что уравнение (18.38) представляет родителя M_1 и что второй родитель задан как

$$M_2 = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}. \tag{18.41}$$

M_2 представляет маршрут $4 \rightarrow 1 \rightarrow 2 \rightarrow 5 \rightarrow 3$. Мы берем пересечение M_1 и M_2, выполняя операцию поэлементного логического «И» на двух матрицах. В результате получаем частично определенного ребенка

$$M_c = M_1 \wedge M_2 = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{18.42}$$

Полученная матрица не представляет допустимый маршрут, но зато она указывает на то, что город 1 находится перед городами 2 и 5, что город 2 находится перед городом 5 и что город 4 находится перед городом 5. Данное упорядочение возникает, потому что то же самое упорядочение встречается у обоих родителей, – на самом деле это единственное упорядочение, которое у обоих родителей является общим. На этом этапе мы можем псевдослучайно добавлять единицы в M_c до тех пор, пока маршрут не станет допустимым (то есть пока он не будет удовлетворять всем перечисленным выше свойствам). Например, мы вполне можем решить добавить единицы в M_c и получить

$$M_2 = \begin{bmatrix} 0 & 1 & \mathbf{1} & \mathbf{1} & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{1} & 0 & 1 \\ 0 & \mathbf{1} & \mathbf{1} & 0 & 0 \end{bmatrix}, \tag{18.43}$$

где добавленные единицы выделены жирным шрифтом. M_c теперь удовлетворяет всем свойствам допустимого маршрута и представляет маршрут $1 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 3$.

18.3.4.2. Скрещивание на основе объединения

Теперь проиллюстрируем пример скрещивания на основе объединения (union crossover). Предположим, что уравнения (18.38) и (18.41) представляют родителей M_1 и M_2. Мы получаем объединение M_1 и M_2, выполняя операцию поэлементного логического «ИЛИ» на двух матрицах. В результате получаем частично определенного ребенка

$$M_c = M_1 \vee M_2 = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}. \tag{18.44}$$

Затем мы отбираем случайную «точку разреза», которая делит M_c на четыре квадранта (не обязательно одинакового размера). Предположим, что мы генерируем случайную точку разреза во второй строке и во втором столбце. Мы записываем M_c с левым верхним и правым нижним квадрантами без изменений, но с левым нижним и правым верхним квадрантами, заполненными неопределенными членами:

$$M_c = \begin{bmatrix} 0 & 1 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ \times & \times & 0 & 1 & 1 \\ \times & \times & 1 & 0 & 1 \\ \times & \times & 1 & 0 & 0 \end{bmatrix}. \tag{18.45}$$

Затем мы вносим необходимые изменения в M_c для устранения противоречий. Например, $M_{c34} = 1$ и $M_{c43} = 1$, поэтому один из этих элементов должен быть изменен на 0. Аналогичным образом $M_{c35} = 1$ и $M_{c53} = 1$, поэтому один из этих элементов должен быть изменен на 0. В результате получаем исправленного, но по-прежнему частично определенного ребенка в виде

$$M_c = \begin{bmatrix} 0 & 1 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ \times & \times & 0 & 0 & 0 \\ \times & \times & 1 & 0 & 1 \\ \times & \times & 1 & 0 & 0 \end{bmatrix}. \tag{18.46}$$

Наконец, мы псевдослучайно добавляем единицы во внедиагональные блоки в M_c до тех пор, пока маршрут не станет допустимым (то есть пока он не будет удовлетворять всем свойствам, перечисленным ранее). Например, мы вполне можем решить добавить единицы в M_c и получить

$$M_c = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}. \tag{18.47}$$

M_c теперь удовлетворяет всем свойствам допустимого маршрута и представляет маршрут $1 \to 2 \to 4 \to 5 \to 3$.

18.4. Мутация в задаче коммивояжера

В данном разделе рассматривается несколько способов мутирования решений задачи коммивояжера. Мы ограничиваем наше обсуждение представлениями в виде пути (см. раздел 18.3.1). Мутации для других представлений можно найти в литературе. Кроме того, любое представление может быть конвертировано в представление в виде пути, а затем мы можем использовать один из методов мутации, описанных в этом разделе.

18.4.1. *Инверсия*

Инверсия меняет порядок маршрута между двумя случайно отобранными индексами [Fogel, 1990]. Например, x может быть мутирован в x_m следующим образом:

$$\begin{aligned} x &= 1 \rightarrow \underbrace{5 \rightarrow 4 \rightarrow 7 \rightarrow 6} \rightarrow 2 \rightarrow 3 \\ x_m &= 1 \rightarrow \overbrace{6 \rightarrow 7 \rightarrow 4 \rightarrow 5} \rightarrow 2 \rightarrow 3 \end{aligned} \tag{18.48}$$

где мы случайно отобрали стартовую и финишную точки мутирующего сегмента. Инверсия также называется мутацией 2-opt[2] [Beyer и Schwefel, 2002]. Существует $n(n-1)/2$ уникальных способов реализации инверсии в маршрут задачи коммивояжера для n городов. Самое низкостоимостное решение, которое получается из всех возможных инверсий маршрута задачи коммивояжера для n городов, всегда приводит к маршруту без каких-либо пересеченных ребер [Bäck и соавт., 1997a].

18.4.2. *Вставка*

Вставка перемещает город в позиции i в позицию k, где i и k отбираются случайно [Fogel, 1988]. Например, предположим, что мы имеем маршрут x, показанный в уравнении (18.48). Предположим, что мы случайно отбираем $i = 4$ и $k = 2$. Тогда мы перемещаем город 7, который находится в позиции 4, в позицию 2, чтобы получить мутированный маршрут

$$x_m = 1 \rightarrow 7 \rightarrow 5 \rightarrow 4 \rightarrow 6 \rightarrow 2 \rightarrow 3. \tag{18.49}$$

Вставка также называется мутацией or-opt [Beyer and Schwefel, 2002].

[2] Эвристика 2-opt относится классу постоптимизационных алгоритмов k-opt, которые отличаются тем, что вы продолжаете совершенствовать уже найденное решение. – *Прим. перев.*

18.4.3. *Перенесение*

Перенесение (displacement) является обобщением вставки [Michalewicz, 1996, глава 10]. Перенесение берет последовательность из q городов, начинающуюся с i-й позиции, и переносит их в i-ю позицию в маршруте, где q, i и k отбираются случайно. Например, предположим, что мы имеем маршрут x, показанный в уравнении (18.48). Предположим, что мы случайно отбираем $q = 2$, $i = 4$ и $k = 2$. Тогда мы берем последовательность из двух городов, начинающуюся с позиции 4 (города 7 и 6), и переносим ее в позицию 2, получив мутированный маршрут

$$x_m = 1 \rightarrow 7 \rightarrow 6 \rightarrow 5 \rightarrow 4 \rightarrow 2 \rightarrow 3. \tag{18.50}$$

Перенесение также называется сдвигом (shifting) [Beyer и Schwefel, 2002]. Мы можем скомбинировать перенесение с инверсией, изменив порядок отобранных городов, перед тем как переместить их в новую позицию.

18.4.4. *Взаимообмен*

Взаимообмен (reciprocal exchange) меняет местами города в i-й и k-й позициях, где i и k отбираются случайно [Banzhaf, 1990]. Например, предположим, что мы имеем маршрут x, показанный в уравнении (18.48). Предположим, что мы случайно отбираем $i = 5$ и $k = 1$. Тогда мы меняем местами города на первой и пятой позициях и получаем мутированный маршрут

$$x_m = 6 \rightarrow 5 \rightarrow 4 \rightarrow 7 \rightarrow 1 \rightarrow 2 \rightarrow 3. \tag{18.51}$$

Взаимообмен также называется мутацией 2-exchange (2-обменной) [Beyer и Schwefel, 2002]. Мы могли бы обобщить этот метод путем взаимообмена последовательностей городов, а не отдельных городов. Тогда мы могли бы скомбинировать это обобщение с инверсией, изменив порядок одной или нескольких обмениваемых последовательностей.

18.5. Эволюционный алгоритм для задачи коммивояжера

С учетом предыстории предыдущих разделов теперь можно представить базовый эволюционный алгоритм для решения задачи коммивояжера, который показан на рис. 18.5.

```
p_m = скорость мутации
Инициализировать N кандидатных решений {x_i} (см. раздел 18.2).
Представить кандидатные решения, используя желаемое представление:
        в виде пути, смежности, порядкового числа или матрицы.
Рассчитать расстояние маршрута для кандидатного решения.
While not (критерий останова)
  For k = 1 to N
    Отобрать родителей из {x_i} для создания ребенка.
    Создать ребенка C_k, используя один из методов скрещивания,
      обсуждавшихся ранее:
    If используется представление в виде пути then
      Применить метод скрещивания из раздела 18.3.1 для создания C_k
    else if используется представление в виде смежности then
      Применить метод скрещивания из раздела 18.3.2 для создания C_k
    else if используется порядковое представление then
      Применить метод скрещивания из раздела 18.3.3 для создания C_k
    else if используется матричное представление then
      Применить метод скрещивания из раздела 18.3.4 для создания C_k
    End if
    r ← U[0, 1] (случайное число, равномерно распределенное между 0 и 1)
    If r < p_m then
      Мутировать C_k, используя один из методов из раздела 18.4.
    End if
    Рассчитать растояние маршрута для C_k
  Next ребенок
  Заменить дублирующих особей в {x_i} ∪ {C_i}
  {x_i} ← лучшие N особей из {x_i} ∪ {C_i}
Next поколение
```

Рис. 18.5. Эволюционный алгоритм решения задачи коммивояжера

У нас есть много вариантов реализации рис. 18.5.

- У нас есть несколько вариантов для инициализации популяции, как мы обсуждали в разделе 18.2.
- У нас есть несколько вариантов для скрещивания, как мы обсуждали в разделе 18.3. Мы можем также использовать более одного метода скрещивания, вероятностно переключаясь между разными методами из поколения в поколение. Кроме того, мы можем отслеживать, какой метод скрещивания дает лучшие результаты, и адаптировать частоту методов скрещивания в зависимости от приспособленности их ребенка.

- У нас есть несколько вариантов мутации, как мы обсуждали в разделе 18.4. Как и в случае со скрещиванием, мы можем использовать более одного метода мутации, вероятностно переключаясь между различными методами из поколения в поколение. Так же, как и со скрещиванием, мы можем отслеживать, какой метод мутации дает лучшие результаты, и адаптировать частоту методов мутации в зависимости от приспособленности их результатов.
- При реализации алгоритма нам нужно указать оператор «Отобрать родителей» на рис. 18.5. Мы можем использовать любой из методов отбора, описанных в разделе 8.7: отбор пропорциональной приспособленности, ранговый отбор, турнирный отбор и т. д.
- С такими задачами, как задача коммивояжера, где специфичная для конкретной задачи информация (расстояния между городами) легко доступна, мы часто можем получить гораздо более высокие результаты, объединив эволюционный алгоритм с алгоритмом, в котором используется информация о расстоянии. Например, после получения каждого ребенка C_k на рис. 18.5 мы могли бы отобрать случайный подмаршрут из C_k и перестроить его, используя любую из эвристик раздела 18.2, либо, если подмаршрут достаточно мал, использовать грубую силу [Jayalakshmi и соавт., 2001]. Эти типы подходов называются гибридными эволюционными алгоритмами, поскольку они сочетают стандартные операторы эволюционных алгоритмов с неэволюционными алгоритмами, специально разработанными для задачи коммивояжера. В литературе предлагается множество гибридных вариаций эволюционного алгоритма для задачи коммивояжера.
- Строка «Заменить дублирующих особей» на рис.18.5 часто необходима, потому что в комбинаторной оптимизации лучшие несколько кандидатных решений, как правило, доминируют над всей популяцией в целом, поскольку поисковое пространство дискретно, а не непрерывно. То есть популяция сходится так, что она состоит только из нескольких особей (иногда только из одного). Мы обсудили многообразие для непрерывной оптимизации в разделе 8.6, и в комбинаторной оптимизации многообразие требует еще большего внимания. Мы можем применить несколько методов замены дублирующих лиц. А также заменить их случайно сгенерированными особями; особями, созданными одной из эвристик раздела 18.2; мы можем их мутировать, используя один из методов раздела 18.4, либо применить некую комбинацию этих методов.

■ Пример 18.1

В данном примере мы исследуем эталонную задачу Berlin52, которая представляет собой множество из 52 местоположений в Берлине, Германия. Эта задача имеется на веб-сайте TSPLIB (см. приложение В.6). На рис. 18.6 показаны участок из 52 городов и замкнутый маршрут с минимальным расстоянием. Единицы широты и долготы нормализованы. Маршрут с минимальным расстоянием составляет 7542 единицы.

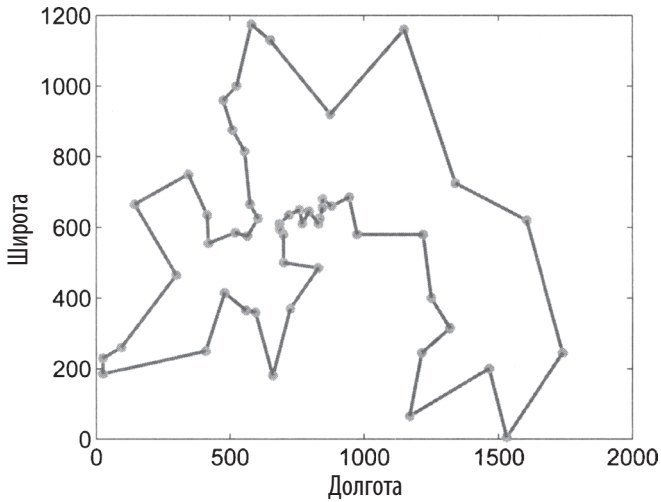


Рис. 18.6. Пример 18.1: города эталонной задачи Berlin52 и маршрут с минимальным расстоянием, который составляет 7542 единицы

В данном примере мы реализуем эволюционный алгоритм задачи коммивояжера на рис. 18.5 на эталонной задаче Berlin52 со следующими параметрами.

- Мы используем популяцию размером $N = 53$ (на один город больше, чем число городов).
- Мы инициализируем популяцию, генерируя N случайных маршрутов.
- Мы используем представление в виде пути.
- В каждом поколении мы определяем приспособленность $f(x)$ как

$$f(x) = \max_z D(z) + \min_z D(z) - D(x), \tag{18.52}$$

где максимум и минимум берутся по всей популяции кандидатных решений и $D(z)$ – это расстояние маршрута z. Это уравнение

трансформирует расстояние в приспособленность таким образом, что решение с малым расстоянием имеет высокое значение приспособленности, и наоборот. Данное уравнение отображает расстояние в приспособленность таким образом, что все значения приспособленности положительны.

- Мы отбираем родителей в алгоритме рис. 18.5, используя рулеточный отбор.
- Используем скрещивание с частичным совпадением (PMX) (см. раздел 18.3.1).
- Используем скорость мутации p_m = 5 % и мутацию на основе инверсии (см. раздел 18.4).
- Заменяем дублирующих особей с помощью двухэтапного процесса. Во-первых, мы сканируем родительскую/детскую популяции и мутируем дублирующих особей. Во-вторых, снова сканируем родительскую/детскую популяции и заменяем дубликаты случайными маршрутами.
- Мы выполняем 20 симуляций Монте-Карло, каждую для 300 поколений, и находим среднее расстояние D^* лучшего из 20 симуляций маршрута.

В данном примере мы проводим несколько экспериментов. Сначала испытываем пять разных методов скрещивания и получаем следующие результаты:

$$\begin{aligned} \text{скрещивание с частичным совпадением:} &\quad D^* = 8724 \\ \text{упорядоченное скрещивание:} &\quad D^* = 8393 \\ \text{цикличное скрещивание:} &\quad D^* = 9493 \\ \text{реорганизующее скрещивание:} &\quad D^* = 17109 \\ \text{с инверсией по опорному элементу:} &\quad D^* = 10595 \end{aligned} \tag{18.53}$$

Мы видим, что упорядоченное скрещивание работает лучше всего.

Во-вторых, мы применяем упорядоченное скрещивание и испытываем три разных метода мутации, получая следующие результаты:

$$\begin{aligned} \text{мутация на основе инверсии:} &\quad D^* = 8393 \\ \text{мутация на основе вставки:} &\quad D^* = 9776 \\ \text{мутация на основе взаимообмена:} &\quad D^* = 10036 \end{aligned} \tag{18.54}$$

Мы видим, что мутация на основе инверсии работает лучше всего.

В-третьих, мы используем упорядоченное скрещивание и мутацию на основе инверсии и испытываем три разных метода инициализации.

В первом методе мы инициализируем всю популяцию случайными маршрутами. Во втором методе инициализируем двух особей, используя инициализацию на основе ближайшего соседа (см. раздел 18.2.1), и случайно инициализируем оставшиеся (n – 2) городов. В третьем методе инициализируем всю популяцию, используя инициализацию на основе ближайшего соседа. Мы получаем следующие результаты.

$$\begin{aligned} N \text{ случайных маршрутов и 0 маршрутов} \\ \text{на основе ближайшего соседа:} \quad D^* &= 8393 \\ (N-2) \text{ случайных маршрутов и 2 маршрута} \\ \text{на основе ближайшего соседа:} \quad D^* &= 8140 \\ 0 \text{ случайных маршрутов и } N \text{ маршрутов} \\ \text{на основе ближайшего соседа:} \quad D^* &= 8115 \end{aligned} \tag{18.54}$$

Мы видим, что инициализация всей популяции на основе алгоритма ближайшего соседа работает лучше всего. Вместе с тем мы можем получить почти все преимущества инициализации на основе ближайшего соседа, инициализируя только пару особей с помощью алгоритма ближайшего соседа. На рис. 18.7 показаны результаты типичной симуляции эволюционного алгоритма с упорядоченным скрещиванием, мутацией на основе инверсии и инициализацией всех особей целиком алгоритмом ближайшего соседа.

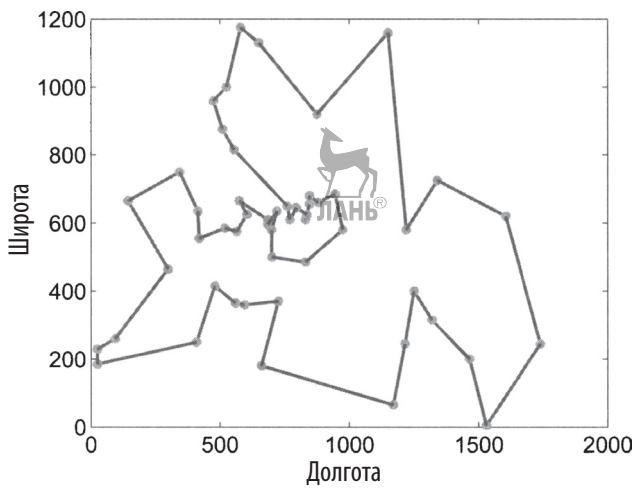


Рис. 18.7. Пример 18.1: типичный результат эволюционного алгоритма для эталонной задачи Berlin52. Показанное здесь расстояние составляет 8036 единиц, что на 6.5 % хуже оптимального решения

■

В примере 18.1 исследовалось несколько разных вариантов эволюционного алгоритма, но мы по-прежнему не смогли получить глобально-оптимальное решение. В каком-то смысле это неудивительно, так как мощность поискового пространства составляет порядка $51!/2 = 10^{66}$. В каждой симуляции эволюционного алгоритма мы выполнили 300 поколений с популяцией размером 53 особей. Следовательно, мы вычисляли $300 \times 53 = 15\,900$ потенциальных решений, что является незначительной, почти ничтожной частью поискового пространства, и мы все равно оказались в пределах менее 10 % оптимального решения. Однако эталон Berlin52 считается довольно легкой задачей коммивояжера. На рис. 18.6 показано расположение городов, и не похоже, что человеку или хорошей компьютерной программе будет очень сложно найти оптимальный маршрут. Эти результаты подчеркивают один из моментов, о которых мы говорили в конце предыдущего раздела и который мы вновь подчеркиваем здесь.

> *С такими задачами, как задача коммивояжера, где специфичная для конкретной задачи информация (расстояния между городами) легко доступна, мы часто можем получить гораздо более высокие результаты, объединив эволюционный алгоритм с алгоритмом, который использует информацию о расстоянии.*

Любая реализация эволюционного алгоритма для задачи коммивояжера должна серьезно относиться к этому совету. Исследователю необходимо изучить современные неэволюционные эвристики задачи коммивояжера и аккуратно их встраивать в эволюционный алгоритм, чтобы получить конкурентные результаты.

Наконец, для того чтобы получить хорошие результаты, нам нужно выполнить гораздо больше, чем 300 поколений каждой симуляции в примере 18.1. На рис. 18.8 показан типичный график решения задачи с минимальным расстоянием в эволюционно-алгоритмической популяции как функции от номера поколения. Данный график говорит о том, что лучшее кандидатное решение продолжает улучшаться даже после 300 поколений и что мы могли бы ожидать гораздо более высокой результативности, если бы дали эволюционному алгоритму поработать еще несколько сотен поколений. Реализации эволюционного алгоритма, которые надеются получить конкурентные результаты, обычно должны работать, по крайней мере, десятки тысяч поколений. Конечно, эволюционные алгоритмы с ограниченными вычислительными мощностями интересны и важны для получения решений задачи комми-

вояжера в реальном времени, но цена, которая будет уплачена за более высокую результативность, – это более худшие результаты.

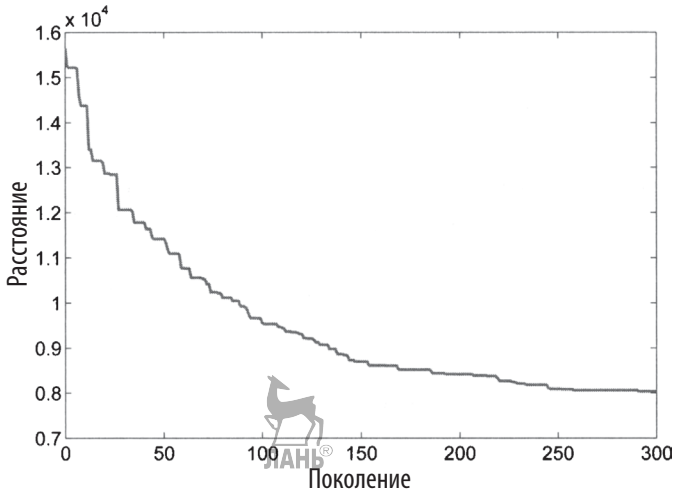


Рис. 18.8. Пример 18.1: типичное поведение при схождении эволюционного алгоритма для эталонной задачи Berlin52. Эволюционный алгоритм в основном сошелся после 300 поколений, но похоже, что лучшее кандидатное решение будет продолжать улучшаться еще несколько сотен поколений

18.6. Задача раскраски графа

Граф – это множество частично связных вершин. Каждая вершина имеет уникальный индекс и вес, который обычно не является уникальным [Pardalos и Mavridou, 1998]. На рис. 18.9 показан пример графа.

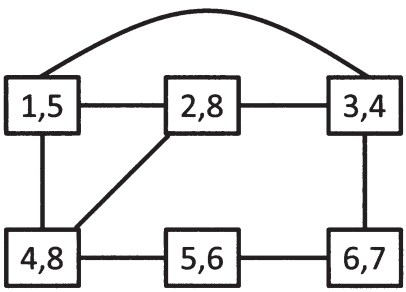


Рис. 18.9. Пример связного графа. Каждая вершина помечена (*i*, *w*), где *i* – уникальный индекс вершины и *w* – его вес

Существует много родственных, но отличающихся задач раскраски графов. Классическая задача раскраски графа определяется как:

1) определить наименьшее число цветов *n*, таких, что каждая вершина связного графа может быть окрашена одним из этих *n* цветов, при условии что связанным вершинам не назначен один и тот же цвет, либо
2) раскрасить каждую вершину в связном графе одним из *n* цветов, где *n* задано с ограничением, что связанным вершинам не назначен один и тот же цвет.

Заметим, что первая из перечисленных выше задач, которая также называется *n*-цветной задачей раскраски, может быть решена многократным решением второй задачи при последовательно уменьшающихся значениях *n*.

Задача раскраски взвешенных графов является обобщением второго приведенного выше определения. Задача раскраски взвешенного графа состоит в том, чтобы присвоить каждой вершине один из *n* цветов, при условии что связанным вершинам не назначен один и тот же цвет, так чтобы максимизировать сумму весов цветных вершин. В этом разделе мы будем делать акцент именно на данной задаче. Обратите внимание, что вторая приведенная выше классическая задача раскраски графа может быть решена путем назначения каждой вершине веса 1 и решения задачи раскраски взвешенного графа.

В задаче раскраски взвешенных графов приспособленность кандидатного решения равна сумме весов цветных вершин, и задача состоит в том, чтобы раскрасить вершины таким образом, чтобы максимизировать приспособленность. Задача взвешенной раскраски графов имеет приложения в планировании, компьютерных сетях, обнаружении и диагностике неисправностей, сопоставлении шаблонов, теории связи, играх и многих других областях [Ufuktepe и Bacak, 2005]. Когда мы сталкиваемся с практической оптимизационной задачей, если можем преобразовать ее в эквивалентную задачу раскраски графа, то для решения нашей практической оптимизационной задачи мы можем применить множество инструментов, предназначенных для задач раскраски графа.

Причина, по которой эти задачи называются *задачами раскраски графов*, или иногда *задачами раскрашивания карт*, заключается в том, что карта может быть представлена в виде связного графа[3]. В качестве

[3] Знаменитая теорема о четырех красках, или четырехкрасочная теорема, гласит, что, для того чтобы раскрасить любую карту таким образом, чтобы два связанных участка не были одинаковыми, требуется не более четырех цветов.

примера конвертирования карты в граф на рис. 18.10 показана карта, на которой каждый участок помечен индексом. Для того чтобы конвертировать карту в граф, сначала заметим, что на карте показано, что участок 1 имеет общую границу с участками 2, 4 и 5; поэтому на графе справа показано, что вершина 1 связана с вершинами 2, 4 и 5. Далее следует отметить, что на карте показано, что участок 2 граничит с участками 1, 3 и 4; поэтому граф справа показывает, что вершина 2 связана с вершинами 1, 3 и 4. Мы продолжаем этот процесс, конвертируя карту в эквивалентный связный граф. Мы видим, что задача раскрашивания карты n цветами так, чтобы соседние участки не имели одинаковые цвета, эквивалентна задаче раскраски графа. Обратите внимание, что обратное утверждение неверно, то есть связный граф не обязательно может быть конвертирован в плоскую карту. Например, полносвязанный граф с пятью вершинами не может быть конвертирован в плоскую карту.

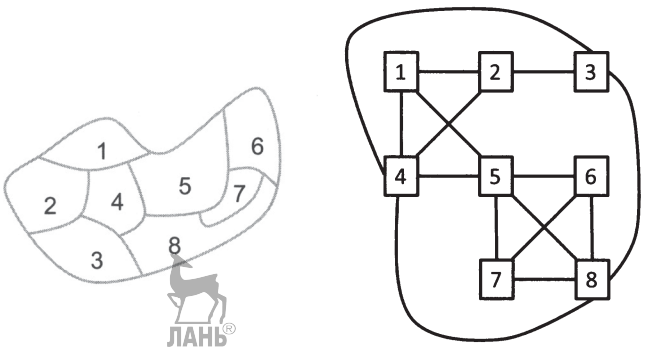


Рис. 18.10. Рисунок слева показывает карту. Рисунок справа показывает эквивалентный связный граф. Карта всегда может быть конвертирована в эквивалентный связный граф, но обратное утверждение неверно

Рассмотрим задачу раскраски одноцветного графа для графа на рис. 18.9. Вершины 2 и 4 имеют наибольший вес, но мы не сможем покрасить их обе в один цвет, потому что они связаны. Какие вершины мы должны покрасить, чтобы получить максимальную приспособленность? Одним из популярных алгоритмов является жадный алгоритм, который показан на рис. 18.11. Жадный алгоритм прост; он сортирует вершины в порядке уменьшения веса, а затем назначает первый допустимый цвет вершинам в отсортированном порядке.

С учетом графа рис. 18.9 жадный алгоритм сортирует вершины в порядке {2, 4, 6, 5, 1, 3}, хотя вершины 2 и 4 могут быть взаимозаменяемы-

ми, так как они имеют одинаковый вес. В случае одноцветной задачи мы окрашиваем вершины 2 и 6, в результате получая приспособленность 15. В случае двухцветной задачи (к примеру, красный и зеленый) мы присваиваем красный вершине 2, зеленый вершине 4, красный вершине 6 и зеленый вершине 3, в результате получая приспособленность 27.


```
{x_i} = N вершин, отсортированных в порядке уменьшающихся весов
{C_k} = n цветов
For i = 1 to N
    For k – 1 to n
        If допустимо, then назначить C_k вершине x_i и выйти из цикла «For k»
    Next цвет
Next вершина
```

Рис. 18.11. Жадный алгоритм для *N*-вершинной *n*-цветной задачи раскраски графа

Жадный алгоритм прост и быстр, и он часто работает довольно хорошо. Однако не слишком сложно найти случай, когда жадный алгоритм терпит неудачу. Например, рассмотрим граф на рис. 18.12. Когда мы применяем жадный алгоритм для одноцветной задачи для этого графа, мы окрашиваем только вершину 7, в результате получая приспособленность 5. Из графа видно, что мы можем получить более высокую приспособленность, раскрашивая вершины 1, 3 и 5, в результате получив приспособленность 9.

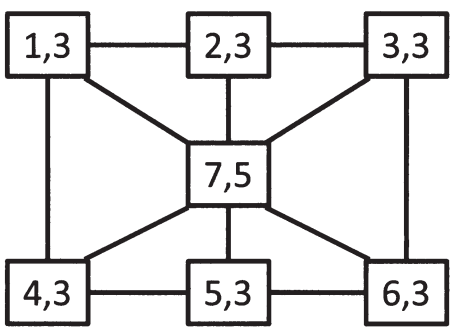


Рис. 18.12. Когда мы применяем жадный алгоритм для задачи раскраски одноцветного графа к этому графу, мы окрашиваем только вершину 7, получая в итоге неоптимальное решение

Эволюционные алгоритмы и задачи раскраски графов

Как применить эволюционный алгоритм для решения задачи раскраски графа? Одним из способов является определение особи в эволюционно-алгоритмической популяции в виде упорядоченного списка вершин. Затем мы можем применить жадный алгоритм для назначения цветов на основе порядка следования вершин. Каждая особь тогда будет иметь значение приспособленности. Мы можем использовать любой тип зависимого от приспособленности отбора, а затем для создания детей применить любой из методов рекомбинации раздела 18.3 и любой из методов мутации раздела 18.4. Этот подход преобразует задачу раскраски графа в формат задачи коммивояжера, что позволяет использовать все результаты данной задачи предыдущих разделов.

Как и в случае с задачей коммивояжера, любой серьезный эволюционный алгоритм раскраски графов для получения хороших результатов должен включать в себя неэволюционные эвристики. Существует много литературы, посвященной задаче раскраски графов. Публикация [Jensen и Toft, 1994] послужит неплохой основой по данной задаче, анализу, теоретическим результатам и некоторым эвристическим алгоритмам. Кроме того, существует множество других эволюционно-алгоритмических подходов к решению задач раскраски графов. Гибридные эволюционные алгоритмы, сочетающие в себе эволюционный поиск с локальной оптимизацией, являются одними из наиболее эффективных алгоритмов раскраски графов. Обратитесь к публикации [Galinier и соавт., 2013] для получения более полного обзора данной темы.

■ Пример 18.2

В этом примере показано, как задача планирования может быть представлена в виде задачи раскраски графа. Предположим, мы хотим запланировать события 1, 2, 3, 4, 5 и 6, чтобы следующие ниже пары событий не происходили одновременно:

(1 и 2), (1 и 3), (3 и 5), (3 и 6) и (4 и 6).

Мы можем представить эту задачу в виде графа рис. 18.13. Каждая вершина имеет одинаковый вес, и поэтому веса на рисунке не показаны. Вершины, соответствующие событиям, которые не могут быть запланированы одновременно, в графе соединены между собой. Этот граф не имеет одноцветного решения. Однако мы можем получить трехцветное решение, раскрасив вершины 1, 5 и 6 одним цветом, и вершину 2 другим цветом, и вершину 3 и 4 третьим цветом. Другими словами, мы можем

планировать события 1, 5 и 6 в течение первого интервала, и событие 2 в течение второго интервала, и события 3 и 4 в течение третьего интервала времени. Мы видим, что можем использовать алгоритмы раскраски графов для планирования операций на производственном предприятии, где определенные задачи используют одни и те же ресурсы, для планирования занятий в школе, где определенные учащиеся и учителя участвуют в нескольких учебных занятиях, и для решения целого ряда других задач планирования.

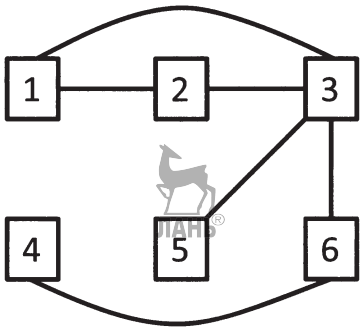


Рис. 18.13. Мы можем представить задачу планирования примера 18.2 с помощью данного графа

■

18.7. Заключение

Мы обобщили задачу коммивояжера и обсудили несколько представлений и операторов, наиболее часто используемых в задаче коммивояжера. Мы также обсудили задачу раскраски графа и показали, как ее можно конвертировать в задачу коммивояжера. Исследователи предлагают целый ряд операторов задачи коммивояжера, которые мы не имели времени охватить в этой главе. Публикация [Larrañaga и соавт., 1999b] дает хороший обзор представлений и операторов задачи коммивояжера. Задача коммивояжера имеет длинную историю, и исследователи решали ее с использованием многих методов, за исключением эволюционных алгоритмов. Задаче коммивояжера посвящен ряд хороших книг, в том числе [Applegate и соавт., 2007] и [Lawler и соавт., 1985]. Обратитесь к публикации [Hao and Middendorf, 2012] относительно материалов конференции, посвященной эволюционным алгоритмам для комбинаторных задач.

В этой главе обсуждались только две комбинаторно-оптимизационные задачи (задача коммивояжера и задача раскраски графов), но есть и многие другие популярные и широко применяемые комбинаторные оптимизационные задачи. К ним относятся задача о нахождении минимального остовного дерева, задача об оптимальном использовании ресурсов при производственном планировании, задача о ранце и задача об упаковке в контейнеры. Эволюционные алгоритмы нашли свое применение во всех этих задачах, но существуют самые широкие возможности для дополнительных исследований. Некоторые новые эволюционные алгоритмы и вариации эволюционных алгоритмов еще предстоит применить к некоторым из этих комбинаторно-оптимизационных задач. Эффективные способы гибридизации эволюционных алгоритмов с неэволюционными комбинаторными эвристиками могут стать плодотворной областью для будущих исследований. Наконец, нам нужно больше теоретических результатов, которые квантифицируют результативность эволюционных алгоритмов на комбинаторных задачах и которые могут служить руководством для практических приложений.

Наше обсуждение задач коммивояжера в этой главе охватывало только симметричные задачи, то есть задачи, в которых расстояние из города *i* до города *k* такое же, что и из города *k* до города *i*. В приложении С.6 рассматривается несколько других типов задачи коммивояжера, в том числе асимметричная задача коммивояжера, задача с последовательным упорядочиванием, задача ограниченной по емкости маршрутизации транспортных средств и задача гамильтонова пути.

Еще одной интересной вариацией является задача коммивояжера с прохождением на достаточно близком расстоянии от вершин (close-enough TSP). В этой задаче дан связный граф, в котором каждая вершина *i* имеет «достаточно близкий» связанный с ней радиус r_i окрестности. Задача состоит в нахождении минимального расстояния гамильтонова цикла, проходящего в пределах r_i окрестных блоков каждой вершины *i* [Yuan и соавт., 2007]. Данная задача тесно связана с задачей коммивояжера, но на самом деле она является непрерывной оптимизационной задачей, хотя и включает комбинаторные элементы. Наконец, мы упомянем задачу для машины Дубинса, которая является задачей коммивояжера для транспортного средства с кинематическими ограничениями. Например, транспортному средству вполне может потребоваться посетить множество мест, при этом проехав минимально возможное расстояние, при условии что оно не может мгновенно менять направление движения [Savla и соавт., 2008].

Задачи

Письменные упражнения

18.1. Какой маршрут и расстояние задачи коммивояжера получатся в результате инициализации на основе ближайшего соседа с учетом матрицы расстояний уравнения (18.5), при условии что мы начнем в городе 3?

18.2. Предположим, что вам дана задача коммивояжера для n городов и вы хотите определить M маршрутов в исходной популяции, используя стратегию на основе ближайшего соседа, описанную в разделе 18.2.1. Какова вероятность того, что с этой стратегией вы выберете M разных стартовых городов? Какова вероятность, если $n = 100$ и $M = 10$?

18.3. Сформулируйте незамкнутую задачу коммивояжера для пяти городов таким образом, чтобы алгоритм инициализации на основе ближайших двух соседей, описанный в разделе 18.2.1, работал лучше, чем алгоритм инициализации на основе ближайшего соседа.

18.4. На втором шаге примера инициализации на основе кратчайшего ребра в разделе 18.2.2 мы должны были сделать случайный отбор, так как два ребра имели одинаковую длину. Предположим, что в данном примере мы выбрали другой вариант. Каким будет заключительный замкнутый маршрут, и как общее расстояние будет соотноситься с примером в разделе 18.2.2?

18.5. На втором шаге примера инициализации на основе вставки в разделе 18.2.3 мы должны были сделать случайный отбор, так как два города находились на одинаковом расстоянии от T. Предположим, что в данном примере мы выбрали другой вариант. Каким будет заключительный замкнутый маршрут?

18.6. Незамкнутый маршрут $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$. Каким будет представление в виде пути, представление в виде смежности, порядковое представление и матричное представление этого маршрута?

18.7. Каков ранг матричного представления маршрута задачи коммивояжера?

18.8. Мы использовали жадный алгоритм раскраски графов для решения двухцветной задачи на рис. 18.9, отсортировав вершины в порядке {2, 4, 6, 5, 1, 3}, присвоив первый цвет вершинам 2 и 6

и второй цвет вершинам 4 и 3, получив в результате приспособленность 27. Однако мы также можем отсортировать вершины в порядке {4, 2, 6, 5, 1, 3}, так как вершины 2 и 4 имеют одинаковый вес. Какое назначение цвета приведет к этому порядку, и какова будет приспособленность?

18.9. Объясните, каким образом головоломку судоку с игровым полем 9×9 можно сформулировать как задачу раскраски графа.
Подсказка: граф будет иметь 81 вершину.

18.10. Рассмотрим задачу раскраски графа для графа в левой части рис. 18.14, где каждая вершина имеет одинаковый вес.

а) Используйте жадный алгоритм раскраски графа, чтобы найти минимальное число цветов, необходимых для раскраски всех вершин, где вершины упорядочены, как показано на рисунке.

б) Повторите для графа в правой части рисунка, который является тем же графом, за исключением того, что его вершины упорядочены по-другому.

Рис. 18.14. Задача 18.10: связные графы слева и справа эквивалентны. Единственное различие заключается в порядке следования вершин

Компьютерные упражнения

18.11. Повторите пример 18.1, используя одну из других задач коммивояжера на веб-сайте TSPLIB.

18.12. Повторите пример 18.1 с упорядоченным скрещиванием, случайной инициализацией всей популяции целиком и оппозиционной логикой, схематично показанной на рис. 16.10, для мутации. Каково расстояние лучшего найденного решения, усредненно по 20 симуляциям Монте-Карло? Как этот результат соотносится с результатами, полученными с использованием трех методов мутации, используемых в примере 18.1?

Глава 19

Ограниченная оптимизация

Необходимо найти способы встраивания ограничений (обычно существующих в любом реальном приложении) в функцию приспособленности.

– Карлос А. Коэльо Коэльо [Coello Coello, 2002]

Все реальные оптимизационные задачи ограничены, по крайней мере, если не эксплицитно, то имплицитно. В этой главе рассматриваются различные подходы к обработке ограничений в оптимизационных задачах. Ограниченная оптимизационная задача может быть записана в виде:

$$\min_x f(x), \text{ такая что } g_i(x) \le 0 \text{ для } i \in [1, m] \text{ и } h_j(x) = 0 \text{ для } j \in [1, p] \tag{19.1}$$

Такая задача включает в себя ограничения $(m + p)$, m из которых являются ограничениями в виде неравенства и p – ограничениями в виде равенства. Множество x, удовлетворяющее всем ограничениям $(m + p)$, называется допустимым множеством, а множество x, нарушающее одно или несколько ограничений, называется недопустимым множеством[1]:

$$\begin{aligned} &\text{допустимое множество } \mathcal{F} = \{x : g_i(x) \le 0 \text{ для } i \in [1, m] \text{ и } h_j(x) = 0 \text{ для } j \in [1, p]\} \\ &\text{недопустимое множество } \overline{\mathcal{F}} = \{x : x \notin \mathcal{F}\} \end{aligned} \tag{19.2}$$

[1] В оригинале используются прилагательные соответственно feasible и infeasible. – *Прим. перев.*

Мы используем термин *ограниченные эволюционные алгоритмы* для обозначения эволюционных алгоритмов, которые предназначены для решения задач формы уравнения (19.1)[2].

Краткий обзор главы

Ограниченные эволюционные алгоритмы могут быть разделены на несколько разных категорий.

1. *Подходы на основе штрафной функции* модифицируют стоимостную функцию эволюционно-алгоритмической особи x на основе некой меры нарушения ее ограничений. Штрафные функции, которые позволяют, а иногда даже поощряют, наличие недопустимых решений в популяции, называются внешними подходами. В этом случае они накладывают штраф на стоимость или отбор недопустимых кандидатных решений. Штрафные функции, которые не разрешают наличия недопустимых решений в популяции, называются методами внутренней точки. В разделе 19.1 обсуждаются общие способы реализации штрафных функций. Мы покажем, как реализовывать разные подходы к штрафным функциям в ограниченных эволюционных алгоритмах, в разделе 19.2.
2. *Специальные представления* – это зависимые от конкретной задачи подходы для представления ограниченных задач таким образом, чтобы представление было неограниченным, при этом оставляя кандидатные решения ограниченными. Специальные операторы – это зависимые от конкретной задачи подходы для выполнения отбора, рекомбинации и мутации таким образом, чтобы ограничения всегда удовлетворялись детскими особями. Эти два подхода не разрешают наличия недопустых кандидатных решений в популяции. Мы обсудим эти подходы в разделе 19.3.
3. *Исправительные алгоритмы* модифицируют недопустимых эволюционно-алгоритмических особей так, что они становятся допустимыми. Эти алгоритмы в значительной степени зависят от задачи. Они могут разрешать отдельным недопустимым особям оставаться в популяции и вместе с тем исправлять другие недопустимые особи. Единственный исправительный метод, который

[2] Термин *ограниченные эволюционные алгоритмы* не совсем корректен с грамматической точки зрения. Грамматически строгая интерпретация фразы будет означать, что она относится к эволюционным алгоритмам, которые ограничены, а не к эволюционным алгоритмам, которые предназначены для решения ограниченных задач. Но термин удобен, лаконичен и популярен, поэтому с этой оговоркой мы уверены, что читателя не смутит его смысл.

мы обсудим в этой главе, – это генетический алгоритм для численной оптимизации ограниченных задач Genocop раздела 19.3.2.

4. *Гибридные методы* сочетают в себя элементы из вышеперечисленных методов или из алгоритмов неэволюционной ограниченной оптимизации. Например, во многих эволюционных алгоритмах используется один из вышеперечисленных методов наряду с локальным поиском. В этой главе мы представим несколько основных подходов к ограниченным эволюционным алгоритмам, но не будем касаться того, как они могут быть гибридизированы. Вместе с тем литература содержит много примеров гибридизации. После того как читатель ознакомится с основными подходами к ограниченным эволюционным алгоритмам в этой главе, он будет хорошо подготовлен, для того чтобы заняться изучением гибридных алгоритмов в специализированной литературе либо взять лучшие свойства разных ограниченных эволюционных алгоритмов для разработки своего собственного гибридного алгоритма.

Мы не претендуем на то, чтобы охватить все методы работы с ограничениями, которые были предложены на протяжении многих лет, но в разделе 19.4 излагается несколько других подходов к непрерывной оптимизации, включая использование культурных алгоритмов и многокритериальной оптимизации.

Одна из основных задач, которую нам необходимо решить при использовании алгоритма ограниченной оптимизации, заключается в ранжировании кандидатных решений. Некоторые из решений имеют высокую стоимость, но удовлетворяют ограничениям, в то время как другие решения имеют низкую стоимость, но нарушают ограничения. В разделе 19.5 обобщаются подходы к ранжированию, представленные ранее в этой главе, и обсуждается несколько альтернативных методов ранжирования. Раздел 19.6 связывает вместе весь материал данной главы и демонстрирует сравнение разных алгоритмов ограниченной биогеографической оптимизации (BBO) на нескольких эталонах.

19.1. Подходы на основе штрафной функции

Подходы на основе штрафной функции штрафуют кандидатные решения, которые нарушают ограничения или придвигаются близко к нарушению ограничений. Подходы на основе штрафных функций для общих ограниченных оптимизационных задач были впервые предло-

жены Ричардом Курантом в 1943 году [Courant, 1943]. Они часто упоминаются как наиболее популярные ограниченные оптимизационные алгоритмы, хотя другие подходы к ограниченным эволюционным алгоритмам быстро набирают популярность.

Мы можем разработать метод на основе штрафной функции двумя разными способами. Во-первых, мы можем штрафовать допустимых особей, когда они подходят ближе к границе ограничения; такие методы называются методами внутренней точки, или барьерными методами. Подход, который мы кратко обсудим в разделе 19.1.1, не разрешает наличия каких-либо недопустимых особей в популяции.

Во-вторых, мы можем разрешить наличие недопустимых особей в популяции, но штрафовать их с точки зрения стоимости или с точки зрения отбора для вклада в следующее поколение. Этот подход, несколько примеров которого мы приводим в разделе 19.1.2, как правило, не штрафует допустимых особей, независимо от того, насколько они близки к границе ограничения. Такие подходы называются внешними методами.

19.1.1. *Методы внутренней точки*

Методы внутренней точки (interior point method) разрешают наличие только допустимых особей в популяции. Данные методы штрафуют стоимость особей по мере их приближения к границе ограничения, тем самым поощряя этих особей оставаться в рамках ограничений. Проиллюстрируем идею методов внутренней точки на простом примере.

■ Пример 19.1

Рассмотрим скалярную задачу

$$\min_x f(x), \quad \text{такая что } x \geq c, \quad \text{где } f(x) = x^2. \tag{19.3}$$

Мы можем модифицировать эту задачу так, чтобы допустимые значения x штрафовались по мере приближения к ограничению. Модифицированная функция называется барьерной. Например, мы можем конвертировать ограниченную задачу уравнения (19.3) в неограниченную задачу

$$\min f'(x), \quad \text{где } f'(x) = x^2 + (x - c + \delta)^{-\alpha}, \tag{19.4}$$

где $\sigma > 0$ – это малая константа, а $\alpha > 0$ – еще одна константа. По мере того как $\alpha \to 0$, $\arg\min_x f'(x) \to \arg\min_x f(x)$, но мы также получаем $f'(x)$, которая ведет себя еще хуже, то есть $f'(x)$ становится менее гладкой.

На рис. 19.1 показаны $f(x)$ и $f'(x)$ для $c = 1$, $\delta = 0.01$ и $\alpha = 1$. Разумеется, в случае этого простого примера мы все еще должны убедиться, что $x \geq c$, поэтому наш подход с внутренней точкой не особо помог. Однако этот пример иллюстрирует, как методы внутренней точки способны мешать допустимым особям в эволюционном алгоритме нарушать ограничения после рекомбинации или мутации.

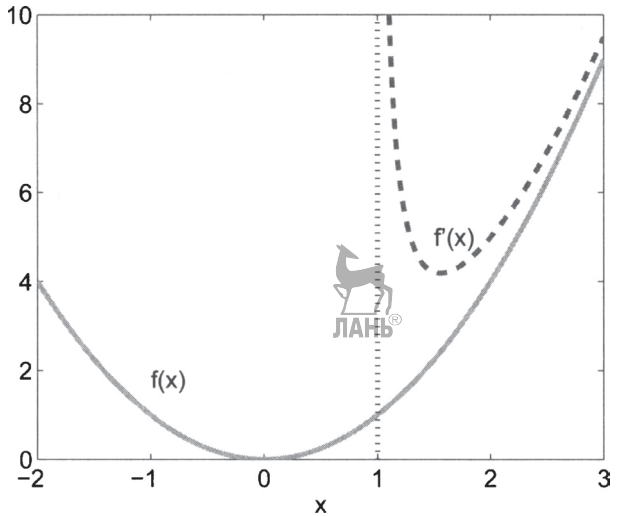


Рис. 19.1. Пример 19.1. Мы хотим минимизировать $f(x)$ так, что $x \geq 1$. Используем метод внутренней точки для конвертации ограниченной минимизации $f(x)$ в неограниченную минимизацию $f'(x)$

■

Методы внутренней точки используются в ограниченных эволюционных алгоритмах не очень часто. Это связано с тем, что для многих ограниченных оптимизационных задач отыскание кандидатных решений, удовлетворяющих всем ограничениям, само по себе является сложной задачей. Кроме того, недопустимые решения могут содержать информацию, ценную для поиска ограниченного оптимума. Например, задача с небольшим допустимым участком вполне может быть решена путем комбинирования двух недопустимых особей (см. рис. 19.2).

Вместе с тем существует также ряд оптимизационных задач, для которых относительно легко найти допустимые кандидатные решения. Поэтому малочисленность методов внутренней точки для ограниченных эволюционных алгоритмов удивляет ввиду обширной литературы по методам внутренней точки для универсальных оптимизационных

алгоритмов [Wright, 1987]. Это может указывать на существование запущенной исследовательской области ограниченных эволюционных алгоритмов.

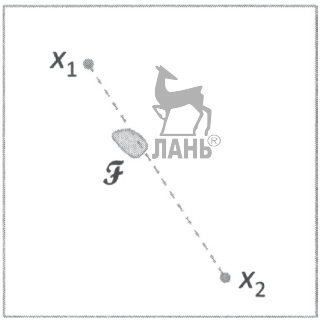


Рис. 19.2. Пример малого допустимого множества $\mathcal{F}$ в большом поисковом пространстве. Возможно, будет трудно найти кандидатное решение $x \in \mathcal{F}$ непосредственным образом, но, вероятно, будет гораздо легче найти двух недопустимых особей x_1 и x_2, которые могут скомбинироваться для порождения допустимой особи

19.1.2. *Внешние методы*

Внешние методы разрешают наличие недопустимых особей в популяции, но штрафуют их стоимость либо вероятность отбора. В этом разделе представлен широкий обзор внешних методов ограниченной оптимизации.

19.1.2.1. Подходы на основе смертной казни

Подходы на основе смертной казни (death penalty approach) – это внешние методы, которые разрешают наличие недопустимых особей в популяции, но только в течение коротких периодов времени. Подход на основе смертной казни доводит метод на основе штрафной функции до крайности. При таком подходе мы немедленно удаляем из популяции любую недопустимую особь $\bar{x}$. Если мы получаем $\bar{x}$ путем рекомбинации, то мы ее отбраковываем и повторяем операцию рекомбинации до тех пор, пока не получим допустимую особь. Если мы получаем $\bar{x}$ путем мутации, то мы ее отбраковываем и повторяем операцию мутации до тех пор, пока не получим допустимую особь.

Смертная казнь является удобным подходом к ограниченной оптимизации. Она имеет то преимущество, что не требует оценивания стоимости недопустимых особей, что в итоге может сэкономить вычис-

лительные усилия. Вместе с тем для многих задач изначально трудно получить допустимых особей, поэтому отбраковывание недопустимых особей, возможно, будет чрезмерно жесткой мерой. Вместо того чтобы полностью отбраковывать недопустимых особей, нам, возможно, придется сохранить их в популяции, давая относительно более низкую стоимость тем, кто нарушает ограничения менее серьезно, тем самым поощряя популяцию двигаться в направлении допустимого участка. Таким образом, эффективность применения смертной казни зависит от конкретной задачи.

19.1.2.2. Подходы без смертной казни

В остальной части этой главы рассматриваются подходы к обработке ограничений без смертной казни. Эти подходы являются более щадящими внешними методами, чем подходы, предусматривающие смертную казнь, поскольку они разрешают недопустимым особям оставаться в популяции в течение всего времени работы эволюционного алгоритма. Мы трансформируем стандартную ограниченную оптимизационную задачу уравнения (19.1) в следующую ниже неограниченную задачу:

$$\begin{aligned} &\min_x \phi(x), \quad \text{где} \quad \phi(x) = f(x) + \sum_{i=1}^{m} r_i G_i(x) + \sum_{j=1}^{p} c_j L_j(x) \\ &G_i(x) = [\max(0, g_i(x))]^{\beta} \\ &L_j(x) = |h_j(x)|^{\gamma} \end{aligned} \tag{19.5}$$

где r_i и c_j – это положительные константы, которые называются штрафными сомножителями (факторами), β и γ – положительные константы, которые часто принимаются равными 1 или 2. $\phi(x)$ называется функцией оштрафованной стоимости, при этом мы получаем $\phi(x)$ как взвешенную сумму функции первоначальной стоимости $f(x)$ и размеров нарушений ограничений $\{G_i(x)\}$ и $\{L_j(x)\}$. Мы видим, что если $x \in \mathcal{F}$, то $\phi(x) = f(x)$. Однако если $x \notin \mathcal{F}$, то $\phi(x) > f(x)$ на величину, которая увеличивается вместе с величиной нарушения ограничений.

Теперь, когда у нас есть функция оштрафованной стоимости $\phi(x)$, мы можем выполнить эволюционный алгоритм, который использует $\phi(x)$ в качестве стоимостной функции для отбора особей для следующего поколения. Поэтому мы можем распространить любой из неограниченных эволюционных алгоритмов, обсуждаемых в этой книге, на ограниченную оптимизацию. В качестве стоимостной функции вместо $f(x)$ просто применяем $\phi(x)$.

Ограничения $h_j(x) = 0$ очень суровы. Если мы случайно сгенерируем исходную популяцию в непрерывной поисковой области, то у нас будет

практически нулевой шанс получить особей, удовлетворяющих ограничениям в виде равенства. Поэтому мы часто меняем жесткие ограничения в виде равенства на мягкие ограничения, которые требуют, чтобы $h_j(x)$ равнялись нулю приближенно, а не строго. В результате получим

$$|h_j(x)| \leq \epsilon, \tag{19.6}$$

где ϵ – малая положительная константа. Это эквивалентно двум ограничениям

$$\begin{aligned} h_j(x) - \epsilon &\leq 0 \\ -h_j(x) - \epsilon &\leq 0 \end{aligned} \tag{19.7}$$

В зависимости от значения ϵ у нас есть разумный шанс получить особей, удовлетворяющих мягкому ограничению уравнения (19.6). Одним из способов задавать значение ϵ является использование относительно больших значений ϵ в начале работы эволюционного алгоритма, для того чтобы можно было получить несколько допустимых особей, а затем постепенно уменьшать ϵ по мере увеличения числа поколений [Brest, 2009], [Zavala и соавт., 2009]. Во многих исследовательских работах, в которых выполняется сопоставительный анализ ограниченных оптимизационных алгоритмов на эталонных функциях, используется $\epsilon = 0.0001$ [Liang и соавт., 2006].

В результате конвертации ограничений в виде равенства в ограничения в виде неравенства уравнение (19.5) трансформируется в

$$\min_x \phi(x), \quad \text{где} \quad \phi(x) = f(x) + \sum_{i=1}^{m+p} r_i G_i(x)$$

$$G_i(x) = \begin{cases} [\max(0, g_i(x))]^\beta & \text{для } i \in [1, m] \\ [\max(0, |h_i(x)| - \epsilon)]^\beta & \text{для } i \in [m+1, m+p] \end{cases} \tag{19.8}$$

где мы упростили задачу, приняв $\gamma = \beta$. Задачи, подобные задаче в уравнении (19.8), могут быть решены статическими либо динамическими методами. К статическим методам используются значения r_i, β и ϵ, не зависящие от номера эволюционно-алгоритмического поколения.

Напротив, в динамических методах используются значения r_i, β и ϵ, зависящие от номера эволюционно-алгоритмического поколения. Статические методы проще в реализации, но динамические могут работать лучше из-за их гибкости. Динамические методы могут быть способны разумно адаптировать свои веса на основе распределения популяции или характеристик задачи, тем самым повышая результативность. Ди-

намические методы часто увеличивают r_i и β и уменьшают ϵ по мере увеличения числа поколений. В результате вес, придаваемый нарушению ограничения, увеличивается, что приводит к постепенному привлечению все более и более недопустимых особей в сторону допустимого участка.

19.2. Популярные методы обработки ограничений

В этом разделе обсуждается несколько популярных подходов к обработке ограничений, используемых в эволюционных алгоритмах. Все эти подходы не предусматривают смертной казни.

19.2.1. *Статические штрафные методы*

Уравнение (19.8) предлагается в публикации [Homaifar и соавт., 1994] с $\beta = 2$ и r_i, являющейся функцией размера нарушения ограничений. То есть r_i является неубывающей функцией от $G_i(x)$. Иногда штрафной сомножитель r_i принимается равным одному из множества дискретных значений в зависимости от величины нарушения ограничений:

$$r_i = \begin{cases} R_{i1} & \text{если } G_i(x) \in [0, T_{i1}] \\ R_{i2} & \text{если } G_i(x) \in [T_{i1}, T_{i2}] \\ \vdots & \\ R_{iq} & \text{если } G_i(x) \in [T_{i,q-1}, \infty] \end{cases} \tag{19.9}$$

где q – это задаваемое пользователем число уровней ограничения, значения R_{ij} – задаваемые пользователем веса, а значения T_{ij} – задаваемые пользователем пороги ограничения. Статическим данный подход является потому, что штраф на ограничениях не является функцией числа поколений. В исследовательской литературе часто критикуют этот хорошо известный подход, поскольку он требует многих регулировочных параметров. Он требует $(2q - 1)(m + p)$ регулировочных параметров, правда, мы можем сократить это число, объединив некоторые весовые уровни и пороги, тем самым упростив алгоритм.

19.2.2. *Превосходство допустимых точек*

Метод превосходства допустимых точек (superiority of feasible points) [Powell и Skolnick, 1993] модифицирует функцию оштрафованной стоимости уравнения (19.8) следующим образом:

$$\min_x \phi'(x), \quad \text{где } \phi'(x) = \phi(x) + \theta(x)$$
$$= f(x) + \sum_{i=1}^{m+p} r_i G_i(x) + \theta(x), \qquad (19.10)$$

где $\theta(x)$ – это дополнительный член, который предназначен для того, чтобы гарантировать, что $\phi'(x) \le \phi'(\bar{x})$ для всех $x \in \mathcal{F}$ и для всех $\bar{x} \notin \mathcal{F}$. То есть $\phi'(x) \le \phi'(\bar{x})$ для всех допустимых x и для всех недопустимых $\bar{x}$. Этого можно достичь, определив $\theta(x)$ следующим образом:

$$\theta(x) = \begin{cases} 0 & \text{если } x \in \mathcal{F} \\ \max f(y) : y \in \mathcal{F} & \text{если } x \notin \mathcal{F} \end{cases} \qquad (19.11)$$

приняв, что $f(x) \ge 0$ для всех x. Менее консервативный способ [Michalewicz и Schoenauer, 1996] состоит в реализации данного метода, который не исходит из $f(x) \ge 0$, состоит в том, чтобы определить $\theta(x)$ следующим образом:

$$\theta(x) = \begin{cases} 0 & \text{если } x \in \mathcal{F} \\ \max [0, \max_{y \in \mathcal{F}} f(y) - \min_{y \notin \mathcal{F}} \phi(y)] & \text{если } x \notin \mathcal{F} \end{cases} \qquad (19.12)$$

Данное определение $\theta(x)$ дает $\phi'(x) = \phi(x)$ для всех x при условии, что $\phi(x) \le \phi(\bar{x})$ для всех $x \in \mathcal{F}$ и для всех $\bar{x} \notin \mathcal{F}$. То есть если функция оштрафованной стоимости уравнения (19.8) приводит к тому, что все допустимые особи ранжируются лучше, чем все недопустимые, то мы не делаем никаких изменений в уравнении (19.8). Однако если уравнение (19.8) приводит к $\phi(x) > \phi(\bar{x})$ для некоторого $x \in \mathcal{F}$ и для некоторого $\bar{x} \notin \mathcal{F}$, то уравнение (19.12) сдвигает значения функции оштрафованной стоимости у всех недопустимых особей так, что $\min_{\bar{x}} \phi'(\bar{x}) = \max_x \phi'(x)$, то есть лучшая недопустимая оштрафованная стоимость равна худшей допустимой оштрафованной стоимости.

Метод превосходства допустимых точек может быть привлекательным подходом, если оптимизационная задача сопряжена со сложными ограничениями. Если ограничения трудно удовлетворить, то этот метод обеспечивает большое селекционное давление на то, чтобы допустимые точки оставались в популяции, а это, в свою очередь, позволяет переносить их информацию в следующее поколение.

19.2.3. *Эклектичный эволюционный алгоритм*

Эклектичный эволюционный алгоритм (eclectic EA) предлагает еще один подход к обеспечению превосходства допустимых точек [Morales

и Quezada, 1998]. Эклектичный эволюционный алгоритм определяет функцию оштрафованной стоимости как

$$\phi(x) = \begin{cases} f(x) & \text{если } x \in \mathcal{F} \\ K\left(1 - \dfrac{s(x)}{m+p}\right) & \text{если } x \notin \mathcal{F} \end{cases} \tag{19.13}$$

где K – это большая константа, $m + p$ – общее число ограничений и $s(x)$ – число ограничений, которые удовлетворяются x. Определяемая пользователем константа K должна быть достаточно большой, чтобы гарантировать, что $\phi(\bar{x}) > \phi(x)$ для всех $\bar{x} \notin \mathcal{F}$ и для всех $x \in \mathcal{F}$. Если мы применяем метод ранжирования для отбора особей для их последующей рекомбинации, то нет никакой верхней границы для K. Однако если мы применяем рулеточный метод либо какой-либо другой метод, который для отбора использует абсолютные значения $\phi(\cdot)$, то мы должны проявлять осторожность, не установив K слишком большой; мы хотим гарантировать, что, несмотря на то что недопустимые особи получают ранг хуже, чем у допустимых, у недопустимых особей по-прежнему оставался разумный шанс быть отобранными для рекомбинации.

Эклектический эволюционный алгоритм отличается от уравнения (19.10) тем, что не вычисляет стоимость $f(x)$ для недопустимых особей, что в итоге может привести к значительной вычислительной экономии. Кроме того, при определении функции оштрафованной стоимости эклектичный эволюционный алгоритм учитывает только число нарушений ограничений и не учитывает размер нарушения ограничений. Уравнение (19.10), с другой стороны, учитывает только размер нарушения ограничений и не учитывает число нарушений ограничений. Это может придать еще одно вычислительное преимущество эклектичному эволюционному алгоритму, поскольку в реальных задачах часто намного проще подсчитать число нарушений ограничений, чем квантифицировать точный уровень этих нарушений.

19.2.4. *Коэволюционные штрафы*

Было бы интересно объединить подходы уравнений (19.10) и (19.13), потому что иногда для нас может быть важен размер нарушений ограничений, но в других случаях число нарушений ограничений может оказаться важнее. Коэволюционный подход, который включает в себя эту идею, предлагается в публикациях [Coello Coello, 2000b], [Coello Coello, 2002] и использует отштрафованную стоимость

$$\phi(x) = f(x) + w_1 \sum_{i=1}^{m+p} G_i(x) + w_2 \left(1 - \frac{s(x)}{m+p}\right), \tag{19.14}$$

где w_1 и w_2 – это веса. Данный подход называется коэволюционным, потому что он затрагивает две популяции. Одна популяция, P_1, состоит из кандидатных решений x и эволюционирует в соответствии с оштрафованной стоимостью $\phi(x)$. Вторая популяция, P_2, состоит исключительно из пар (w_1, w_2). Особь в P_1 эволюционирует, используя специфическую особь из P_2 (то есть специфическую пару (w_1, w_2). Стоимость пары (w_1, w_2) оценивается как

$$\begin{aligned} \psi(w) &= 1/M_1(w) \sum_{x \in \mathcal{F}} \phi(x) - M_1(w) \\ M_1(w) &= |x : x \in \mathcal{F}| \end{aligned} \tag{19.15}$$

где w относится к конкретной паре (w_1, w_2) из P_2. Обратите внимание, что $M_1(w)$ – это число допустимых особей в P_1, после того как она завершила эволюционировать с использованием w. Стоимость $\psi(w)$ особи w зависит от средней оштрафованной стоимости всех допустимых особей, которых она эволюционно формирует в P_1, а также зависит от числа допустимых особей, которых она эволюционно форирует в P_1. Уравнение (19.15) не определено, если $M_1(w) = 0$. Если $M_1(w) = 0$, то $\psi(w)$ предположительно может быть принята равной сколь угодно высокой стоимости.

Для каждой особи и поколения в P_2 эволюционный алгоритм эволюционно формирует популяцию P_1. Такой коэволюционный подход может быть вычислительно емким из-за вложенных эволюционных алгоритмов, но он поддается параллельной реализации, которая уменьшает вычислительные усилия.

На рис. 19.3 представлено схематичное описание алгоритма коэволюционных штрафов. Внешний цикл эволюционно формирует популяцию P_2. По каждому поколению P_2 (то есть для каждой итерации внешнего цикла) эволюционные алгоритмы $|P_2|$ выполняются во внутреннем цикле, эволюционно формируя кандидатные решения x с использованием оштрафованной стоимости уравнения (19.14).

Мы можем модифицировать алгоритм рис. 19.3 несколькими способами, чтобы попытаться улучшить его результативность. Например, мы могли бы применить разные типы элитарности с целью оставления лучших особей P_1 из одной эволюции P_1 в другую. Мы могли бы также выполнить более одной эволюции P_1 для каждой особи P_2, для того чтобы получить среднюю или более высокую результативность для заданного w.

Рисунок 19.3 представляет собой пример коэволюции. Здесь мы используем ее для ограниченной оптимизации, но коэволюция также имеет ряд других интересных применений. В природе мы встречаем коэволюцию повсюду. Например, цветы и пчелы эволюционировали таким образом, что они зависят друг от друга ради их взаимного выживания [Pyke, 1978]. Коэволюция также по-разному изучалась в эволюционных алгоритмах [Paredis, 2000], и она, несомненно, станет активной и плодотворной областью для будущих исследований.

$P_2 = \{w\} \leftarrow$ случайно инициализированная популяция кандидатных весов
While not (критерий останова)
  For each $w \in P_2$
    $P_1 = \{x\} \leftarrow$ случайно инициализированная популяция кандидатных решений
    Выполнить эволюционный алгоритм для минимизации уравнения (19.14) по x
    Применить уравнение (19.15) для вычисления $\psi(w)$
  Next w
  Применить стоимости $\psi(w)$ для отбора, рекомбинации и мутации P_2
Next поколение P_2

Рис. 19.3. Схематичное описание алгоритма коэволюционных штрафов для минимизации $f(x)$ при условии $G_i(x) = 0$ для $i \in [1, m + p]$

19.2.5. *Динамические штрафные методы*

Функция оштрафованной стоимости уравнения (19.8) предлагается в публикации [Joines и Houck, 1994] с $\beta = 1$ или 2 и $r_i = (ct)^\alpha$, где c и α – это константы и где t – это число поколений:

$$\begin{aligned} \phi(x) &= f(x) + (ct)^\alpha M(x) \\ M(x) &= \sum_{i=1}^{m+p} G_i(x) \end{aligned} \qquad (19.16)$$

Данный подход называется динамическим, потому что штраф на ограничениях увеличивается вместе с числом поколений. Однако, для того чтобы добиться успеха при таком подходе, стоимость $f(\cdot)$ и размер нарушения ограничений $M(\cdot)$ должны быть нормализованы так, чтобы функция оштрафованной стоимости $\phi(\cdot)$ была записана следующим образом:

$$\begin{aligned} \phi(x) &= f'(x) + (ct)^{\alpha} M'(x) \\ M'(x) &= \begin{cases} M(x) / \max_x M(x) & \text{если } \max_x M(x) > 0 \\ 0 & \text{если } \max_x M(x) = 0 \end{cases} \\ f'(x) &= f(x) / \max_x |f(x)| \end{aligned} \tag{19.17}$$

с учетом, что $f(x) > 0$ для всех x. Этим гарантируется, что компоненты оштрафованной стоимости $\phi(x)$ имеют примерно одинаковую величину. Другим вариантом является сочетание динамического штрафного метода с методом превосходства допустимых точек, описанным в разделе 19.2.2. При таком подходе оштрафованная стоимость записывается как

$$M'(x) = \begin{cases} f'(x) & \text{если } x \in \mathcal{F} \\ f'(x) + (ct)^{\alpha} M'(x) + \theta(x) & \text{если } x \notin \mathcal{F} \end{cases} \tag{19.18}$$

где $\theta(x)$ определяется таким образом, что все допустимые точки имеют более низкую оштрафованную стоимость, чем все недопустимые точки. В исследовательской литературе [Joines и Houck, 1994] часто сообщается о типичных значениях констант $c = 1/2$ и $\alpha = 1$ или 2, но подходящие значения c зависят от максимального числа поколений. Для более коротких симуляций эволюционного алгоритма (несколько сотен поколений или меньше) c должна быть больше 1/2 на один или два порядка.

Если c слишком мала, то штраф за нарушение ограничения будет слишком мал, и эволюционный алгоритм установит очень высокое значение для особей с низкими стоимостями, но большими нарушениями ограничения.

19.2.5.1. Экспоненциально-динамические штрафы

Функция экспоненциально-динамического штрафа (exponential dynamic penalty) предлагается в публикации [Carlson и Shonkwiler, 1998] как[3]

$$\phi(x) = f(x) \exp(M(x)/T), \tag{19.19}$$

где $M(x)$ – это размер нарушения ограничений, определенный в уравнении (19.16), а T – монотонно неубывающая функция от числа поколений t. $T = 1/\sqrt{t}$ предлагается в публикации [Carlson и Shonkwiler, 1998].

[3] Обратите внимание, что в публикации [Carlson и Shonkwiler, 1998] используется $\phi(x) = f(x)\exp(-M(x)/T)$, потому что там оптимизационная задача является максимизационной задачей.

В результате $\lim_{t\to\infty} T = 0$, поэтому оштрафованная стоимость недопустимых особей стремится к бесконечности, по мере того как число поколений стремится к бесконечности.

Уравнение (19.19) исходит из допущения, что $f(x) \geq 0$ для всех x: в противном случае штраф за нарушение ограничений уменьшит стоимость (то есть сделает ее более отрицательной). Если это допущение не выполняется, то мы должны сдвинуть стоимостную функцию, перед тем как ее штрафовать. Мы также можем добавить регулировочный параметр в штрафную часть $\phi(x)$:

$$\begin{aligned} \phi(x) &= f'(x)\exp(\alpha M'(x)/T) \\ f'(x) &= f(x) - \min_x f(x) \end{aligned} \tag{19.20}$$

где нормализованный размер нарушения ограничений $M'(x)$ определен в уравнении (19.17) и α – регулировочный параметр для корректировки относительного веса нарушения ограничений. Мы находим, что значения α в пределах 10 обычно работают довольно хорошо.

Как и в случае аддитивного штрафного метода, описанного в уравнении (19.17), мы можем объединить экспоненциально-динамический штрафной метод с методом превосходства допустимых точек, описанным в разделе 19.2.2. При таком подходе оштрафованная стоимость записывается как

$$\phi(x) = \begin{cases} f'(x) & \text{если } x \in \mathcal{F} \\ f'(x)\exp(M(x)/T) + \theta(x) & \text{если } x \notin \mathcal{F} \end{cases} \tag{19.21}$$

или

$$\phi(x) = \begin{cases} f'(x) & \text{если } x \in \mathcal{F} \\ f'(x)\exp(\alpha M'(x)/T) + \theta(x) & \text{если } x \notin \mathcal{F} \end{cases} \tag{19.22}$$

где $\theta(x)$ – достаточно крупная, чтобы гарантировать, чтобы все допустимые точки имели более низкую стоимость, чем все недопустимые точки.

19.2.5.2. Другие динамические штрафные подходы

Более сложные формы динамических штрафных функций предложены в публикациях [Coit и Smith, 1996], [Coit и соавт., 1996], [Joines и Houck, 1994], [Kazarlis и Petridis, 1998] и [Smith и Tate, 1993]. Динамические штрафные функции рассмотрены в публикации [Coello Coello, 2002]. Динамические штрафные методы часто работают лучше, чем

статические методы, но они требуют дополнительной регулировки. Регулировка зависит от конкретной задачи. Слишком высокие штрафы препятствуют разведыванию недопустимого множества, но иногда, для того чтобы найти хорошие решения, которые удовлетворяют ограничениям, нам нужно использовать недопустимых особей (вспомните рис. 19.2). Однако слишком низкие штрафы приводят к чрезмерному разведыванию недопустимого множества и слабой сходимости к допустимым решениям. Эти соображения подводят нас к обсуждению адаптивных штрафных методов в следующем далее разделе.

19.2.6. *Адаптивные штрафные методы*

Задачи со статическими и динамическими штрафными методами обусловливают развитие специальных типов динамических методов, которые называются адаптивными методами. Адаптивные методы используют обратную связь от популяции для корректировки штрафных весов. Один адаптивный подход предлагается в публикации [Hadj-Alouane и Bean, 1997] и устанавливает штрафные веса уравнения (19.8) следующим образом:

$$r_i(t+1) = \begin{cases} r_i(t)/\beta_1 & \text{если случай 1} \\ \beta_2 r_i(t) & \text{если случай 1} \\ r_i(t) & \text{в противном случае} \end{cases} \qquad (19.23)$$

где t – это номер поколения, β_1 и β_2 – константы, удовлетворяющие $\beta_1 > \beta_2 > 1$. *Случай* 1 означает, что лучшая особь была допустимой для каждого из прошлых k поколений, и *случай* 2 означает, что в любом из прошлых k поколений не было ни одной допустимой особи. Окно k поколения – это регулировочный параметр, влияющий на скорость адаптации. Мы видим, что если лучшая особь в популяции допустима, то мы уменьшаем вес ограничения, чтобы разрешить наличие в популяции более недопустимых особей. Если в популяции нет ни одной допустимой особи, то мы увеличиваем вес ограничения, чтобы попытаться получить несколько допустимых особей. Цель состоит в том, чтобы получить сбалансированное сочетание допустимых и недопустимых особей с целью тщательного разведывания поискового пространства и эксплуатирования информации от недопустимых особей, даже если они не удовлетворяют ограничениям. Типичными значениями констант для этого метода являются $r_i(1) = 1$, $\beta_1 = 4$, $\beta_2 = 3$ и $k = n$, где n – это размерность задачи (то есть число независимых переменных в $f(x)$) [Hadj-Alouane и Bean, 1993].

19.2.7. *Сегрегированный генетический алгоритм*

Сегрегированный генетический алгоритм (segregated GA) [Le Riche и соавт., 1995] представляет собой хитроумный подход к решению затруднений, возникающих при регулировке параметров штрафной функции. Параметры r_i в уравнении (19.8) трудно отрегулировать. Если они слишком велики, то ограниченный эволюционный алгоритм слишком много фокусируется на удовлетворении ограничений и недостаточно на минимизации стоимостной функции. Если они слишком малы, то ограниченный эволюционный алгоритм слишком много фокусируется на минимизации стоимостной функции и недостаточно на удовлетворении ограничений. Сегрегированный генетический алгоритм решает эту проблему, создавая два ранжированных списка особей: в первом списке используются малые штрафные веса r_{1i}, а во втором списке – большие штрафные веса r_{2i}. Мы отбираем особей для следующего поколения, поочередно выбирая из двух списков. Это примерно эквивалентно использованию двух субпопуляций, одна из которых имеет малый штрафной вес, а другая – большой.

Данный подход, по-видимому, раскрывает много возможностей для дополнительных исследований. Например, мы могли бы использовать более двух штрафных весов. Мы могли бы также использовать идею сегрегированного генетического алгоритма для объединения нескольких методов обработки ограничений.

19.2.8. *Самоадаптивная формулировка приспособленности*

Подход, именуемый самоадаптивной формулировкой приспособленности (self-adaptive fitness formulation) [Farmani и Wright, 2003], штрафует стоимостные значения недопустимых особей в два этапа. Сначала, если любая из недопустимых особей $\bar{x}$ имеет неоштрафованную стоимость, которая лучше, чем у лучшей допустимой особи x (то есть если $f(\bar{x}) < f(x)$ для некоторого $\bar{x} \notin \mathcal{F}$ и для лучшего $x \in \mathcal{F}$), то стоимостное значение каждой недопустимой особи штрафуется. Однако если $f(\bar{x}) > f(x)$ для всех $\bar{x} \notin \mathcal{F}$ и для лучших $x \in \mathcal{F}$, то ни одна из недопустимых особей не штрафуется. Этим обеспечивается предотвращение ненужного штрафования недопустимых особей. Это позволяет недопустимым особям иметь достаточно низкие значения оштрафованной стоимости, чтобы они могли оставаться в популяции и чтобы их информация могла эксплуатироваться.

Далее мы реализуем второй этап наложения штрафа. Все недопустимые особи штрафуются таким образом, что особь с самым большим нарушением ограничений имеет худшую оштрафованную стоимость. Мы делаем это, сначала определяя общую недопустимость для каждой особи x следующим образом:

$$\iota(x) = 1/m + p \sum_{i=1}^{m+p} G_i(x)/\max_{\bar{x} \notin \mathcal{F}} G_i(\bar{x}), \tag{19.24}$$

где $G_i(\cdot)$ определяется в уравнении (19.8) с $\beta = 1$. Далее мы определим особей, которые являются лучшими (x_b), худшими с точки зрения допустимости (x_{wf}) и худшими с точки зрения стоимости (x_{wc}), следующим образом:

$$x_b = \begin{cases} \arg\min_x f(x) : x \in \mathcal{F} & \text{если } \mathcal{F} \neq 0 \\ \arg\min_x \iota(x) & \text{в противном случае} \end{cases}$$

$$x_{wf} = \begin{cases} \arg\max_x \iota(x) : f(x) < f(x_b) & \text{если } \exists \bar{x} \notin \mathcal{F} \text{ такой что } f(\bar{x}) < f(x_b) \\ \arg\max_x \iota(x) & \text{в противном случае} \end{cases}$$

$$x_{wf} = \arg\max_x f(x) \tag{19.25}$$

Обратите внимание, что если $\mathcal{F} \neq 0$, то $x_b \in \mathcal{F}$, даже если существуют некоторые $\bar{x} \notin \mathcal{F}$ с более низкой стоимостью. С помощью этих определений метрический показатель неосуществимости нормализуется в диапазоне [0, 1] следующим образом:

$$\tilde{\iota}(x) = \frac{\iota(x) - \iota(x_b)}{\iota(x_{wf}) - \iota(x_b)}. \tag{19.26}$$

Теперь мы можем математически определить первый штрафной этап как

$$\phi(x) = \begin{cases} f(x) + \tilde{\iota}(x)(f(x_b) - f(x) & \text{если } \exists \bar{x} \notin \mathcal{F} \text{ такой, что } f(\bar{x}) < f(x_b) \\ f(x) & \text{в противном случае} \end{cases} \tag{19.27}$$

Второй штраф отображает $\phi(x)$ в дополнительно оштрафованную стоимость $\phi'(x)$:

$$\phi'(x) = \phi(x) + \gamma|\phi(x)|\left(\frac{\exp(2\tilde{\iota}(x)) - 1}{\exp(2) - 1}\right)$$

$$\gamma = \begin{cases} (f(x_{wc}) - f(x_b)) / f(x_b) & \text{если } f(x_{wf}) < f(x_b) \\ 0 & \text{если } f(x_{wf}) = f(x_b) \\ (f(x_{wc}) - f(x_{wf})) / f(x_{wf}) & \text{если } f(x_{wf}) > f(x_b) \end{cases} \tag{19.28}$$

Экспоненциальная функция в уравнении (19.28) приводит совсем к небольшому штрафу для особей с небольшими нарушениями ограничений. Коэффициент шкалирования 7 гарантирует, что особь с наибольшим нарушением ограничений имеет наибольшую оштрафованную стоимость:

$$\phi'(x_{wf}) \geq \phi'(x) \quad \text{для всех } x. \tag{19.29}$$

Данный двухэтапный штрафной метод до некоторой степени запутан, но основная его цель состоит в том, чтобы после наложения штрафа лучшие особи в популяции включали в себя немного тех, которые допустимы, и немного тех, которые недопустимы. Особи с низкой стоимостью и небольшими нарушениями ограничений конкурентоспособны с допустимыми особями с точки зрения оштрафованной стоимости. Эта идея выглядит довольно эффективной, и она была применена во многих ограниченных оптимизационных задачах. Дополнительная работа могла бы быть сосредоточена на более четкой регулировке и упрощении подхода на основе штрафа при одновременном достижении его основополагающих целей.

19.2.9. *Самоадаптивная штрафная функция*

Алгоритм с самоадаптивной штрафной функцией (self-adaptive penalty function, SAPF) [Tessema и Yen, 2006] адаптирует штрафные функции на основе распределения популяции. Если имеется только несколько допустимых особей, то мы хотим назначить низкие оштрафованные стоимости $\phi(\cdot)$ особям с небольшими нарушениями ограничений, даже если они могут иметь высокие стоимости $f(\cdot)$. С другой стороны, если имеется много допустимых особей, то мы хотим назначить низкие оштрафованные стоимости $\phi(\cdot)$ только тем особям, которые имеют низкие стоимости $f(\cdot)$. Алгоритм с самоадаптивной штрафной функцией состоит из следующих шагов.

1. Нормализовать значение стоимостной функции для каждой особи x:

$$f_1(x) = \frac{f(x) - \min_x f(x)}{\max_x f(x) - \min_x f(x)}. \tag{19.30}$$

   В результате получится нормализованная стоимость $f_1(x) \in [0, 1]$ для всех x, где лучшая особь с точки зрения стоимости имеет нормализованную стоимость, равную 0, и худшая особь с точ-

ки зрения стоимости имеет нормализованную стоимость, равную 1.

2. Вычислить нормализованный размер нарушения ограничений $\iota(x)$ каждой особи, как показано в уравнении (19.24). В результате получится $\iota(x) \in [0, 1]$ для всех x. Обратите внимание, что $\iota(x) = 0$ для всех $x \in \mathcal{F}$ и $\iota(\bar{x}) > 0$ для всех $\bar{x} \notin \mathcal{F}$. Кроме того, может существовать или не существовать $\bar{x}$, такой что $\iota(\bar{x}) = 1$.
3. Вычислить значение расстояния для каждой особи x:

$$d(x) = \begin{cases} \iota(x) & \text{если } \mathcal{F} = \emptyset \\ \sqrt{f_1^2(x) + \iota^2(x)} & \text{если } \mathcal{F} \neq \emptyset \end{cases} \tag{19.31}$$

Если в популяции нет допустимых особей, то значение расстояния x равно общему нарушению ограничений x, без учета стоимости x. Среди двух допустимых особей та, которая имеет более низкую стоимость, будет иметь меньшее расстояние. Если в популяции есть допустимые особи, то расстояние недопустимой особи представляет собой комбинацию ее стоимости и нарушения ограничений. Следовательно, когда мы сравниваем допустимую особь x с недопустимой особью $\bar{x}$, любая из них может иметь меньшее расстояние, в зависимости от их относительных стоимостных значений.

4. Вычислить две дополнительные функции оштрафованной стоимости:

$$\begin{aligned} X(x) &= \begin{cases} 0 & \text{если } \mathcal{F} = \emptyset \\ \iota(x) & \text{если } \mathcal{F} \neq \emptyset \end{cases} \\ Y(x) &= \begin{cases} 0 & \text{если } x \in \mathcal{F} \\ f_1(x) & \text{если } x \notin \mathcal{F} \end{cases} \end{aligned} \tag{19.32}$$

Оштрафованная стоимость $X(x)$ равна 0, если в популяции нет допустимых особей, и равна $\iota(x)$, если в популяции есть допустимые особи. Данная оштрафованная стоимость служит для штрафования недопустимых особей на основе размера их нарушений ограничений, но только в том случае, если популяция содержит допустимые особи. Оштрафованная стоимость $Y(x)$ равна 0, если x является допустимой, и равна нормализованной стоимости $f_1(x)$, если x является недопустимой. Данная оштрафованная стоимость

служит для дальнейшего штрафования недопустимых особей на величину, пропорциональную их стоимости.

5. Вычисление функции оштрафованной стоимости:

$$\phi(x) = d(x) + (1 - r)X(x) + rY(x), \tag{19.33}$$

где $r \in [0, 1]$ – это доля допустимых особей в популяции. Если в популяции много допустимых особей, то $\phi(x)$ подчеркивает $Y(x)$, что включает стоимостно-ориентированные штрафы, накладываемые на недопустимых особей. С другой стороны, если в популяции мало допустимых особей, то $\phi(x)$ подчеркивает $X(x)$, что включает штрафы за нарушение ограничений, накладываемые на недопустимых особей.

Алгоритм с самоадаптивной штрафной функцией (SAPF) также был адаптирован для ограниченных многокритериальных оптимизационных задач [Yen, 2009].

19.2.10. *Адаптивная сегрегационная обработка ограничений*

Эволюционный алгоритм с адаптивной сегрегационной обработкой ограничений (adaptive segregational constraint handling evolutionary algorithm, ASCHEA) предлагается в публикациях [Hamida и Schoenauer, 2000], [Hamida и Schoenauer, 2002]. Алгоритм ASCHEA опирается на две идеи. Во-первых, мы стараемся поддерживать определенное соотношение допустимых и недопустимых особей. Такой подход похож на адаптивный подход, который мы обсуждали в разделе 19.2.6. Он позволяет нам разведывать все поисковое пространство, включая допустимую и недопустимую части.

Во-вторых, если в популяции мало допустимых особей, то мы позволяем допустимым особям рекомбинироваться только с допустимыми особями. Такое правило опирается на идею, что решения ограниченных оптимизационных задач часто лежат на границе ограничения либо вблизи нее [Leguizamón и Coello Coello, 2009], [Ray и соавт., 2009b]. Следовательно, в случае решения ограниченных оптимизационных задач имеет смысл подталкивать как допустимых, так и недопустимых особей к границе ограничения. Рекомбинация допустимых особей с недопустимыми особями, как правило, приближает их детей к границе ограничения.

Алгоритм ASCHEA использует штрафной подход уравнения (19.8) с $\beta = 1$ и следующий метод обновления штрафного веса:

$$\phi(x) = f(x) + \sum_{i=1}^{m+p} r_i G_i(x)$$

$$r_i(t+1) = \begin{cases} r_i(t)/\gamma & \text{если } \tau(t) > \tau_d \\ \gamma r_i(t) & \text{в противном случае} \end{cases} \tag{19.34}$$

где t – это номер поколения, $\gamma > 1$ – регулировочный параметр, τ_d – целевое отношение числа допустимых особей к недопустимым особям и $\tau(t)$ – отношение в поколении t:

$$f_1(x) = \frac{|\{x : x \in \mathcal{F}\}|}{|\{\bar{x} : \bar{x} \notin \mathcal{F}\}|} \quad \text{в поколение } t. \tag{19.35}$$

Алгоритм ASCHEA часто реализуется с целевым значением $\tau_d = 1/2$.

Вторая идея, реализованная в алгоритме ASCHEA, заключается в том, чтобы позволить допустимым особям рекомбинировать только с недопустимыми особями, если $\tau(t) < \tau_d$. Это побуждает детей недопустимых особей двигаться к допустимому участку, который, возможно, увеличивает число допустимых особей.

Вместе с тем если $\tau(t) \geq \tau_d$, то у нас в популяции уже достаточно допустимых особей, поэтому мы проводим отбор со следующими двумя шагами: во-первых, мы отбираем указанное число особей из допустимого множества $\mathcal{F}$; во-вторых, мы отбираем оставшихся особей для рекомбинации на основе значений оштрафованной стоимости $\phi(\cdot)$ без какого-либо явного рассмотрения допустимости. Минимальное число допустимых особей, которых мы отбираем для рекомбинации, обычно составляет около 30 % от общего числа особей, которых нам нужно отобрать. Например, если для рекомбинации мы хотим отобрать 100 особей, мы сначала отбираем 30 допустимых особей на основе стоимости, а затем оставшихся 70 особей из всей популяции на основе оштрафованной стоимости.

Аналогичный алгоритм, именуемый эволюционным алгоритмом под управлением недопустимости (infeasibility driven evolutionary algorithm, IDEA), предлагается в публикации [Ray и соавт., 2009b] как для однокритериальной, так и для многокритериальной оптимизации.

19.2.11. *Поведенческая память*

В поведенческой памяти (behavioral memory) для решения ограниченных оптимизационных задач используется метод «разделяй и властвуй» [Michalewicz и соавт., 1996], [Schoenauer и Xanthakis, 1993].

С учетом задачи уравнения (19.8) с ограничениями $m + p$ мы сначала эволюционно формируем популяцию, которая минимизирует $G_1(x)$, то есть мы минимизируем нарушение первого ограничения, не рассматривая никаких других ограничений либо стоимостной функции. Мы завершаем этот эволюционный алгоритм, после того как задаваемая пользователем доля популяции удовлетворяет первому ограничению. После завершения этой эволюции мы используем ее окончательную популяцию для инициализации эволюционного алгоритма, который минимизирует $G_2(x)$. Во время работы этого второго эволюционного алгоритма мы удаляем из популяции особей, нарушающих ограничение $G_1(x)$, но не учитываем другие ограничения или стоимостную функцию.

Мы повторяем эти шаги для каждого ограничения. Мы инициализируем i-й эволюционный алгоритм результатами $(i - 1)$-го эволюционного алгоритма. i-й эволюционный алгоритм эволюционно формирует популяцию, которая минимизирует $G_i(x)$, и в ходе этой эволюции мы удаляем любых особей, которые выходили за пределы любого из $G_j(x)$ ограничений для $j \in [1, i - 1]$. Наконец, после того как все $m + p$ ограничений были удовлетворены последующей реализацией $m + p$ эволюционных алгоритмов, мы используем результирующую популяцию для инициализации эволюционного алгоритма, который минимизирует стоимостную функцию, применяемую для всех $m + p$ ограничений. На рис. 19.4 схематично представлен алгоритм поведенческой памяти.

Поведенческая память может быть классифицирована как подход на основе штрафной функции, поскольку в ней используется смертная казнь. Однако строка на рис. 19.4, начинающаяся с инструкции «Выполнить эволюционный алгоритм для минимизации...», может включать, а может и не включать, подходы, предусматривающие применение штрафных функций. Данный эволюционный алгоритм может быть любым эволюционным алгоритмом и включать любой подход к обработке ограничений, если он в конечном итоге удаляет особей, которые нарушают ранее рассмотренные ограничения.

$\{x\}_0 \leftarrow$ популяция, инициализированная случайно
For $i = 1, ..., m + p$
  Инициализировать i-й эволюционный алгоритм: $\{x\} \leftarrow \{x\}_{i-1}$
  Выполнить эволюционный алгоритм для минимизации $G_i(x)$, при условии что $G_j(x) = 0$ для $j < i$, используя смертную казнь, которая обеспечивает, что все особи удовлетворяют G_j ограничениям, и обозначить окончательную популяцию данного эволюционного алгоритма как $\{x\}_i$
Next i

Инициализировать окончательный эволюционный алгоритм: $\{x\} \leftarrow \{x\}_{m+p}$
Выполнить эволюционный алгоритм для минимизации $f(x)$, при условии что $G_j(x) = 0$ для $j \in [1, m + p]$, используя смертную казнь, которая обеспечивает, что все особи удовлетворяют G_j ограничениям.

Рис. 19.4. Схематичное описание алгоритма поведенческой памяти для минимизации $f(x)$ при условии $G_i(x) = 0$ для $i \in [1, m + p]$

Поведенческая память, по сути дела, является обобщением неограниченных оптимизационных алгоритмов, которые постепенно увеличивают число оцениваний стоимостной функции [de Garis, 1990], [Gathercole и Ross, 1994]. Например, предположим, что мы хотим минимизировать $f_i(x)$ по x, где i может быть любым из большого множества значений $\mathcal{L} = \{1, 2, \ldots, i_{max}\}$. Одним из способов решения этой задачи является минимизация $f_1(x)$ с помощью эволюционного алгоритма. Затем, используя это окончательную популяцию, мы минимизируем $f_1(x) + f_2(x)$. Опять же, используя эту вторую окончательную популяцию, мы минимизируем $f_1(x) + f_2(x) + f_3(x)$. Мы продолжаем этот процесс до тех пор, пока не убедимся, что мы минимизировали $f_i(x)$ усредненно по всем $i \in \mathcal{L}$. Еще один подход состоит в том, чтобы минимизировать комбинацию случайной выборки образцов $f_i(x)$, постепенно увеличивая число образцов по мере увеличения числа поколений. Такой подход называется стохастической выборкой [Banzhaf и соавт., 1998, раздел 10.1.5]. Мы часто встречаем этот подход в генетическом программировании, так как оценивание стоимости одной компьютерной программы требует компьютерных прогонов большого числа входных случаев (см. главу 7).

Минимизация $f_i(x)$ для $i \in \mathcal{L}$, по-видимому, имеет ту же форму, что и многокритериальная оптимизационная задача (см. главу 20), но обсуждаемая здесь задача на самом деле является однокритериальной оптимизационной задачей; каждая стоимостная функция $f_i(x)$ является той же самой функцией, но она вычисляется с разными параметрами для разных значений i. Вместе с тем разделительная линия между однокритериальными оптимизационными задачами с несколькими параметрами и многокритериальными оптимизационными задачами нечеткая. Одни, возможно, будут рассматривать или обрабатывать задачу как однокритериальную, другие – как многокритериальную.

19.2.12. *Стохастическое ранжирование*

Стохастическое ранжирование (stochastic ranking) [Runarsson и Yao, 2000] добавляет в ограниченные эволюционные алгоритмы стохасти-

ческую составляющую. Поскольку случайность в эволюционных алгоритмах является особо важным компонентом, имеет смысл включить случайность в эволюционно-алгоритмический подход с использованием обработки ограничений. Стохастическое ранжирование иногда ранжирует кандидатные решения по их стоимости $f(\cdot)$, а иногда ранжирует их в соответствии с размером нарушений ограничений. Выбор того, как ранжировать особей, является стохастическим. Когда мы сравниваем двух особей x_1 и x_2, то считаем особь x_1 лучше x_2, если:

- оба решения являются допустимыми и $f(x_1) < f(x_2)$, либо
- случайно сгенерированное число $r \sim U[0, 1]$ меньше определяемой пользователем вероятности P_f и $f(x_1) < f(x_2)$, либо
- ни одно из вышеперечисленных условий не выполняется, и x_1 имеет меньшее нарушение ограничения, чем x_2.

В противном случае мы считаем x_2 лучше x_1. Мы видим, что вполне можем сравнить x_1 и x_2 на основе их стоимости, либо мы вполне можем сравнить их на основе их нарушений ограничений, в зависимости от исхода генератора случайных чисел. После того как мы сравнили и отсортировали всех особей в популяции, мы проводим отбор и рекомбинацию для следующего поколения. Значения вероятности $P_f \in (0.4, 0.5)$ дают хорошие результаты для многих эталонных задач [Runarsson и Yao, 2000].

19.2.13. *Нишевой штрафной метод*

Нишевой штрафной метод (niched-penalty approach) [Deb и Agrawal, 1999], [Deb, 2000] обусловлен трудностью регулировки параметров штрафных методов. В нем для отбора особей с целью рекомбинации применяется турнирный отбор в соответствии со следующими правилами.

- При наличии двух допустимых особей турнир выигрывает та, у которой более низкая стоимость.
- При наличии одной допустимой и одной недопустимой особей турнир выигрывает допустимая особь.
- При наличии двух недопустимых особей турнир выигрывает та, у которой меньшее нарушение ограничений.

Этот метод привлекателен своей простотой; он не требует регулировки штрафных параметров. Сравнение двух недопустимых особей не требует каких-либо оцениваний стоимостной функции, что в итоге

может уменьшить вычислительные усилия. Нишевой штрафной метод часто получает хорошие результаты в задачах ограниченной оптимизации. Вместе с тем его простота также может быть недостатком, потому что он считает допустимую особь с очень высокой стоимостью лучше недопустимой особи с очень низкой стоимостью. Поэтому он может плохо работать в задачах, решения которых находятся на границе ограничений, что имеет место в случае многих реальных оптимизационных задач [Leguizamón и Coello Coello, 2009], [Ray и соавт., 2009b].

«Нишевая» часть этого подхода не является неотъемлемой частью его способности обрабатывать ограничения, но она предназначена для сохранения многообразия в популяции и описывается следующим образом. Мы не разрешаем особям участвовать в турнире друг с другом (для отбора), если они находятся далеко друг от друга в поисковом пространстве. После того как мы случайно выбираем особей для турнирного отбора, мы вычисляем их расстояние друг от друга. Если участники друг от друга слишком далеки, то мы случайно выбираем других участников турнира. Такое решение не дает исчезнуть удаленным кластерам особей из популяции и, таким образом, поддерживает многообразие.

Многочленная эволюционная стратегия (multimembered evolution strategy, MMES) [Mezura-Montes и Coello Coello, 2005] похожа на нишевой штрафной метод. Отличительной особенностью алгоритма MMES является то, что время от времени (около 3 % поколений) недопустимые особи с наименьшими значениями стоимостной функции (либо недопустимые особи с наименьшими нарушениями ограничений) гарантированно отбираются для следующего поколения. Такой элитарный подход гарантирует, что хорошие недопустимые особи будут вносить свой вклад в процесс эволюционно-алгоритмического поиска.

19.3. Специальные представления и специальные операторы

Специальное представление – это способ формулирования задачи таким образом, чтобы ограничения автоматически удовлетворялись кандидатными решениями. Специальный оператор – это способ определения эволюционного-алгоритмических операторов рекомбинации и мутации таким образом, чтобы детские особи автоматически удовлетворяли ограничениям. Оба этих подхода в значительной степени зависят от конкретных задач. Мы не можем написать исходный код для специального представления или исходный код для специального

оператора, который будет применяться к широкому классу задач. Тем не менее, несмотря на то что зависимый от конкретной задачи характер специальных представлений и специальных операторов создает больше работы для контруктора эволюционного алгоритма, эта работа часто приносит высокие дивиденды. Причиной тому является то, что результирующий эволюционный алгоритм использует специфичную для конкретной задачи информацию, которая часто приводит к более высокой результативности, чем мы можем рассчитывать от более универсального эволюционного алгоритма. Эта концепция имеет непосредственное отношение к теореме об отсутствии бесплатных обедов (см. приложение B.1).

В разделе 19.3.1 обсуждаются специальные представления, включая метод, именуемый подходом на основе декодера. В разделе 19.3.2 обсуждаются специальные операторы, в том числе популярный ограниченный эволюционный алгоритм Genocop.

19.3.1. *Специальные представления*

В качестве простого примера подхода к ограниченной оптимизации на основе специального представления рассмотрим двумерную задачу

$$\min_x f(x) \quad \text{такая, что} \quad x_1^2 + x_2^2 \le K, \tag{19.36}$$

где k – это некая константа. Она может быть трансформирована в эквивалентную неограниченную задачу

$$\min_{\rho,\theta} f(x) \quad \text{такая, что} \quad \rho \in [0, K] \ \text{ и } \ \theta \in [0, 2\pi],$$

$$\text{где} \quad x_1 = \rho \cos\theta \ \text{ и } \ x_2 = \rho \sin\theta. \tag{19.37}$$

Эта простая трансформация из прямоугольной в полярную преобразовывает ограниченную задачу уравнения (19.36) в неограниченную задачу уравнения (19.37). Исходная задача уравнения (19.36) имеет нелинейное ограничение, но мы трансформировали ее в задачу уравнения (19.37), единственными ограничениями которого являются простые ограничения на поисковую область.

Декодеры

Один из подходов к решению ограниченных оптимизационных задач заключается в кодировании команд, определяющих кандидатные решения, гарантируя, что набор команд всегда приводит к допустимой

особи. То есть вместо прямого кодирования кандидатного решения задачи мы кодируем набор команд, который используем для получения кандидатного решения. Затем каждая особь в эволюционно-алгоритмической популяции состоит из набора команд для построения кандидатного решения. Мы также можем применять этот подход для неограниченных оптимизационных задач, но он, по-видимому, особо применим к ограниченным оптимизационным задачам, потому что в зависимости от конкретной задачи мы можем определить набор команд, который всегда удовлетворяет ограничениям задачи.

Набор команд для построения кандидатных решений называется декодером [Palmer и Kershenbaum, 1994], [Koziel и Michalewicz, 1998], [Koziel и Michalewicz, 1999] и должен удовлетворять ряду свойств.

- Для каждого допустимого решения оптимизационной задачи должен существовать хотя бы один декодер.
- Каждое допустимое решение должно соответствовать одинаковому числу декодеров, чтобы не вводить смещение в поиск.
- Каждый декодер должен соответствовать допустимому решению.
- Трансформация между декодером и кандидатным решением должна быть вычислительно быстрой, по сравнению с оцениванием стоимости.
- Малые изменения в декодере должны соответствовать малым изменениям в кандидатном решении.

Эти правила могут быть смягчены в конкретных приложениях, если они слишком сложны для их удовлетворения [Koziel и Michalewicz, 1999], но они, по крайней мере, предоставляют полезные рекомендации. Например, у нас может быть задача, ограничения которой очень трудно удовлетворить, но мы можем найти набор декодеров, которые дают кандидатные решения, допустимый высокий процент времени. Несмотря на то что такой набор декодеров не полностью удовлетворяет вышеуказанным условиям, он может быть предпочтительнее кодирования потенциальных решений непосредственно в нашем эволюционном алгоритме.

После того как мы получим набор команд для построения кандидатных решений, мы переопределяем эволюционно-алгоритмическую популяцию, чтобы она состояла из набора команд (декодеров). Затем мы выполняем отбор, рекомбинацию и мутацию на декодерах, гарантируя тем самым удовлетворение ограничений.

■ Пример 19.2

Рассмотрим выпуклый допустимый участок $\mathcal{F}$ задачи двумерной ограниченной оптимизации, представленной на рис. 19.5. Опорная точка r – это произвольная точка в $\mathcal{F}$. Любая точка $x \in \mathcal{F}$ однозначно представлена формулой $\alpha b + (1 - \alpha)r$ для некоторого $\alpha \in [0, 1]$ и некоторого b на границе поисковой области[4].

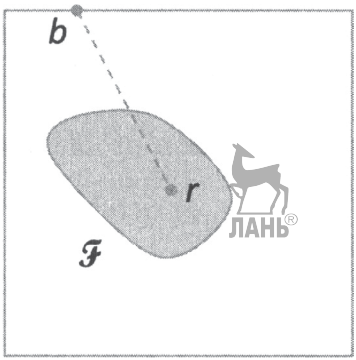


Рис. 19.5. Пример алгоритма декодера для оптимизации на $\mathcal{F}$. При наличии любой точки $r \in \mathcal{F}$ любая точка $x \in \mathcal{F}$ однозначно представлена формулой $\alpha b + (1 - \alpha)r$ для некоторого $\alpha \in [0, 1]$ и некоторого b на границе поисковой области

Декодер для решения этой оптимизационной задачи вполне может действовать следующим образом.

1. Мы каким-то образом находим допустимую точку r.
2. Мы находим границу $\mathcal{F}$, перемещая b по всей границе поисковой области. Для каждого b мы находим максимум α – такой, что $\alpha b + (1 - \alpha)r \in \mathcal{F}$. Этот максимум обозначается как $\alpha_{\max}(b) = \max\{\alpha : \alpha b + (1 - \alpha)r \in \mathcal{F}\}$.
3. Мы определяем эволюционно-алгоритмическую популяцию особей так, чтобы каждая особь состояла из пары (b, α). Параметром b может быть любая точка на границе поисковой области, а параметром α – любое вещественное число в $[0, \alpha_{\max}(b)]$.
4. Мы выполняем эволюционный алгоритм, производя рекомбинацию на парах (b, α). Единственная проверка на допустимость, о которой нам нужно беспокоиться, – это $\alpha \in [0, \alpha_{\max}(b)]$.

■

[4] Есть одно исключение: r имеет бесконечное число таких представлений, так как $r = (0)(b) + (1)(r)$ для всех b.

Пример 19.2 необходимо будет модицифировать для невыпуклых допустимых участков, но он иллюстрирует то, как декодер может использоваться для упрощения ограничений.

19.3.2. *Специальные операторы*

Во многих реальных оптимизационных задачах решение лежит на границе ограничений [Leguizamón и Coello Coello, 2009], [Ray и соавт., 2009b]. Рассмотрим оптимизационную задачу покупки некого оборудования. Если бы нам было поручено закупить лучшее оборудование, то это поручение, возможно, было бы ограничено верхней границей цены. Мы, вероятно, потратили бы максимально допустимую сумму денег, потому что лучшее оборудование обычно стоит больше, чем худшее. То есть ограниченный оптимум будет лежать на границе ценового ограничения. Ограничения, связанные с цветом оборудования, как правило, *не являются* частью задания на закупку оборудования, поскольку оптимальность и цвет оборудования обычно не связаны.

Точно так же, если бы мы хотели проехать из точки A в точку B за минимальное время с верхним пределом использования топлива, то мы, вероятно, использовали бы максимально допустимое количество топлива, потому что обычно едем быстрее, используя больше топлива. То есть ограниченный оптимум будет лежать на границе ограничения топлива. Когда мы думаем о реальных задачах, то понимаем, что большинство решений оптимизационных задач лежит на границе ограничений[5].

Это приводит к идее решения оптимизационной задачи путем разведывания границы ограничения. Мы вполне можем спокойно проигнорировать внутреннюю часть границы ограничения, так как ожидаем, что решения оптимизационной задачи лежат на границе ограничения. Например, рассмотрим оптимизационную задачу с ограничением произведения

$$x_1 x_2 x_3 x_4 \geq 0.75. \tag{19.38}$$

Если у нас есть знание предметной области, которое побуждает нас ожидать, что ограниченный оптимум лежит на границе ограничения, то мы можем попытаться решить оптимизационную задачу путем по-

[5] Это утверждение относится к реальным задачам, но не обязательно к эталонным задачам, которые мы встречаем в литературе. Эта разница между эталонными задачами и реальными практическими задачами связана с обсуждаемой в приложении В теоремой об отсутствии бесплатных обедов.

иска комбинаций (x_1, x_2, x_3, x_4), которые удовлетворяют следующему [Michalewicz и Schoenauer, 1996]:

$$x_1 x_2 x_3 x_4 = 0.75. \tag{19.39}$$

Мы можем инициализировать особей в эволюционно-алгоритмической популяции следующим образом:

$$\begin{aligned} x_1 &= U[-x_{\max}, x_{\max}] \\ x_2 &= U[-x_{\max}, x_{\max}]] \\ x_3 &= U[-x_{\max}, x_{\max}] \\ x_4 &= 0.75/(x_1 x_2 x_3) \end{aligned} \tag{19.40}$$

Этим гарантируется, что (x_1, x_2, x_3, x_4) удовлетворяет уравнению (19.38). Теперь предположим, что у нас есть еще одна особь (y_1, y_2, y_3, y_4) – такая, что $y_1 y_2 y_3 y_4 = 0.75$. Тогда если мы создадим детскую особь z – такую, что $z_i = x_i^{\alpha} y_i^{1-\alpha}$ для $i \in [1, 4]$, где действительное число $\alpha \in [0, 1]$, то ребенок всегда будет удовлетворять ограничению $z_1 z_2 z_3 z_4 = 0.75$. Это верно, потому что

$$\begin{aligned} z_1 z_2 z_3 z_4 &= x_1^{\alpha} y_1^{1-\alpha} x_2^{\alpha} y_2^{1-\alpha} x_3^{\alpha} y_3^{1-\alpha} x_4^{\alpha} y_4^{1-\alpha} \\ &= (x_1 x_2 x_3 x_4)\alpha\, (y_1 y_2 y_3 y_4) 1-\alpha \\ &= (0.75)^{\alpha}\, (0.75)^{1-\alpha} = 0.75 \end{aligned} \tag{19.41}$$

Мы видим, что наш специализированный оператор скрещивания гарантирует, что ребенок двух допустимых родителей всегда будет допустимым.

Мы также можем разработать для этой задачи специализированный оператор мутации:

$$\begin{aligned} x'_i &\leftarrow qx_i \\ x'_j &\leftarrow x_j/q \end{aligned} \tag{19.42}$$

где $i \in [1, 4]$ и $j \in [1, 4]$ – это разлные случайные целые числа и q – случайное число. Любая допустимая особь x, которая мутируется таким образом, приведет к допустимой особи x'.

Эти простые операторы рекомбинации и мутации иллюстрируют использование специализированных операторов для ограниченных эволюционных алгоритмов. Обратите внимание, что специализированные операторы специфичны для конкретной задачи; проектировщик эволюционного алгоритма должен сформулировать свои собственные специфичные для конкретной задачи операторы, которые будут обеспечивать допустимость, и это касается любых ограниченных оптимиза-

ционных задач. Дополнительный пример специализированного оператора приведен в публикации [Michalewicz и Schoenauer, 1996].

19.3.3. *Алгоритм Genocop*

Далее мы обсудим алгоритм, известный как генетический алгоритм для численной оптимизации ограниченных задач (genetic algorithm for numerical optimization of constrained problems, Genocop) [Michalewicz и Janikow, 1991].

Данный алгоритм включает в себя несколько специальных функциональных особенностей и методов для ограниченной оптимизации. Мы включили алгоритм Genocop в данный раздел, потому что он был изначально предложен вместе с идеей реализации специальных операторов на эволюционно-алгоритмических особях, обеспечивающих удовлетворение ими линейных ограничений.

Идея алгоритма Genocop заключается в том, что иногда, в зависимости от формы ограничения, мы можем использовать специфичные для конкретной задачи операторы, для того чтобы превратить недопустимых особей в допустимых. Мы можем сделать это с помощью ограничения в виде уравнения (19.38). Если у нас есть особь, которая не удовлетворяет ограничению, то мы можем заменить ее четвертую составляющую, как показано в последней строке уравнения (19.40). Это будет исправительный подход. В качестве альтернативы, если создадим первые три элемента детской особи, мы можем затем создать четвертый элемент, как показано в последней строке уравнения (19.40). Это будет подход на основе специального оператора.

Предположим, что у нас есть линейное ограничение

$$-8x_1 + x_3 \leq 0. \tag{19.43}$$

Если у нас есть особь, которая не удовлетворяет этому ограничению, то мы можем легко исправить ее, установив x_3 равным любому числу меньше или равным $8x_1$. Модифицированная особь будет удовлетворять ограничению. Этот пример показывает, что любое линейное ограничение может быть легко выполнено с помощью алгоритмов исправления или специальных операторов.

Алгоритм Genocop является эффективным, но его конструктивное воплощение зависит от конкретной задачи. Как показано выше, алгоритм Genocop лимитирован линейными ограничениями и специальными формами нелинейных ограничений, в которых одна переменная может быть решена в терминах других. Это является недостатком с точ-

ки зрения усилий пользователя, но преимуществом с точки зрения эффективности эволюционного алгоритма.

19.3.4. *Алгоритм Genocop II*

Алгоритм Genocop II [Michalewicz и Attia, 1994] совмещает алгоритм Genocop, как описано выше, с динамическим штрафом, аналогичным тому, который рассматривался в разделе 19.2.5. В алгоритме Genocop II используются специализированные операторы, максимизирующие допустимость эволюционно-алгоритмической популяции. Во-первых, мы удовлетворяем все линейные ограничения путем исправления недопустимых особей, как это предлагается в алгоритме Genocop. Во-вторых, обрабатываем нелинейные ограничения, минимизируя $\phi(x)$в уравнении (19.8), где все ограничения в этом уравнении нелинейны, поскольку мы уже удовлетворили линейные ограничения с помощью специальных операторов. Вес r_i в уравнении (19.8) равен $1/\tau$. Алгоритм Genocop II поддерживает постоянное значение τ в течение нескольких поколений. Через некоторое время (например, после определенного числа поколений или после того, как определенная часть популяции стала допустимой) алгоритм Genocop II уменьшает τ. Этим увеличивается давление ограничения, что приводит к постепенному привлечению все большего числа особей к допустимому множеству. Ранние публикации по алгоритму Genocop II предполагают уменьшение τ в 10 раз при каждом его уменьшении [Michalewicz и Attia, 1994], [Michalewicz и Schoenauer, 1996].

19.3.5. *Алгоритм Genocop III*

Алгоритм Genocop III [Michalewicz и Nazhiyath, 1995] является дальнейшей модификацией алгоритма Genocop. В этом методе коэволюционный алгоритм поддерживает популяцию $P_r = \{x_r\}$ опорных точек, удовлетворяющих всем ограничениям, и популяцию $P_s = \{x_s\}$ поисковых точек, удовлетворяющую линейным ограничениям из-за подхода, используемого в алгоритме Genocop. P_r и P_s могут быть разных размеров. Мы назначаем значение стоимостной функции каждой особи x_s из P_s, используя ее исправленную версию. То есть мы используем информацию из P_r для исправления x_s и для получения особи x'_s, удовлетворяющей всем ограничениям. Затем мы назначаем $f(x_s) \leftarrow f(x'_s)$. Мы создаем x'_s, генерируя последовательность точек $x = \alpha x_s + (1 - \alpha)x_r$ для множества случайных чисел $\alpha \in [0, 1]$ и для случайно отобранной $x_r \in P_r$. То есть мы выполняем поиск в случайном множестве точек, которые находятся на

прямой линии, соединяющей x_s и x_r. После того как этот поиск дает нам допустимую x, мы назначаем $x'_s \leftarrow x$ и $f(x_s) \leftarrow f(x'_s)$. Также если $f(x'_s) < f(x_r)$, мы заменяем x_r на x'_s в P_r. Наконец, мы также заменяем x_s на x'_s в P_s с некой определяемой пользователем вероятностью замены ρ.

Вопрос о том, заменять или не заменять особь x_s ее исправленной версией x'_s, связан с ламарковым наследованием: то есть может ли организм передавать своим детям черты, которые он приобретает в течение своей жизни? Одни исследователи никогда не заменяют особей на их исправленные версии ($\rho = 0$), другие всегда заменяют особей на их исправленные версии ($\rho = 1$), а иные отмечают, что значения ρ между 5 % и 20 % дают хорошие результаты [Michalewicz и Schoenauer, 1996], [Orvosh и Davis, 1993].

На рис. 19.6 приводится схематичное описание алгоритма Genocop III. Первый шаг в алгоритме Genocop III заключается в случайной инициализации P_s. Мы создаем эту популяцию без всякой оглядки на допустимость, кроме удовлетворения линейных ограничений, как предложено в алгоритме Genocop. Вторым шагом является вычисление $f(x_s)$ для каждой $x_s \in P_s$. Мы выполняем это с помощью функции «Оценить $f(x_s)$» в нижней части рис. 19.6. Третий шаг – случайная инициализация P_r. Мы генерируем эту популяцию таким образом, что каждая особь удовлетворяет всем ограничениям[6]. Четвертый шаг – вычисление $f(x_r)$ для каждой $x_r \in P_r$. Наконец, мы выполняем цикл эволюционного алгоритма для эволюционного формирования популяций P_s и P_r. Во время модификации популяций P_s и P_r мы можем применить любой эволюционный алгоритм для реализации отбора, рекомбинации и мутации.

Алгоритм Genocop III, по-видимому, дает хорошие результаты и, следовательно, является самым привлекательным алгоритмом для приложений и дальнейших исследований. Например, мы могли бы попробовать различные эволюционные алгоритмы для эволюционного формирования популяций P_s и P_r, и нам не нужно применять один и тот же эволюционный алгоритм для двух эволюций. Мы могли бы также поэкспериментировать с самоадаптацией ρ. Стоит отметить, что, как указано в публикации [Michalewicz и Schoenauer, 1996], новая популяция P_r не должна создаваться каждое поколение; например, нам, возможно, потребуется выполнять шаг «Сгенерировать новую популяцию P_r» на рис. 19.6 один раз в несколько итераций цикла. Это позволит уменьшить вычислительные усилия. Наконец, гибриды алгоритма Gencop III

[6] Отыскание особей, которые удовлетворяют всем ограничениям, может быть сложной задачей само по себе, но алгоритм Genocop III исходит из того, что для этого у нас есть какой-то метод. Обсуждение программирования с учетом ограничений в разделе 19.7 имеет отношение к этой задаче.

с другими ограниченными эволюционными алгоритмами вполне могут улучшить результативность.

$P_s \leftarrow$ случайно инициализированная популяция поисковых точек
Оценить $f(x_s)$ для каждого $x_s \in P_s$, как показано в алгоритме ниже.
$P_r \leftarrow$ случайно инициализированная популяция допустимых точек
Оценить $f(x_r)$ для каждого $x_r \in P_r$
While not (критерий останова)
  Применить эволюционный алгоритм со значениями $f(x_s)$
    для сгенерирования новой популяции P_s
  Оценить $f(x_s)$ для каждого $x_s \in P_s$, как показано в алгоритме ниже.
  Применить эволюционный алгоритм со значениями $f(x_r)$
    для сгенерирования новой популяции P_r
  Оценить $f(x_r)$ для каждого $x_r \in P_r$
Next поколение
~~~~~~~~~~~~~~~~
Оценить $f(x_s)$:
If $x_s \in \mathcal{F}$ **then**
  Вычислить $f(x_s)$, используя стоимостную функцию.
else
  $x'_s \leftarrow x_s$
  While $x'_s \notin \mathcal{F}$
    Случайно отобрать x_r из P_r
    Случайно сгенерировать $\alpha \sim U[0, 1]$
    $x'_s \leftarrow \alpha x_s + (1 - \alpha)x_r$
  End while
  $f(x_s) \leftarrow f(x'_s)$
  If $f(x'_s) < f(x_r)$ **then** $x_r \leftarrow x'_s$
  Случайно сгенерировать $\alpha \sim U[0, 1]$
  If $\alpha < \pi$ **then** $x_s \leftarrow x'_s$
End if

Рис. 19.6. Схематичное описание алгоритма Genocop III для минимизации $f(x)$ с учетом ограничений

19.4. Другие подходы к ограниченной оптимизации

В этом разделе кратко обсуждается несколько других подходов к ограниченной оптимизации. Данные подходы не являются подходами на основе штрафных функций, и они не связаны со специальными пред-
~~~~~~~~~~~~~~~~

ставлениями или специальными операторами, поэтому мы обсудим их в этом отдельном разделе. Раздел 19.4.1 посвящен культурным алгоритмам, раздел 19.4.2 затрагивает многокритериальную оптимизацию для ограниченных задач.

19.4.1. *Культурные алгоритмы*

Культурные алгоритмы (CA) – это эволюционные алгоритмы, в которых для управления их эволюцией используются пространства убеждений. То есть, по мере того как культурный алгоритм пытается решить оптимизационную задачу, его поиск смещается в сторону определенных направлений. Ограниченные культурные алгоритмы (в общем случае) не являются штрафными методами, поскольку штрафные методы увеличивают оценочно-стоимостную функцию недопустимых решений, в то время как ограниченные культурные алгоритмы изначально смещают поиск так, чтобы недопустимые решения вряд ли существовали в популяции. Вместе с тем использование пространств убеждений в культурном алгоритме открывает широкую область возможных подходов и реализаций из-за культурных основ культурных алгоритмов. Мы обсудили культурные алгоритмы в главе 15, и поэтому больше не будем их здесь рассматривать, но мы оставляем на усмотрение читателя вопрос исследования применений культурных алгоритмов для ограниченной оптимизации [Becerra и Coello Coello, 2004], [Coello Coello и Becerra, 2002].

19.4.2. *Многокритериальная оптимизация*

В главе 20 обсуждаются многокритериальные оптимизационные задачи (MOP). Многокритериальная оптимизация – это задача, для которой мы хотим одновременно минимизировать M стоимостных функций:

$$\min_x [f_1(x), \ldots, f_M(x)]. \tag{19.44}$$

Ограниченная оптимизационная задача может рассматриваться как многокритериальная оптимизационная задача путем определения первого целевого критерия как стоимости и остальных целевых критериев как ограничений. Рассмотрим ограниченную оптимизационную задачу, записанную в стандартной форме уравнения (19.1):

$$\begin{aligned} &\min_x f(x) \quad \text{такой, что} \quad g_i(x) \le 0 \quad \text{для} \quad i \in [1, m] \\ &\qquad\qquad \text{и} \qquad\quad h_j(x) = 0 \quad \text{для} \quad j \in [1, p]. \end{aligned} \tag{19.45}$$

Эта задача эквивалентна многокритериальной оптимизационной задаче уравнения (19.44), если

$$\begin{aligned} f_1(x) &= f(x) \\ f_1(x) &= G_1(x) \\ &\vdots \\ f_M(x) &= G_{m+p}(x) \end{aligned} \tag{19.46}$$

где $G_i(x)$ приводится в уравнении (19.8). Следовательно, мы можем применить любой алгоритм многокритериальной оптимизационной задачи для решения ограниченной оптимизационной задачи. В главе 20 обсуждаются эволюционные алгоритмы для многокритериальных оптимизационных задач. Исследования по использованию алгоритмов для многокритериальных оптимизационных задач в сфере ограниченной оптимизации можно найти в публикациях [Aguirre и соавт., 2004], [Cai и Wang, 2006], [Coello Coello, 2000A], [Coello Coello, 2002], и [Mezura-Montes и Coello Coello, 2008], помимо многих других.

19.5. Ранжирование кандидатных решений

В предыдущих разделах обсуждалось несколько способов ранжирования кандидатных решений для ограниченных оптимизационных задач. В этом разделе обобщаются ранее обсуждавшиеся подходы к ранжированию и представлено несколько альтернативных подходов. Сначала подытожим рассмотренные ранее подходы.

- Уравнение (19.8) штрафует стоимостную функцию функцией от размеров нарушений ограничений.
- Уравнение (19.10) модифицирует уравнение (19.8) таким образом, чтобы все допустимые особи имели более высокий ранг, чем все недопустимые, в то время как недопустимые особи ранжируются в соответствии с размером их нарушений ограничений. В разделе 19.2.13 этот подход также используется.
- Уравнение (19.13) ранжирует всех допустимых особей лучше, чем всех недопустимых особей, при этом ранжируя недопустимых особей на основе числа нарушений ограничений, а не размера нарушений ограничений.
- Уравнение (19.14) накладывает на стоимостную функцию штраф как за размер нарушений ограничений, так и за число нарушений ограничений.

- Уравнение (19.19) накладывает на стоимостную функцию штраф за нарушение ограничений, который увеличивается по мере увеличения числа поколений.
- Уравнения (19.23) и (19.34) накладывают штраф за нарушение ограничений, который зависит от числа допустимых особей в популяции.
- Разделы 19.2.7 и 19.2.8 корректируют размер стоимостного штрафа на основе сочетания числа допустимых особей и относительных стоимостей разных особей.
- Раздел 19.2.12 использует случайный процесс для определения того, как выполнять ранжирование кандидатных решений.

Мы уже обсудили целый ряд подходов к ранжированию для ограниченной оптимизации, и специальная литература содержит несколько других. Далее мы представим три дополнительных подхода к ранжированию.

19.5.1. *Ранжирование по максимальному нарушению ограничений*

Вместо того чтобы использовать сумму размеров нарушений ограничений или число нарушений ограничений, мы можем ранжировать особей, используя максимальный размер их нарушений ограничений [Takahama и Sakai, 2009]. В этом случае мы заменим функцию оштрафованной стоимости уравнения (19.8) на

$$\phi(x) = f(x) + \max_i G_i(x). \tag{19.47}$$

Мы также можем ранжировать кандидатные решения, используя комбинацию суммы размеров нарушений ограничений, числа нарушений ограничений и максимального размера нарушения ограничений.

19.5.2. *Ранжирование по порядку следования ограничений*

В публикации [Ray и соавт., 2009b] предлагается способ объединения размера нарушений ограничений с числом нарушений ограничений. Предположим, что x_k является k-й особью в популяции из N особей. Предположим, что у нас есть ограниченная оптимизационная задача с $m + p$ ограничениями. Мы используем $G_i(x_k)$ для обозначения размера i-го нарушения ограничения x_k с $G_i(x_k) \geq 0$. Затем применяем $c_i(x_k)$ для

обозначения ранга i-го нарушения ограничения x_k, где более низкий ранг означает меньшее нарушение ограничений, и мы задаем $c_i(x_k) = 0$, если $G_i(x_k) = 0$. Обратите внимание, что $c_i(x_k) \in [0, N]$. Приведем простой пример с популяцией размером 5 особей:

$$\left.\begin{array}{l} G_1(x_1) = 3.5 \\ G_1(x_2) = 5.7 \\ G_1(x_3) = 0.0 \\ G_1(x_4) = 1.3 \\ G_1(x_5) = 0.0 \end{array}\right\} \quad \rightarrow \quad \left\{\begin{array}{l} c_1(x_1) = 2 \\ c_1(x_2) = 3 \\ c_1(x_3) = 0 \\ c_1(x_4) = 1 \\ c_1(x_5) = 0 \end{array}\right. \tag{19.48}$$

Затем мы определяем меру нарушения ограничения как

$$v(x_k) = \sum_{i=1}^{m+p} c_i(x_k). \tag{19.49}$$

19.5.3. *Метод сравнений ϵ-уровня*

Метод сравнений ϵ-уровня аналогичен подходу на основе статического штрафа в разделе 19.2.1, в котором используются разные веса штрафных функций в зависимости от уровня нарушения ограничений. Однако при сравнении ϵ-уровня для ранжирования используются только два уровня нарушений ограничений [Takahama и Sakai, 2009].

Во-первых, мы квантифицируем нарушение ограничений $M(x)$ каждой особи x путем комбинирования всех нарушений ограничений либо путем нахождения максимального нарушения ограничений:

$$M(x) = \begin{cases} \sum_{i=1}^{m+p} G_i(x) & \text{метод суммы нарушений ограничений} \\ \max_i G_i(x) & \text{метод максимального нарушения ограничений} \end{cases} \tag{19.50}$$

Как упоминалось в разделе 19.5.1, мы можем также объединить метод суммы нарушений ограничений и метод максимального нарушения ограничений, получив $M(x)$.

Во-вторых, мы ранжируем двух особей x и y следующим образом:

$$x \text{ лучше } y, \text{ если: } \begin{cases} f(x) < f(y) \text{ и } M(x) \le \epsilon \text{ и } M(y) \le \epsilon, \text{ или} \\ f(x) < f(y) \text{ и } M(x) = M(y), \text{ или} \\ M(x) < M(y) \text{ и } M(y) > \epsilon \end{cases} \tag{19.51}$$

где $\epsilon \geq 0$ – это задаваемый пользователем порог нарушения ограничения[7]. Мы видим, что нарушение ограничения, которое меньше ϵ, считается допустимым решением для целей ранжирования. Обратите внимание, что если $\epsilon = \infty$, то особи ранжируются исключительно на основе стоимости. Если $\epsilon = 0$, то допустимые особи ранжируются на основе стоимости, недопустимые особи ранжируются исключительно на основе их нарушения ограничений, и допустимые особи всегда ранжируются лучше, чем недопустимые. Мы обычно уменьшаем ϵ по мере увеличения числа поколений, что постепенно увеличивает важность удовлетворения ограничений:

$$\begin{aligned} \epsilon(0) &= M(x_p) \\ \epsilon(t) &= \begin{cases} \epsilon(0)\,(1 - \mathrm{t}/T_c)^c & \text{если } 0 < t < T_c \\ 0 & \text{если } t \geq T_c \end{cases} \end{aligned} \tag{19.52}$$

где $\epsilon(t)$ – это значение ϵ в t-м поколении, x_p – особь с p-м наименьшим нарушением ограничения, $p = N/5$, где N – это размер популяции, а c и T_c – регулировочные параметры, которые зачастую принимаются примерно равными $c = 100$ и $T_c = t_{\max}/5$ [Takahama и Sakai, 2009]. Мы можем также попробовать другие регулировочные параметры и иные профили для уменьшения ϵ как функции от t.

19.6. Сравнение методов обработки ограничений

В этом разделе представлено сравнение между девятью методами обработки ограничений в виде неравенства. Мы используем уравнение (19.16) для измерения у кандидатного решения его размера нарушения ограничения, где $G_i(x)$ задается уравнением (19.8) с $\beta = 1$. Тестируемые нами методы обработки ограничений включают следующее.

1. EE: эклектичный эволюционный алгоритм раздела 19.2.3.
2. DP: динамический штрафной метод уравнения (19.16) в разделе 19.2.5 с $c = 10$ и $\alpha = 2$.
3. DS: динамический штрафной метод в сочетании с превосходством допустимых точек, как определено уравнением (19.18) в разделе 19.2.5, с $c = 10$ и $\alpha = 2$.

[7] Переменная ϵ – это не та же самая переменная, что в уравнении (19.6). Точно такая же переменная используется в литературе как для уравнения (19.6), так и для уравнения (19.51), и поэтому мы следуем данному именованию и в этой главе.

4. EP: экспоненциально-динамический штрафной метод уравнения (19.20) в разделе 19.2.5.1 с $\alpha = 10$.
5. ES: экспоненциально-динамический штрафной метод в сочетании с превосходством допустимых точек, определенным уравнением (19.22) в разделе 19.2.5.1, с A = 10.
6. AP: адаптивный штрафной метод уравнения (19.23) в разделе 19.2.6 с $\beta_1 = 4$, $\beta_2 = 3$ и $k = n$, где n – это размерность задачи.
7. SR: метод стохастического ранжирования раздела 19.2.12 с $P_f = 0.45$.
8. NP: нишевый штрафной метод раздела 19.2.13.
9. ϵC: метод сравнений ϵ-уровня раздела 19.5.3 с $c = 100$, $T_c = 200$ и $p = N/5$, где N – это размер популяции.

Поскольку перечисленные выше методы обработки ограничений касаются исключительно ранжирования кандидатных решений, мы можем использовать любой эволюционный алгоритм в сочетании с любым из указанных методов обработки ограничений. В этом разделе мы используем алгоритм биогеографической оптимизации (BBO), показанный на рис. 14.3, при расчете приспособленности каждой особи с помощью одного из девяти перечисленных выше методов обработки ограничений.

Мы тестируем методы обработки ограничений на эталонах CEC 2010 (Congress on Evolutionary Computation, Конгресса по эволюционным вычислениям 2010 года), перечисленных в приложении C.2, с $n = 10$ размерностям. Однако мы тестируем только те эталоны, которые не включают ограничения в виде равенства: C01, C07, C08, C13, C14 и C15.

Задачи с ограничениями в виде равенства требуют особого подхода. Как уже упоминалось в разделе 19.1.2.2, ограничения в виде равенства очень суровы. Если мы случайно сгенерируем исходную популяцию, то у нас будет практически нулевая вероятность получить любых особей, удовлетворяющих ограничениям в виде равенства. Существует два основных подхода к генерированию особей, удовлетворяющих ограничениям в виде равенства: (1) использовать специфичную для конкретных задач информацию, как это обсуждается в разделе 19.3; (2) использовать уравнение (19.8) для конвертирования ограничений в виде равенства в ограничения в виде неравенства, применять большие значения ϵ в начале работы эволюционного алгоритма и постепенно уменьшать ϵ по мере увеличения числа поколений. Хотя можно сформулировать универсальные алгоритмы оптимизации с ограничениями

в виде равенства, этот раздел посвящен обработке ограничений в виде неравенства; мы оставляем читателю возможность использовать динамические адаптации ϵ для выполнения подобных сравнений в задачах с ограничениями в виде равенства.

Мы используем популяцию размером 100 и лимит числа поколений 100. В общей сложности это дает 10 000 оцениваний функций во время симуляции эволюционного алгоритма. Обратите внимание, что во многих исследованиях в литературе при эталонном тестировании эволюционных алгоритмов используются сотни тысяч оцениваний функций (то есть сотни особей и тысячи поколений). Мы считаем, что такое большое число оцениваний функций нереалистично в большинстве случаев реальных задач. При решении реальных задач, для которых оценивание функций является относительно дорогостоящим вычислением, выполнять сотни тысяч оцениваний функций нецелесообразно. Мы призываем студентов и исследователей уделять больше внимания достижению хорошей сходимости с относительно низким числом оцениваний функций, а не пытаться достичь отличной сходимости с неоправданно высоким числом оцениваний функций. В эволюционно-алгоритмических исследованиях это позволит сократить путь от академической теории к их практическому применению. Дополнительные сведения см. в разделе 21.1.

Из-за относительно небольшого числа используемых нами оцениваний функций приведенные в данном разделе результаты несопоставимы со многими опубликованными результатами для эталонов CEC 2010. Но дело здесь не в том, чтобы попытаться достичь лучшей возможной результативности при неоправданно большом числе оцениваний функций, а в том, чтобы сравнить методы обработки ограничений на ровном игровом поле. Наш выбор из 100 особей и 100 поколений является хорошим компромиссом, потому что для выполнения таких симуляций не требуется слишком много времени, но при этом дает различным эволюционным алгоритмам и методам обработки ограничений достаточно времени, для того чтобы себя проявить.

Мы реализуем мутацию путем замены признака в особи значением, случайно отбираемым из равномерного распределения в поисковой области. Каждое поколение каждый признак в каждой особи имеет 1%-ную вероятность мутации. Мы также используем параметр элитарности 2, что означает, что из поколения в поколение мы оставляем двух лучших особей.

Для целей элитарности мы определяем лучшую особь как допустимую особь с наименьшей стоимостью. Если ни одной допустимой особи

нет, то мы определяем лучшую особь как особь с самой низкой оштрафованной стоимостью, при этом получаем оштрафованную стоимость, используя один из перечисленных выше девяти методов. Мы используем этот подход, потому что некоторые перечисленные выше методы обработки ограничений ранжируют недопустимых особей лучше, чем допустимых. Мы можем себе позволить такой подход, когда имеется относительно большое число особей и ранжирование приводит к отбору для рекомбинации. Но если же из поколения в поколение мы сохраняем только двух элитарных особей, то должны обеспечить, чтобы допустимые особи всегда были предпочтительнее недопустимых. Этим гарантируется, что когда эволюционный алгоритм находит допустимую особь, у него всегда будет, по крайней мере, одна допустимая особь для остальной части симуляции.

В табл. 19.1 приведено сравнение девяти методов обработки ограничений на шести эталонах ограниченной оптимизации CEC 2010, которые не имеют ограничений в виде равенства. Результаты в таблице усреднены по 20 симуляциям Монте-Карло. Мы случайно сгенерировали сдвинутые значения $\{o_i\}$ для эталонов в приложении C.2, но использовали те же $\{o_i\}$ для всех методов обработки ограничений в конкретном испытании Монте-Карло. В некоторых эталонах используется матрица поворота M, которую мы произвольно генерируем так же, как и сдвинутые значения.

Таблица 19.1. Сравнение лучших допустимых значений стоимостной функции, найденных девятью алгоритмами BBO с обработкой ограничений на шести 10-мерных эталонных задачах, усредненно по 20 симуляциям Монте-Карло. См. список определений аббревиатур в начале раздела 19.6. Лучшая стоимость по каждому эталону выделена **жирным** шрифтом

	C01	C07	C08	C13	C14	C15
EE	−0.46	**19800**	2.39×10^5	∞	5.78×10^{13}	6.911×10^{13}
DP	−0.46	31900	1.51×10^5	−600	1.64×10^{13}	0.066×10^{13}
DS	−0.45	24700	1.46×10^5	−601	2.34×10^{13}	0.228×10^{13}
EP	−0.44	22000	**1.03×10^5**	−593	**0.01×10^{13}**	**0.001×10^{13}**
ES	**−0.47**	56800	1.32×10^5	**−606**	**0.01×10^{13}**	0.005×10^{13}
AP	−0.46	21000	1.64×10^5	−599	0.13×10^{13}	0.072×10^{13}
SR	−0.46	50500	1.25×10^5	−592	4.93×10^{13}	0.231×10^{13}
NP	−0.46	30900	1.97×10^5	−596	0.99×10^{13}	0.353×10^{13}
ϵC	−0.46	30900	7.68×10^5	−604	2.74×10^{13}	0.160×10^{13}

В табл. 19.1 показаны некоторые интересные свойства. Во-первых, отметим, что все алгоритмы работают аналогично эталонной задаче C01. Это указывает на то, что эталонная задача C01 либо очень легкая, то есть любой метод работает хорошо, либо она очень трудная, то есть ни один метод не работает хорошо. Во-вторых, отметим, что все алгоритмы, кроме EE, работают примерно одинаково с эталонной задачей C13; алгоритм EE не может найти допустимое решение для C13 даже после 20 прогонов Монте-Карло, и поэтому его значение стоимостной функции записывается как ∞. Алгоритм EE хуже всего работает с эталонными задачами C13, C14 и C15, что вызывает интерес, поскольку эти три эталона являются наиболее сложными для выполнения (см. табл. C.1 в приложении C.2).

В табл. 19.1 показано, что в среднем подход на основе экспоненциального штрафа уравнения (19.20) в разделе 19.2.5.1 работает лучше. Вместе с тем существует ряд ограниченных оптимизационных задач, которые обсуждались в литературе, и упомянутые выше методы обработки ограничений имеют несколько регулировочных параметров. Если выполнять тесты с другими регулировочными значениями и другими эталонными функциями, то вполне могут быть сделаны разные заключения.

19.7. Заключение

Из этой главы мы видим, что существует ряд методов обработки ограничений, которые можно использовать с эволюционными алгоритмами. Многие из этих алгоритмов имеют схожие уровни результативности. Вместо того чтобы испытывать все эти алгоритмы в поисках лучшего метода, будет лучше запомнить некоторые основные принципы, которые могут быть важны при решении ограниченных оптимизационных задач.

Важные принципы ограниченной оптимизации

- Как и в случае с неограниченными оптимизационными задачами, чем больше специфичной для конкретной задачи информации мы можем встроить в эволюционный алгоритм, тем выше наши шансы на успех. Обработка ограничений с помощью специальных представлений или специальных операторов обычно эффективнее, чем использование универсального подхода. Инструменты оптимизации по принципу черного ящика просты в использовании, и иногда они необходимы, но мы почти всег-

да можем получить более высокую результативность с помощью трудных в получении экспертных знаний предметной области.

- При наличии ограниченной оптимизационной задачи мы должны квантифицировать ее сложность в удовлетворении ограничений. Эту сложность можно измерить с помощью параметра

$$\rho = |\mathcal{F}| / |S|, \tag{19.53}$$

где $|\mathcal{F}|$ – это размер допустимого множества и $|S|$ – размер поискового пространства. Мы можем аппроксимировать ρ, случайно генерируя большое число особей в поисковой области и проверяя, сколько из них удовлетворяют ограничениям. На данном этапе нам не нужно оценивать функцию стоимости; цель этой работы состоит в том, чтобы увидеть, насколько ограничения сложны. Если ограничения удовлетворяются лишь крошечным процентом случайных особей, то такие ограничения сложны, и наш ограниченный эволюционный алгоритм должен сосредоточиться на удовлетворении ограничений. Если ограничения удовлетворяются достаточно высоким процентом случайных особей, то такие ограничения довольно просты, и наш ограниченный эволюционный алгоритм может больше сосредоточиться на минимизации функции стоимости.

- Мы должны квантифицировать сложность удовлетворения *индивидуальных* ограничений. Как указано выше, это может быть сделано путем случайной генерации большого числа особей в поисковой области. Ограничения, которые удовлетворяются очень малым числом случайных особей, являются сложными, и поэтому ограниченный эволюционный алгоритм должен сосредоточиться на удовлетворении этих ограничений. Ограничения, которые удовлетворяет относительно большим числом особей, являются легкими, и поэтому ограниченный эволюционный алгоритм не должен на них сосредоточиваться. В качестве альтернативы мы можем нормализовать ограничения так, чтобы удовлетворение каждого ограничения было одинаково трудным.
- Одной из самых больших проблем во многих ограниченных оптимизационных задачах является поиск допустимых решений. Это в особенности касается задач, связанных с ограничениями в виде равенства. В этом случае мы, возможно, захотим выполнить алгоритмы удовлетворения ограничений перед или вместо выполнения ограниченного эволюционного алгоритма. Алгорит-

мы удовлетворения ограничений относятся к области исследований, именуемой программированием с учетом ограничений (constraint programming). Программирование с учетом ограничений выходит за рамки этой книги, но является важной областью исследования, связанной с ограниченной оптимизацией. Любой, кто серьезно заинтересован в ограниченной оптимизации, должен изучить программирование с учетом ограничений. Некоторые хорошие вводные сведения по данной теме доступны в публикациях [Dechter, 2003], [Marriott и Stuckey, 1998] и [Rossi и соавт., 2006].

- Несмотря на вышеизложенные моменты, сложность удовлетворения ограничений не обязательно является показателем сложности ограниченной оптимизационной задачи. Некоторые задачи с относительно небольшим допустимым участком не являются для ограниченных эволюционных алгоритмов сложными. Например, в публикации [Michalewicz и Schoenauer, 1996] сообщается о двух задачах с $\rho = 0{,}0111\ \%$ и $\rho = 0{,}0003\ \%$ как относительно легких для ограниченных эволюционных алгоритмов.
- При выполнении ограниченного эволюционного алгоритма мы должны отслеживать число особей, удовлетворяющих ограничениям из поколения в поколение. Популяции, в которых все особи допустимы, часто неэффективны, поэтому мы должны позаботиться о том, чтобы включать в наш поиск ограниченного оптимума как допустимых, так и недопустимых особей.

Текущие и будущие исследования в области ограниченных эволюционных алгоритмов

Ограниченная эволюционная оптимизация является активной областью исследований, поскольку: (1) она является относительно новой областью; (2) в ней отсутствуют теоретические результаты (что характерно для этой главы, которая не содержит никаких теоретических результатов); (3) реальные оптимизационные задачи почти всегда ограничены. Мы завершаем эту главу упоминанием некоторых популярных и важных направлений современных исследований.

- Большая часть текущих исследований включает встраивание стандартных методов обработки ограничений, таких как те, которые обсуждались в этой главе, в новые эволюционные алгоритмы. В литературе постоянно появляются новые эволюционные алгоритмы. Эти новые эволюционные алгоритмы часто являются не

более чем модификациями старых эволюционных алгоритмов, но иногда они имеют отличительные новые функциональные особенности и возможности (см. главу 17). Важно изучить, насколько хорошо работают текущие методы обработки ограничений при их включении в разные типы эволюционных алгоритмов. Относительные результативности разных эволюционных алгоритмов на неограниченных задачах не обязательно коррелируют с их относительными результативностями на ограниченных задачах.

- В этой главе обсуждаются ограниченные оптимизационные задачи, и в главе 20 обсуждаются многокритериальные оптимизационные задачи. Современные исследования начинают объединять эти две области для поиска алгоритмов решения ограниченных многокритериальных оптимизационных задач [Yen, 2009).
- Получение теоретических результатов будет весьма плодотворной сферой будущих исследований в области ограниченной оптимизации. В этой книге обсуждаются марковские модели, модели динамических систем и теория схем для генетических алгоритмов и генетического программирования (GP). Возможно, эти или другие инструменты могут также использоваться для анализа ограниченных эволюционных алгоритмов.
- С приведенным выше обсуждением программирования с учетом ограничений связана идея поиска границы ограничений для решения ограниченных оптимизационных задач [Leguizamón и Coello Coello, 2009]. Поиск границ имеет отношение к программированию с учетом ограничений, но программирование с учетом ограничений сосредоточено на поиске допустимых решений, в то время как поиск границ сосредоточен на эволюционном формировании популяции, которая лежит на границе ограничений.
- Как упоминалось ранее в этой главе, некоторые задачи имеют ограничения, которые трудно удовлетворить, поэтому крайне сложно находить допустимые участки в поисковом пространстве. Вместе с тем, помимо задачи поиска допустимых участков, существует крайне сложная задача проектирования эволюционного алгоритма, который может эффективно колебаться между допустимыми и недопустимыми участками. Этот тип поведения часто желателен для задач, решение которых лежит на границе ограничений [Schoenauer и Michalewicz, 1996].
- Как и в случае эволюционных алгоритмов, которые можно комбинировать различными способами, методы обработки ограни-

чений тоже можно комбинировать. Например, ансамбль методов обработки ограничений может использовать одни и те же результаты функций стоимости, и лучший метод в каждом поколении будет доминировать в следующем поколении [Mallipeddi и Suganthan, 2010]. Это похоже на некоторые из многокритериальных алгоритмов главы 20, в которых на разных этапах процесса оптимизации используются разные функции стоимости.

- В качестве еще одного уровня абстракции за пределами ансамблей гиперэвристики объединяют несколько эволюционных алгоритмов и несколько методов обработки ограничений в один алгоритм. Напомним, что эвристика – это семейство алгоритмов (например, семейство вариаций алгоритма оптимизации на основе муравьиной кучи (ACO) или семейство вариаций алгоритма дифференциальной эволюции (DE)). Гиперэвристика – это семейство семейств алгоритмов (например, семейство, содержащее эвристику ACO, эвристику DE и другие эвристики). Гиперэвристики могут использоваться для любого типа оптимизационных задач, но мы упоминаем их здесь из-за их многообещающих перспектив для ограниченных задач [Tinoco и Coello Coello, 2013].

Обзоры ограниченной оптимизации можно найти в публикациях [Eiben, 2001], [Coello Coello, 2002] и [Coello Coello и Mezura-Montes,, 2011]. Читатель, который заинтересован в дальнейших исследованиях, должен отметить, что Карлос Коэльо Коэльо (Carlos Coello Coello) ведет библиографию статей, связанных с ограниченной эволюционной оптимизацией, которая по состоянию на август 2012 года включала в себя 1036 ссылок [Coello Coello, 2012a].

Задачи

Письменные упражнения

19.1. Во многих эталонах с ограничениями в виде равенства используется $\epsilon \approx 0.0001$ в уравнении (19.7). Какова вероятность удовлетворения скалярного ограничения $|x| \leq \epsilon$ с этим значением ϵ и со случайно сгенерированным $x \in [-1000, +1000]$?

19.2. Данная задача показывает, как метод превосходства допустимых точек, в котором используется уравнение (19.11), успешно работает, если $f(x) \geq 0$ для всех x, но как он может потерпеть неудачу, если это допущение не выполняется. Используйте уравнения

(19.10) и (19.11), для того чтобы найти $\phi'(x)$ для двухэлементной популяции со следующими характеристиками:

a) $f(x_1) = 0, \quad \sum_i r_i G_i(x_1) = 1$

$f(x_2) = 10, \quad \sum_i r_i G_i(x_2) = 0$

b) $f(x_1) = -10, \quad \sum_i r_i G_i(x_1) = 1$

$f(x_2) = 0, \quad \sum_i r_i G_i(x_2) = 0$

19.3. Данная задача показывает, как работает метод превосходства допустимых точек, в котором используется уравнение (19.12). Используйте уравнения (19.10) и (19.12), для того чтобы найти $\phi'(x)$ для двухэлементных популяций, показанных в задаче 19.2.

19.4. Приведите аналитическое выражение для наименьшего значения K в эклектическом эволюционном алгоритме уравнения (19.13), которое гарантирует, что $\phi(\bar{x}) > \phi(x)$ для всех $\bar{x} \notin \mathcal{F}$ и для всех $x \in \mathcal{F}$.

19.5. Предположим, что у нас в эволюционно-алгоритмической популяции есть 4 особи со следующими ниже значениями стоимости и уровнями нарушения ограничений:

$$\begin{array}{lll} f(x_1) = 3, & G_1(x_1) = 0, & G_2(x_1) = 0, \\ f(x_2) = 2, & G_1(x_2) = 1, & G_2(x_2) = 0, \\ f(x_3) = 1, & G_1(x_3) = 1, & G_2(x_3) = 1, \\ f(x_4) = 4, & G_1(x_4) = 0, & G_2(x_4) = 0. \end{array}$$

Примените метод самоадаптивной формулировки приспособленности раздела 19.2.8, для того чтобы для этих особей найти значения оштрафованной стоимости. Дайте интуитивное объяснение вашего ответа.

19.6. Предположим, что у нас в эволюционно-алгоритмической популяции есть 4 особи со значениями стоимости и уровнями нарушения ограничений, показанными в задаче 19.5. Примените метод на основе адаптивной штрафной функции раздела 19.2.9, для того чтобы для этих особей найти значения оштрафованной стоимости. Дайте интуитивное объяснение вашего ответа.

19.7. Данная задача задействует алгоритм адаптивной сегрегационной обработки ограничений раздела 19.2.10.

a) Объясните, как алгоритм обновления r_i уравнения (19.34) достигает целевого отношения допустимых особей к недопустимым особям.

b) Объясните эффект увеличения γ в уравнении (19.34).

19.8. Несколько ограниченных эволюционных алгоритмов, включая алгоритм стохастического ранжирования раздела 19.2.12 и нишевый штрафной подход раздела 19.2.13, включают в себя сравнение особей, для того чтобы увидеть, какая из них имеет «меньшее нарушение ограничения». Предложите три способа, которыми размер нарушения ограничения мог бы быть измерен.

19.9. Задача коммивояжера (TSP) является ограниченной задачей: для того чтобы маршрут считался допустимым, кандидатное решение должно посетить каждый город ровно один раз. Операторы скрещивания для представления особей задачи коммивояжера в виде пути обсуждаются в разделе 18.3.1. Какие из этих операторов соблюдают данное ограничение задачи коммивояжера, а какие нет?

19.10. Предположим, что у нас в эволюционно-алгоритмической популяции есть 4 особи со значениями стоимости и уровнями нарушения ограничений, показанными в задаче 19.5. Предположим, что мы используем сравнения ϵ-уровня уравнения (19.51) в сочетании с методом суммы нарушений ограничений.

a) Для каких значений ϵ x_2 будет ранжироваться лучше, чем x_1?

b) Для каких значений ϵ x_3 будет ранжироваться лучше, чем x_1?

c) Для каких значений ϵ x_2 будет ранжироваться лучше, чем x_3?

19.11. Сколько ссылок приводится на веб-сайте Карлоса Коэльо Коэльо в списке «List of References on Constraint-Handling Techniques used with Evolutionary Algorithms» (Список ссылок на методы обработки ограничений, используемых вместе с эволюционными алгоритмами)?

Компьютерные упражнения

19.12. Воссоздайте рис. 19.1 с α = 0.5. Какая разница между вашим рисунком и рис. 19.1?

19.13. Предположим, что у нас есть круговая поисковая область радиусом 1 единица, центрированная в начале координат. Предположим, что особи ограничены расположением в окружности с радиусом $\rho_c = 0.1$ единицы, также центрированной в начале координат.

a) Примените алгоритм Genocop III для генерации случайной допустимой x_r, случайной недопустимой x_s и случайного параметра $\alpha \in [0, 1]$, для того чтобы создать потенциально исправленную особь x'_s. Выполните данный эксперимент большое число раз, для того чтобы оценить вероятность того, что x'_s является допустимой. Повторите для $\rho_c = 0.5$ единицы.

b) Повторите часть (a) для сферической области.

19.14. В разделе 19.6 сравниваются девять методов обработки ограничений в сочетании с алгоритмом биогеографической оптимизации (BBO). Сравните несколько методов обработки ограничений этой главы на одной или нескольких ограниченных оптимизационных задачах с использованием эволюционного алгоритма, отличного от BBO.

Глава 20

Многокритериальная оптимизация

В большинстве реальных сценариев оптимизации естественным образом возникают многочисленные, часто конфликтующие цели.

– Эккарт Цицлер [Zitzler и соавт., 2004]

Все реальные оптимизационные задачи являются многокритериальными, по крайней мере если не эксплицитно, то имплицитно. В данной главе обсуждается вопрос модифицирования эволюционных алгоритмов для многокритериальных оптимизационных задач (multi-objective optimization problem, MOP). Как утверждает цитата в начале данной главы, реальные оптимизационные задачи обычно (возможно, всегда) включают в себя несколько целевых критериев, и эти целевые критерии обычно конфликтуют. Например:

- при проектировании моста мы, возможно, захотим минимизировать его стоимость и максимизировать прочность. Мост с минимальной стоимостью вполне мог бы быть сделан из пенополистирола и был бы очень слабопрочным. Мост с максимальной прочностью вполне мог бы быть сделан из титана и был бы очень дорогим. Каков лучший компромисс между стоимостью и прочностью?
- Приобретая автомобиль, мы, возможно, захотим максимизировать комфорт и минимизировать стоимость. Автомобиль с максимальным комфортом был бы слишком дорогим, но автомобиль с минимальной стоимостью был бы слишком неудобным.

- При разработке потребительского изделия мы, возможно, захотим максимизировать прибыль и увеличить долю рынка. Изделие с максимальной прибылью не обеспечит достаточного проникновения на рынок, чтобы спозиционировать нашу компанию для будущих изделий, но продукт с максимальной долей рынка не приведет к достаточной прибыли.
- При проектировании системы управления мы, возможно, захотим минимизировать время нарастания и минимизировать проскакивание. Регулятор с минимальным временем нарастания будет иметь слишком большое проскакивание, но критически затухающий регулятор (с нулевым проскакиванием) не будет иметь достаточно быстрого времени нарастания.
- При проектировании системы управления мы, возможно, захотим максимизировать чувствительность к входному синалу и минимизировать чувствительность к помехам. Регулятор с максимальной чувствительностью к входному сигналу будет слишком чувствителен к шуму, а регулятор с минимальной чувствительностью к помехам будет слишко реагировать на управляющие входные сигналы.

Многокритериальная оптимизация также называется многоцелевой, многообъектной, мультирезультативной (multi-performance) и векторной оптимизацией. В этой главе мы будем считать, что независимая переменная x является n-мерной, и мы исходим из того, что наша многокритериальная оптимизационная задача является минимизационной задачей. Многокритериальную оптимизационную задачу можно записать следующим образом:

$$\min_x f(x) = \min_x [f_1(x), f_2(x), \ldots, f_k(x)]. \qquad (20.1)$$

То есть мы хотим минимизировать вектор $f(x)$ функций. Разумеется, мы не можем минимизировать вектор в типичном смысле слова «минимизировать». Тем не менее наша цель в многокритериальной оптимизационной задаче состоит в том, чтобы одновременно минимизировать все k функций $f_i(x)$. Мы видим, что для многокритериальных оптимизационных задач мы должны пересмотреть наше определение оптимальности.

Многокритериальная оптимизация изучалась сообществом исследователей операций на протяжении многих лет [Ehrgott, 2005]. Похоже, что публикация [Rosenberg, 1967] стала первой, в которой для многокритериальных оптимизационных задач было предложено использо-

вать эволюционные алгоритмы, в публикации [Ito и соавт., 1983] была описана первая реализация, и публикация [Schaffer, 1985] была первой широко известной исследовательской работой по этой теме.

Многокритериальные оптимизационные задачи часто включают ограничения, но, как мы видим из постановки задачи уравнения (20.1), в этой главе мы имеем дело не с ограниченными многокритериальными оптимизационными задачами. Мы можем включать ограничения в многокритериальные эволюционные алгоритмы (multi-objective evolutionary algorithm, MOEA) так же, как включаем их в однокритериальные эволюционные алгоритмы (см. главу 19). Некоторые исследователи предложили методы обработки ограничений, которые являются уникальными для многокритериальных оптимизационных задач, но в этой главе мы их касаться не будем.

Краткий обзор главы

В разделе 20.1 обсуждается понятие оптимальности по Парето, являющееся расширением оптимальности до многокритериальных оптимизационных задач в форме уравнения (20.1). Поскольку многокритериальная оптимизационная задача преследует несколько целевых критериев, существует целый ряд способов измерения результативности таких задач, и некоторые из этих способов мы обсудим в разделе 20.2. Вслед за этим обсуждением мы представим несколько популярных многокритериальных оптимизационных задач. В разделе 20.3 обсуждаются многокритериальные оптимизационные задачи, в которых понятие оптимальности по Парето не используется явно, и в разделе 20.4 рассматриваются такие задачи, в которых понятие оптимальности по Парето используется явно. В разделе 20.5 показано, как сочетать биогеографическую оптимизацию (BBO; см. главу 14) с некоторыми подходами на основе многокритериальных эволюционных алгоритмов (MOEA) этой главы, и продемонстрировано сравнительное исследование на нескольких многокритериальных эталонах. В заключительном разделе этой главы приводятся ссылки на дополнительные ресурсы и предлагается ряд важных тем для текущих и будущих исследований в области многокритериальных эволюционных алгоритмов.

20.1. Оптимальность по Парето

В этом разделе описывается несколько основных понятий и примеров, которые относятся к многокритериальным оптимизационным задачам.

Перечислим несколько определений, которые часто используются в многокритериальной оптимизации.

1. *Доминирование*: точка x^* считается доминирующей над x, если выполняются два следующих условия: (1) $f_i(x^*) \le f_i(x)$ для всех $i \in [1, k]$ и (2) $f_j(x^*) < f_j(x)$, по крайней мере, для одного $j \in [1, k]$. То есть x^*, по крайней мере, так же хороша, как x для всех значений целевой функции, и это лучше, чем x, по крайней мере для одного значения целевой функции. Для обозначения того, что x^* доминирует над x, мы используем запись

$$x^* \succ x. \tag{20.2}$$

   Такая запись может ввести в заблуждение, потому что символ $\succ$ похож на символ «больше», но поскольку в этой главе мы имеем дело в основном с минимизационными задачами, символ $\succ$ означает, что значения x^* целевой функции меньше или равны x. Тем не менее эта запись является в литературе стандартной, и поэтому мы ее здесь используем. Утверждение «x^* превосходит x» идентично утверждению «x^* доминирует над x».

2. *Слабое доминирование*: точка x^* считается слабо доминирующей над x, если $f_i(x^*) \le f_i(x)$ для всех $i \in [1, k]$. То есть x*, по крайней мере, так же хороша, как x для всех значений целевой функции. Заметим, что если точка x^* доминирует над x, то она также слабо доминирует над x. Кроме того, обратите внимание, что если $f_i(x^*) = f_i(x)$ для всех $i \in [1, k]$, то x^* и x слабо доминируют друг над другом. Для обозначения того, что x^* слабо доминирует над x, мы используем запись

$$x^* \succcurlyeq x. \tag{20.3}$$

   Некоторые авторы используют эквивалентную терминологию, говоря, что x^* охватывает (покрывает) x.

3. *Недоминированный*: точка x^* называется недоминированной, если нет точки x, которая доминирует над ней. Термины «неподчиненный», «приемлемый» и «эффективный» являются синонимами термина «недоминированный».

4. *Оптимальные по Парето точки*: оптимальная по Парето точка x^*, так называемая точка Парето, – это точка, над которой не доминирует никакая другая x в поисковом пространстве. То есть

$$\begin{aligned} &x^* \text{ является оптимальной по Парето} \Leftrightarrow \\ &\nexists x : (f_i(x) \le f_i(x^*) \text{ для всех } i \in [1, k], \text{ и} \\ &\quad f_j(x) < f_j(x^*) \text{ для некоторых } j \in [1, k]). \end{aligned} \tag{20.4}$$

5. *Оптимальное по Парето множество*: оптимальное по Парето множество, так называемое множество Парето и обозначаемое как P_s, – это множество всех недоминированных x^*.

$$P_s = \{x^* : [\nexists x : (f_i(x) \leq f_i(x^*) \text{ для всех } i \in [1, k], \text{ и } f_j(x) < f_j(x^*) \text{ для некоторых } j \in [1, k])]\}. \quad (20.5)$$

Множество Парето также называется эффективным множеством, и его иногда называют приемлемым множеством, хотя этот термин обычно подразумевает удовлетворение ограничений, а не оптимальность по Парето.

6. *Фронт Парето*: фронт Парето, так называемое недоминированное множество и обозначаемое как P_f, – это множество всех векторов $f(x)$ функций, соответствующих множеству Парето.

$$P_f = \{f(x^*) : x^* \in P_s\}. \quad (20.6)$$

В литературе иногда неправильно или взаимозаменяемо используются термины «множество Парето» и «фронт Парето», но приведенный выше список терминов дает технически правильные определения. Обратите внимание, что утверждение, что x^* недоминирована, не обязательно означает, что x^* доминирует над всеми x, которые не равны x^*. Может быть так, что $f_i(x^*) = f_i(x)$ для всех $i \in [1, k]$. В этом случае как x, так и x^* являются недоминированными по отношению друг к другу, но ни одна из них не доминирует над другой. Возможен и такой случай, когда, например, в двухкритериальной задаче $f_1(x) < f_1(x^*)$ и $f_2(x^*) < f_2(x)$. Опять же, в данном случае как x, так и x^* являются недоминированными по отношению друг к другу, но ни одна из них не доминирует над другой.

Эта идея оптимальности по Парето для многокритериальных оптимизационных задач часто приписывается Фрэнсису Эджуорту, который представил ее в 1881 году [Edgeworth, 1881], и Вильфредо Парето, который обобщил работу Эджуорта в 1896 году [Pareto, 1896]. Однако идея компромисса является общей для всех, кто когда-либо пытался сбалансировать конфликтующие целевые критерии.

■ Пример 20.1

Предположим, что у нас есть многокритериальная оптимизационная задача, для которой независимая переменная x двумерна ($n = 2$), и что x может принимать одно из шести дискретных значений $x^{(i)}$, где $i \in [1,6]$. Далее предположим, что у нас есть два целевых критерия ($k = 2$) со значениями функций

$$
\begin{aligned}
f_1(x^{(1)}) &= 1, \quad & f_2(x^{(1)}) &= 3,\\
f_1(x^{(2)}) &= 1, \quad & f_2(x^{(2)}) &= 4,\\
f_1(x^{(3)}) &= 2, \quad & f_2(x^{(3)}) &= 2,\\
f_1(x^{(4)}) &= 2, \quad & f_2(x^{(4)}) &= 3,\\
f_1(x^{(5)}) &= 3, \quad & f_2(x^{(5)}) &= 1,\\
f_1(x^{(6)}) &= 3, \quad & f_2(x^{(6)}) &= 3.
\end{aligned} \tag{20.7}
$$

Если $x = x^{(1)}$ или $x = x^{(2)}$, то $f_1(x)$ минимизирована. Если $x = x^{(5)}$, то $f_2(x)$ минимизирована. Нет ни одного значения x, которое минимизирует как $f_1(x)$, так и $f_2(x)$. Оптимальное значение x есть значение, которое обеспечивает лучший компромисс между несколькими целевыми критериями, где понятие «лучшее» основывается на нашем суждении, зависящем от конкретной задачи. Еще один интересный момент в уравнении (20.7) – это $x^{(3)}$, так как любая точка $x \neq x^{(3)}$ дает либо $f_1(x) > f_1(x^{(3)})$, либо $f_2(x) > f_2(x^{(3)})$.

Хорошим способом визуализации этой задачи является просмотр графика f_2 в сопоставлении с f_1, как показано на рис. 20.1. Он ясно показывает, что для $x \in \{x^{(1)}, x^{(3)}, x^{(5)}\}$ нет другого x, который одновременно уменьшает все значения целевой функции. Поэтому для данной многокритериальной оптимизационной задачи $x^{(1)}$, $x^{(3)}$ и $x^{(5)}$ являются хорошими компромиссными значениями, и они составляют множество Парето. Если соединить все оптимальные точки в плоскости f_1/f_2 на рис. 20.1, то мы получим кривую, которая образует нижнюю границу для всех других точек на плоскости f_1/f_2.

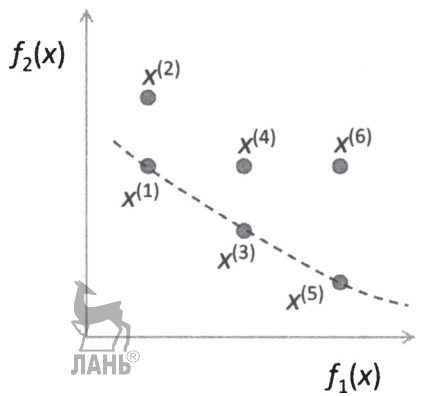


Рис. 20.1. Пример 20.1: $\{x^{(1)}, x^{(3)}, x^{(5)}\}$ образуют множество Парето для этой многокритериальной минимизационной задачи. Соответствующие множеству Парето векторы функций образуют фронт Парето

Еще одним способом визуализации этой задачи является просмотр графика поискового пространства, в котором множество Парето обозначено специальной записью. На рис. 20.2 показана одна из возможностей для поискового пространства в данном примере, где точки Парето обозначены звездочками. Данный график показывает участок поискового пространства, который соответствует фронту Парето на рис. 20.1.

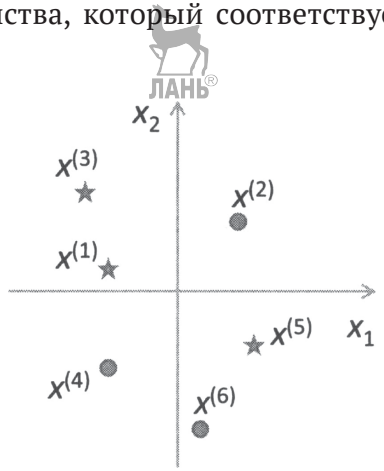


Рис. 20.2. Пример 20.1: на этом рисунке показано возможное двумерное поисковое пространство для примера 20.1. Поисковое пространство состоит из шести двумерных векторов. Звездочки указывают на множество Парето

■

■ Пример 20.2

Рассмотрим многокритериальную оптимизационную задачу

$$\min f(x) = \min[f_1(x), f_2(x)] = \min[x_1^2 + x_2^2, (x_1 - 2)^2 + (x_2 - 2)^2], \qquad (20.8)$$

где $x_1 \in [0, 2]$ и $x_2 \in [0, 2]$. Данная многокритериальная оптимизационная задача ($n = 2$) является двумерной, потому что каждый x в поисковом пространстве имеет два элемента. Эта многокритериальная оптимизационная задача также имеет два целевых критерия ($k = 2$). Точка $x^{(1)} = (0, 0)$ минимизирует $f_1(x)$, и поэтому (0, 0) является одной из точек Парето. Точка $x^{(2)} = (2, 2)$ минимизирует $f_2(x)$, и поэтому (2, 2) также является одной из точек Парето. Если мы применим поиск методом грубой силы, чтобы найти все точки Парето, то обнаружим, что множеством Парето является

$$P_s = \{x : x_1 = x_2, \text{ где } x_1 \in [0, 2]\}. \qquad (20.9)$$

То есть множество Парето образует прямую в поисковом пространстве. Мы можем найти фронт Парето, подставив точки Парето в уравнение (20.8):

$$P_f = \{(f_1, f_2) : f_1 = 2x_1^2, f_2 - 2(x_1 - 2)^2, \text{ где } x_1 \in [0, 2]\}. \qquad (20.9)$$

На рис. 20.3 показаны множество Парето и фронт Парето для приведенного выше примера. Любая точка, кроме точки Парето, отображается в вектор функции, который находится выше и справа от фронта Парето.

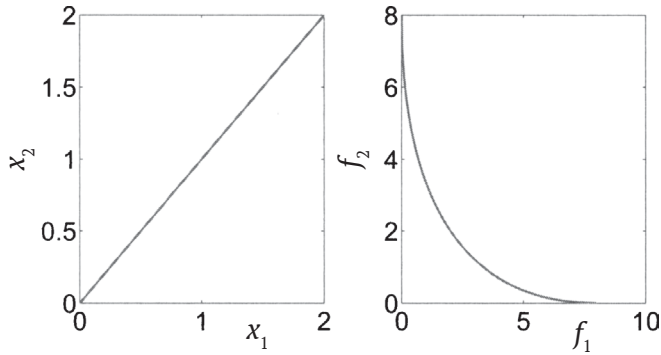


Рис. 20.3. Пример 20.2: на рисунке слева показано множество Парето. На рисунке справа показан соответствующий фронт Парето. Любая точка во множестве Парето обеспечивает разумный компромисс для многокритериальной оптимизационной задачи

■

ϵ-доминирование

Одним из ограничений идеи доминирования по Парето является ее черно-белая природа. Например, рассмотрим следующие три множества значений функции стоимости:

$$\begin{aligned} f_1(x) &= 200, \quad f_2(x) = 300 \\ f_1(y) &= 201, \quad f_2(y) = 301 \\ f_1(z) &= 500, \quad f_2(z) = 600 \end{aligned} \qquad (20.11)$$

x доминирует над *y* и *z*. Но идея доминирования по Парето не допускает какого-либо различия между уровнем доминирования и не распознает два кандидатных решения, которые очень близки друг другу в пространстве целевой функции. В уравнении (20.11) *x* доминирует над *y*, но так как *x* и *y* сильно похожи, они являются почти недомини-

рованными по отношению друг к другу. Фактически мы можем сказать, что y доминирует над x. Это порождает идею ϵ-доминирования.

1. *Аддитивное ϵ-доминирование*: точка x^* считается аддитивно ϵ-доминирующей над x, если $f_i(x^*) \leq f_i(x) + \epsilon$ для некоторого $\epsilon \geq 0$ и для всех $i \in [1, k]$. То есть x^* находится «близко» к доминированию над x, где «близость» квантифицируется параметром ϵ аддитивно.
2. *Мультипликативное ϵ-доминирование*: точка x^* называется мультипликативно ϵ-доминирующей над x, если $f_i(x^*) \leq f_i(x)\ (1 + \epsilon)$ для некоторого $\epsilon \geq 0$ и для всех $i \in [1, k]$. То есть x^* находится «близко» к доминированию над x, где «близость» квантифицируется параметром ϵ мультипликативно.

Мы используем запись

$$(x^* \succ_\epsilon x) \tag{20.12}$$

для обозначения того, что x^* ϵ-доминирует над x, где тип эпсилон-доминирования (аддитивный или мультипликативный) должен быть ясен из контекста. Обратите внимание, что если $\epsilon = 0$, то ϵ-доминирование эквивалентно слабому доминированию:

$$(x^* \succ_\epsilon x) \ \text{ для } \epsilon = 0 \ \Leftrightarrow \ (x^* \succcurlyeq x). \tag{20.13}$$

Также обратите внимание, что ϵ-доминирование для $\epsilon > 0$ является еще более слабым типом доминирования, чем слабое доминирование. То есть если x^* слабо доминирует над x, то x^* также доминирует над x для всех $\epsilon \geq 0$. И наоборот, если x^* e-доминирует над x для некоторого e > 0, то x^* может, а может и нет, слабо доминировать над x:

$$(x^* \succcurlyeq x) \ \Rightarrow \ (x^* \succ_\epsilon x) \quad \text{для всех } \epsilon > 0. \tag{20.14}$$

Отношения ϵ-доминирования между двумя особями зависят от значения ϵ, которое мы используем в нашем определении. В уравнении (20.11) $x \succ_\epsilon y$ для всех $\epsilon \geq 0$. Также $y \succ_\epsilon x$ справделиво в аддитивном смысле, если $\epsilon \geq 1$, и справедливо в мультипликативном смысле, если $\epsilon \geq 0.005$.

20.2. Цели многокритериальной оптимизации

Цель однокритериального оптимизационного алгоритма обычно проста: найти минимальное значение функции стоимости и соответствующий ей вектор решения. Однако даже в случае однокритериальной

оптимизации нас вполне могут заинтересовать несколько разных метрических показателей результативности эволюционного алгоритма. Нас вполне может интересовать не только нахождение минимального значения функции стоимости, но и быстрое нахождение «хорошего» решения, которое не обязательно является лучшим. Нас также вполне может интересовать поиск ряда хороших решений в многообразных участках поискового пространства. Таким образом, даже в кажущейся простой однокритериальной оптимизационной задаче у нас может быть несколько метрических показателей результативности. Это осложнение возрастает вместе с многокритериальной оптимизацией. Некоторые потенциальные цели многокритериальных эволюционных алгоритмов вполне могут заключаться в следующем.

1. Максимизировать число особей, которые мы находим на определенном расстоянии от истинного множества Парето.
2. Минимизировать среднее расстояние между множеством Парето, аппроксимированным многокритериальным эволюционным алгоритмом и истинным множеством Парето.
3. Максимизировать многообразие особей, которые мы находим во множестве Парето, аппроксимированном многокритериальным эволюционным алгоритмом.
4. Минимизировать расстояние от кандидатного решения в пространстве целевой функции до идеальной точки, так называемой точки утопии[1].

Цели 1 и 2 касаются нахождения «лучшей» аппроксимации истинного множества Парето. Цель 3 касается поиска многообразного множества решений, с тем чтобы лицо, принимающее решения, располагало достаточными ресурсами для принятия информированного решения в отношении возможных компромиссов. В отличие от других целей, цель 4 заключается в нахождении решения, максимально приближенного к идеальному решению лица, принимающего решение, которого может и не существовать. Однако большинство современных многокритериальных эволюционных алгоритмов в первую очередь занимаются поиском лучшей аппроксимации истинного множества Парето.

Приведенные выше цели 1 и 2 основаны на допущении, что мы изначально знаем истинное множество Парето, поэтому эти критерии, возможно, будут полезными при тестировании многокритериальных

[1] В некоторых работах термины «идеальная точка» и «точка утопии» (или «утопическая точка») несколько друг от друга отличаются, но для целей настоящей главы мы рассматриваем их как синонимы.

эволюционных алгоритмов на хорошо понимаемых эталонах, но эти критерии бесполезны при выполнении многокритериального эволюционного алгоритма на реальной оптимизационной задаче. Однако если мы знаем истинное множество Парето P_s и многокритериальный эволюционный алгоритм дает нам приближенное множество Парето $\hat{P}_s$, то среднее расстояние M (P_s, $\hat{P}_s$) между ними может быть вычислено как

$$M_1(P_s, \hat{P}_s) = \frac{1}{\left\|\hat{P}_s\right\|} \sum_{x \in \hat{P}_s} \min_{x^* \in P_s^*} \left\| x^* - x \right\|, \tag{20.15}$$

где $\|\cdot\|$ – это любой указываемый пользователем метрический показатель расстояния.

Упомянутая выше цель 3 может быть измерена разными способами. Во-первых, мы можем измерить среднее расстояние каждой особи до ближайшего соседа в аппроксимированном множестве Парето. Во-вторых, мы можем измерить расстояние между двумя крайними особями в аппроксимированном множестве Парето. В-третьих, мы можем вычислить среднее число особей, которые дальше некоторого порога от каждого элемента в аппроксимированном множестве Парето [Zitzler и соавт., 2000]:

$$M_2(\hat{P}_s) = \frac{1}{\left\|\hat{P}_s\right\|} \sum_{x \in \hat{P}_s} \left| x' \in \hat{P}_s : \left\| x' - x \right\| > \sigma \right|, \tag{20.16}$$

где σ – задаваемый пользователем порог расстояния. В общем случае M_2 увеличивается по мере увеличения числа элементов в $\hat{P}_s$, а также по мере увеличения многообразия элементов в $\hat{P}_s$. В публикации [Khare и соавт., 2003] обсуждаются некоторые дополнительные метрические показатели многообразия для многокритериальных эволюционных алгоритмов.

Упомянутая выше цель 4 называется оптимизацией целевого вектора [Wienke и соавт., 1992], достижением цели [Wilson и Macleod, 1993] или программированием цели. Она исходит из того, что пользователь думает о какой-то идеальной точке в пространстве целевой функции, и она требует определения термина «расстояние». Обычно мы используем евклидово расстояние D_2, так называемую норму L2 (евклидову норму) между вектором f целевой функции и идеальной точкой f^*. Расстояние между f и f^* определяется следующим образом:

$$D_2^2(f^*(x), f(x)) = \|f^*(x) - f(x)\|_2^2 = \sum_{i=1}^{k} (f_i^*(x) - f_i(x))^2. \tag{20.17}$$

Кроме того, мы можем также использовать другие меры расстояния, такие как норма L2 взвешенная, норма L1 (манхеттенское расстояние) или норма L-бесконечность.

Вспомним пример 20.2. Пользователь вполне может подумать, что было бы идеально достичь $f_1(x) = 0$ и $f_2(x) = 0$. Получив фронт Парето на рис. 20.3, мы видим, что точка, ближайшая к идеальной точке, в терминах евклидова расстояния равна $x_1 = x_2 = 1$, что дает $f_1(x) = 2$ и $f_2(x) = 2$. С другой стороны, если идеальным решением пользователя является $f_1(x) = 2$ и $f_2(x) = 0$, то ближайшее, что мы можем получить, – это $x_1 = x_2 = 1.20$, что дает $f_1(x) = 2.87$ и $f_2(x) = 1.29$. Используя цель 4, квантификация результативности многокритериального эволюционного алгоритма встраивает предпочтения пользователя в окончательное решение многокритериальной оптимизационной задачи.

Обратите внимание, что мы можем преследовать цели 1–3 с точки зрения либо фронта Парето, либо множества Парето. Например, в цели 1 вместо максимизации числа особей, находящихся на определенном расстоянии от истинного множества Парето, мы могли бы максимизировать число особей, векторы функций которых находятся на определенном расстоянии от истинного фронта Парето. Таким образом, мы видим, что для многокритериального эволюционного алгоритма существует много возможных критериев результативности. Другими словами, оптимизация результативности многокритериального эволюционного алгоритма сама по себе является многокритериальной оптимизационной задачей. Это представляется уместным, но усложняет оценивание многокритериального эволюционного алгоритма.

■ Пример 20.3

На рис. 20.4 показана работа трех разных эволюционных алгоритмов на многокритериальной оптимизационной задаче. На рис. 20.4(a) показано решение, которое является довольно многообразным и достаточно близким к истинному фронту Парето. На рис. 20.4(b) показано решение, которое многообразнее, чем на рис. 20.4(a), в том смысле, что расстояние между крайними решениями дальше, но рис. 20.4(b) включает только три решения, а рис. 20.4(a) включает четыре решения. На рис. 20.4(c) показаны решения, которые ближе к истинному фронту Парето, чем на рис. 20.4(a) или (b), но многообразие не такое хорошее. Какое из трех решений является «лучшим»? Это зависит от приоритетов лица, принимающего решение.

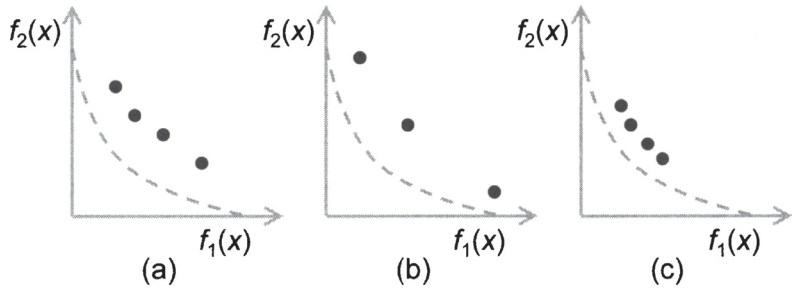


Рис. 20.4. Пример 20.4: на этом рисунке показаны три потенциальных эволюционно-алгоритмических решения двукритериальной оптимизационной задачи. Истинный фронт Парето – это пунктирная линия, а окружности – аппроксимации, найденные каждым эволюционным алгоритмом. Какое решение является «лучшим»? Это зависит от приоритетов лица, принимающего решения, относительно многообразия решений и близости к истинному фронту Парето

■

20.2.1. *Гиперобъем*

Еще одним метрическим показателем, который часто используется для оценивания качества фронта Парето, является его гиперобъем (hypervolume). Предположим, что многокритериальный эволюционный алгоритм нашел M точек в приближенном фронте Парето $\hat{P}_f = \{f(x_j)\}$ для $j \in [1, M]$, где $f(x_j)$ – это k-мерная функция. Гиперобъем может быть вычислен как

$$S(\hat{P}_f) = \sum_{j=1}^{M} \prod_{i=1}^{k} f_i(x_i). \tag{20.17}$$

При наличии двух многокритериальных эволюционных алгоритмов, которые вычисляют две аппроксимации фронта Парето поставленной многокритериальной оптимизационной задачи, мы можем использовать меру гиперобъема, для того чтобы квантифицировать то, насколько хороши две аппроксимации относительно друг друга. Для задачи минимизации меньший гиперобъем указывает на более оптимальную аппроксимацию фронта Парето.

■ Пример 20.4

Предположим, что у нас есть два многокритериальных эволюционных алгоритма, каждый из которых предназначен для аппроксима-

ции фронта Парето двухкритериальной минимизационной задачи. На рис. 20.5 показаны их аппроксимации фронта Парето. Рисунок 20.5(a) имеет следующие ниже точки аппроксимации фронта Парето:

$$\hat{P}_f(1) = \{[f_1(x_j), f_2(x_j)]\} = \{[1, 5], [2, 3], [5, 1]\}, \tag{20.19}$$

что дает гиперобъем 5 + 6 + 5 = 16. Рисунок 20.5(b) имеет следующие ниже точки аппроксимации фронта Парето:

$$\hat{P}_f(2) = \{[f_1(x_j), f_2(x_j)]\} = \{[1, 4], [3, 3], [4, 1]\}, \tag{20.20}$$

что дает гиперобъем 4 + 9 + 4 = 17. Согласно мере гиперобъема уравнения (20.18), рис. 20.5(a) дает немного лучший $\hat{P}_f$, чем на рис. 20.5(b).

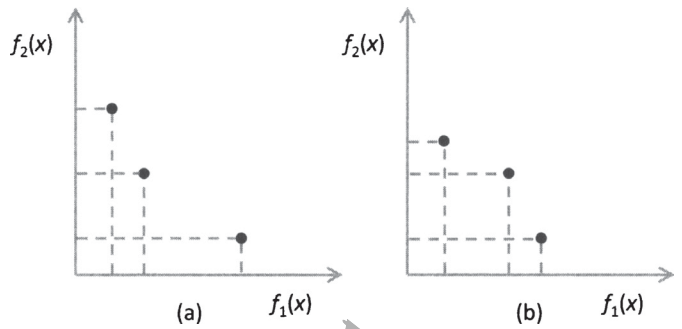


Рис. 20.5. Пример 20.4: на этом рисунке показаны две аппроксимации фронта Парето для двухкритериальной оптимизационной задачи. Мера гиперобъема используется для количественной оценки качества аппроксимаций. Аппроксимация слева имеет гиперобъем 16, аппроксимация справа – гиперобъем 17

■

Гиперобъем нельзя использовать слепо как показатель качества фронта Парето. Уравнение (20.18) показывает, что аппроксмация пустого фронта Парето ($M = 0$) дает наименьшее возможное значение S. Следовательно, более точной мерой, возможно, будет нормализованный гиперобъем $S_n(\hat{P}_f) = S(\hat{P}_f)/M$. Однако даже эта величина не может быть хорошим метрическим показателем для аппроксимации фронта Парето. Мы можем это увидеть, рассматривая возможность того, что некая аппроксимация фронта Парето $\hat{P}_f(1)$ имеет нормализованный гиперобъем $S_n(\hat{P}_f(1))$. Теперь предположим, что мы добавляем одну новую точку в $\hat{P}_f(1)$ с целью получить $\hat{P}_f(2)$. В результате вполне можно получить $S_n(\hat{P}_f(2)) > S_n(\hat{P}_f(1))$, даже если единственное различие между $\hat{P}_f(1)$ и $\hat{P}_f(2)$ состоит в том, что $\hat{P}_f(2)$ имеет дополнительную точку. $\hat{P}_f(2)$ явно лучше $\hat{P}_f(1)$, но $S_n(\hat{P}_f(2))$ больше $S_n(\hat{P}_f(1))$, что противоречит здравому смыслу.

Это приводит нас к изменению меры гиперобъема путем вычисления его *не* относительно начала пространства целевой функции, а относительно опорной точки, лежащей за пределами фронта Парето. Предположим, что мы хотим сравнить Q аппроксимаций фронта Парето $\hat{P}_f(q)$ для $q \in [1, Q]$. Вычисляем опорную точку вектор $r = [r_1, ..., r_k]$, где

$$r_i > \max_q \left[\max_{x \in \hat{P}_S(q)} f_i(x) \right], \tag{20.21}$$

а затем вычисляем гиперобъемы S относительно опорной точки:

$$S'(\hat{P}_f(q)) = \sum_{j=1}^{M(q)} \prod_{i=1}^{k} (r_i - f_i(x_j(q))), \tag{20.22}$$

где $M(q)$ – это число точек в q-й аппроксимации фронта Парето и $x_j(q)$ – j-я точка в q-й аппроксимации множества Парето. Более крупный гиперобъем S' на основе опорной точки указывает на то, что для минимизационной задачи у нас есть более качественный фронт Парето. Мы можем использовать либо нормализованный гиперобъем на основе опорной точки

$$S'_n(\hat{P}_f(q)) = S'(\hat{P}_f(q))/M(q), \tag{20.23}$$

либо общую меру гиперобъема $S'(\hat{P}_f(q))$ на основе опорной точки, если мы хотим, чтобы данный метрический показатель учитывал число точек Парето. Рисунок 20.6 иллюстрирует гиперобъем на основе опорной точки в двух размерностях.

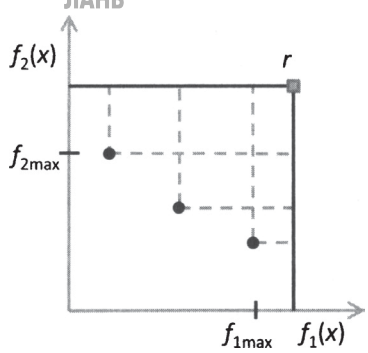


Рис. 20.6. Вычисление гиперобъема с опорной точкой $S'(\hat{P}_f(q))$ уравнения (20.22). Опорная точка r – это произвольная опорная точка, i-я составляющая которой больше, чем у каждой из точек аппроксимации фронта Парето. Большее $S'(\hat{P}_f(q))$ указывает на более качественную аппроксимацию задачи минимизации фронта Парето

Многие обсуждения многокритериальных эволюционных алгоритмов в литературе конвертируют многокритериальные оптимизационные задачи в максимизационные задачи, иными словами, более крупные гиперобъемы в уравнении (20.18) более желательны. Это согласуется с тем, что более желательным является наличие большего числа точек в $\hat{P}_f$ (большего M). Уравнения (20.18) и (20.22) дают основное представление о расчете гиперобъема, однако литература содержит несколько других методов, определений и алгоритмов расчета гиперобъема [Auger и соавт., 2012], [Bringmann и Friedrich, 2010], [Zitzler и соавт., 2003]. В большинстве публикаций для расчета гиперобъема используется *объединение M* гиперобъемов уравнений (20.18) и (20.22). Например, если вычислить гиперобъем объединения гиперобъемов на рис. 20.5, то обе аппроксимации $\hat{P}_f$ будут иметь гиперобъем 11. Другие способы вычисления гиперобъема могут привести к разным выводам об относительных достоинствах двух аппроксимаций $\hat{P}_f$. Однако уравнения (20.18) и (20.22) проще реализовать, чем вычисление объема объединения гиперобъемов. Существует явно высокая корреляция между суммой M гиперобъемов и гиперобъемом объединения M гиперобъемов, хотя данная корреляция не является идеально линейной (см. задачу 20.7).

20.2.2. *Относительный охват*

Еще один способ сравнения аппроксимаций фронта Парето заключается в вычислении среднего числа особей в одной аппроксимации, над которыми слабо доминирует, по крайней мере, одна особь в другой аппроксимации [Zitzler and Thiele, 1999]. Предположим, что у нас есть две аппроксимации $\hat{P}_f$, обозначаемые как $\hat{P}_f(1)$ и $\hat{P}_f(2)$. Мы определяем охват $\hat{P}_f(1)$ относительно $\hat{P}_f(2)$ как среднее число особей в $\hat{P}_f(2)$, над которыми слабо доминирует, по крайней мере, одна особь в $\hat{P}_f(1)$:

$$C(\hat{P}_f(1), \hat{P}_f(2)) = \frac{|\alpha_2 \in \hat{P}_f(2)| \text{ такой, что } \exists[\alpha_1 \in \hat{P}_f(1) \text{ такой, что } \alpha_1 \succcurlyeq \alpha_1}{|\hat{P}_f(2)|}. \tag{20.24}$$

Обратите внимание, что $C(\hat{P}_f(1), \hat{P}_f(2)) \in [0, 1]$. Если $C(\hat{P}_f(1), \hat{P}_f(2)) = 0$, то для каждой особи $\alpha_2 \in \hat{P}_f(2)$ нет ни одной особи в $\hat{P}_f(1)$, которая слабо доминирует над α_2. Если $C(\hat{P}_f(1), \hat{P}_f(2)) = 1$, то для каждой особи $\alpha_2 \in \hat{P}_f(2)$ найдется хотя бы одна особь в $\hat{P}_f(1)$, которая слабо доминирует над α_2. Хотя мы не можем использовать уравнение охвата, чтобы получить

абсолютную меру качества аппроксимации фронта Парето, оно может быть полезным при сравнении нескольких аппроксимаций.

20.3. Непаретоориентированные эволюционные алгоритмы

В этом разделе рассматривается несколько многокритериальных эволюционных алгоритмов, в которых явно не используется идея доминирования по Парето. В разделе 20.3.1 рассматриваются методы агрегирования, в разделе 20.3.2 – генетический алгоритм с векторным оцениванием (VEGA), в разделе 20.3.3 обсуждаются подходы на основе лексикографического упорядочения, в разделе 20.3.4 – метод ϵ-ограничения, раздел 20.3.5 посвящен гендерным подходам.

20.3.1. *Методы агрегирования*

Методы агрегирования объединяют вектор целевой функции многокритериальной оптимизационной задачи в скалярную целевую функцию. Например, мы можем конвертировать k-критериальную оптимизационную задачу уравнения (20.1) в задачу

$$\min_x f(x) \Rightarrow \min_x \sum_{i=1}^{k} w_i f_i(x), \quad \text{где } \sum_{i=1}^{k} w_i = 1. \tag{20.25}$$

$\{w_i\}$ – это множество положительных весовых параметров, элементы которых в сумме составляют 1. Уравнение (20.25) называется подходом на основе взвешенной суммы, но могут использоваться и другие методы агрегирования. Например, мы можем объединить цели в произведении:

$$\min_x f(x) \Rightarrow \min_x \prod_{i=1}^{k} f_i(x). \tag{20.26}$$

Если мы используем метод агрегации на основе произведения, то мы должны убедиться, что все целевые критерии $f_i(x) > 0$ для всех x. Но какой бы метод агрегирования мы не использовали, главное – конвертировать многокритериальную оптимизационную задачу в однокритериальную оптимизационную задачу.

■ Пример 20.5

Рассмотрим многокритериальную оптимизационную задачу из примера 20.2. Если мы применим уравнение (20.25) для ее конвертирования в однокритериальную задачу, то получим

$$\min_x \{w_1 f_1(x) + w_2 f_2(x)\} = \min_x \{w_1(x_1^2 + x_2^2) + (1 - w_1)[(x_1 - 2)^2 + (x_2 - 2^2)]\}. \tag{20.27}$$

Мы можем минимизировать это уравнение, взяв его частную производную по x_1 и x_2, получив

$$\frac{\partial[w_1 f_1(x) + w_2 f_2(x)]}{\partial x_1} = 2x_1 + 4(w_1 - 1)$$
$$\frac{\partial[w_1 f_1(x) + w_2 f_2(x)]}{\partial x_2} = 2x_2 + 4(w_1 - 1) \tag{20.28}$$

Приравняв эти два уравнения к нулю и решив для x, в результате получим множество Парето

$$x_1^* = x_2^* = 2(1 - w_1). \tag{20.29}$$

Подстановка множества Парето в уравнения для $f_1(x)$ и $f_2(x)$ дает фронт Парето

$$f_1(x^*) = 8(1 - w_1)^2$$
$$f_2(x^*) = w_1^2 \tag{20.30}$$

Построение графиков уравнений (20.29) и (20.30) при варьировании w_1 от 0 до 1 дает графики, идентичные рис. 20.3.

■

Пример 20.5 показывает, что метод агрегирования может найти множество Парето и фронт Парето, по крайней мере для некоторых многокритериальных оптимизационных задач. Фактически решение уравнения (20.25) для любого множества весов приводит к оптимальной по Парето точке. Однако, как показано в следующем далее примере, если фронт Парето вогнут, то метод агрегирования не сможет найти полное множество Парето и фронт Парето.

■ Пример 20.6

Рассмотрим задачу

$$\min_x f(x) = \min_x [x^2, \cos^3 x], \tag{20.31}$$

где $x \in [0, 4]$. Это одномерная многокритериальная оптимизационная задача с двумя целевыми критериями. Мы можем агрегировать два целевых критерия в скалярный целевой критерий

$$\min f(x) = \min [wx^2 + (1 - w)\cos^3 x], \quad (20.32)$$

где $w \in [0, 1]$, и мы можем решить уравнения (20.31) и (20.32) с помощью исчерпывающего поиска. Однако решения этих двух уравнений не совпадают. На рис. 20.7 показаны решения двух задач. Мы видим, что истинный фронт Парето вогнут. Метод агрегирования правильно дает выпуклую часть фронта Парето, но зато он неправильно дает вогнутую часть.

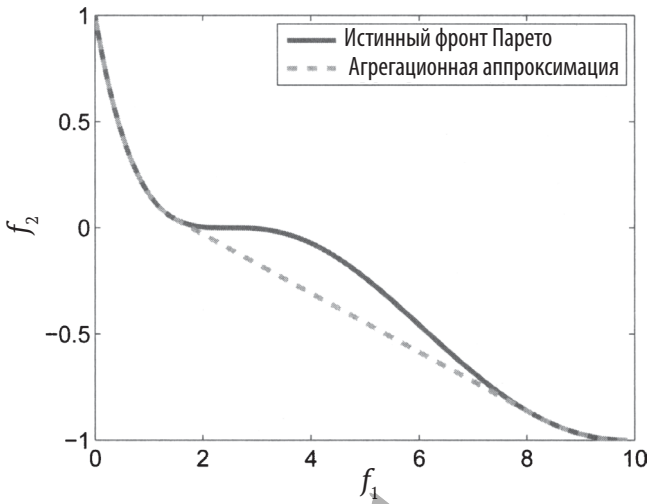


Рис. 20.7. Пример 20.6: метод агрегирования уравнения (20.32) правильно дает выпуклую часть фронта Парето, но не дает правильно вогнутую часть

■

Достижение цели для вогнутых фронтов Парето

Дело не просто в том, что агрегирование не может найти фронт Парето для отдельной задачи примера 20.6. На самом деле с помощью метода агрегирования невозможно найти никакого вогнутого фронта Парето [Fleming и соавт., 2005]. Однако мы можем найти вогнутые фронты Парето, используя расширение метода достижения цели, упомянутого ранее как Цель 4 в разделе 20.2. К достижению цели иногда подходят, решая следующую задачу:

$$\min \alpha \text{ такой, что } f_i(x) \le f_i^* + w_i \alpha \text{ для всех } i \in [1, k] \text{ и для некоторых } x, \quad (20.33)$$

где f_i^* – это идеальное значение i-го целевого критерия, $\{w_i\}$ – множество положительных весов, указывающих на относительную важность каждой цели. Уравнение (20.33) допускает возможность существования

решений, которые лучше, чем идеальное решение пользователя. Если оптимальное $\alpha > 0$, то идеальная точка недостижима, но решение уравнения (20.33) находит вектор решения, максимально близкий к идеальной точке. Если оптимальное $\alpha < 0$, то мы можем найти решение, которое лучше, чем идеальная точка для каждого целевого критерия. Можно показать [Chen и Liu, 1994], [Coello Coello, 1999], что решение уравнения (20.33) для всех весовых комбинаций $\{w_i\}$ – таких, что $\sum_i w_i = 1$ – дает множество Парето для исходной многокритериальной оптимизационной задачи, даже если фронт Парето вогнут.

Обратите внимание, что уравнение (20.33) является оптимизационной задачей с единственным целевым критерием, k ограничениями и $n + 1$ независимыми переменными (скаляр α и исходный вектор x решения). Следовательно, мы можем использовать алгоритмы ограниченной оптимизации (см. главу 19) для решения уравнения (20.33).

20.3.2. *Генетический алгоритм с векторным оцениванием (VEGA)*

Генетический алгоритм с векторным оцениванием (vector evaluated genetic algorithm, VEGA) – это первоначальный многокритериальный эволюционный алгоритм [Schaffer, 1985]. Данный алгоритм работает, выполняя отбор на популяции, используя одну целевую функцию за раз. В результате получается множество субпопуляций, по одному множеству для каждой целевой функции. Затем мы отбираем особей из субпопуляций, получая родителей для следующего поколения, и комбинируем родителей, используя стандартные методы эволюционно-алгоритмической рекомбинации, получая детей. Рисунок 20.8 дает схематичное описание генетического алгоритма с векторным оцениванием.

Инициализировать популяцию кандидатных решений $P = \{x_j\}$ для $j \in [1, N]$.
$M \leftarrow \lceil N/k \rceil$
While not (критерий останова)
  Вычислить стоимость $f_i(x_j)$ для каждого целевого критерия i
    и для каждой особи $x_j \in P$.
  For $i = 1$ **to** k
    $P_i \leftarrow M$ особей, вероятностно отобранных из P, используя $f_i(\cdot)$
  Next i
  $P \leftarrow N$ особей, отобранных из $\{P_1, ..., P_k\}$
  $C \leftarrow N$ детей, созданных в результате рекомбинации особей в P
  Вероятностно мутировать детей в C
  $P \leftarrow C$

Next поколение

$P_s \leftarrow$ недоминированные элементы популяции P

Рис. 20.8. Схематичное описание генетического алгоритма с векторным оцениванием (VEGA) для решения оптимизационной задачи с k целевыми критериями

На рис. 20.8 показано, что генетический алгоритм с векторным оцениванием начинается с популяции из N особей, которых мы обычно генерируем случайно. В каждом поколении мы вычисляем значение всех k значений функции стоимости для всех N особей. Затем мы используем любую желаемую схему отбора (см. раздел 8.7) для отбора M особей, где $M = |N/k|$ – это наименьшее целое число, большее или равное N/k. Мы выполняем этот отбор вероятностно, сначала используя $f_1(\cdot)$ для создания популяции P_1, затем используя $f_2(\cdot)$ для создания популяции P_2 и т. д. После того как мы создали P_i субпопуляций, мы комбинируем их и получаем родительскую популяцию P. Затем мы рекомбинируем особей в P и получаем множество детей C. Мы можем выполнить рекомбинацию, используя любой из эволюционных алгоритмов, обсуждаемых в этой книге (генетические алгоритмы, дифференциальная эволюция DE, биогеографическая оптимизация BBO и т. д.). Мы видим, что название данного алгоритма (VEGA) является чем-то вроде анахронизма; в зависимости от метода рекомбинации, который мы используем, мы могли бы назвать его VEDE, либо VEBBO, либо любой другой аббревиатурой, подходящей для типа рекомбинации, который мы используем. Эта особенность также относится ко многим другим популярным многокритериальным эволюционным алгоритмам, которые мы обсуждаем в этой главе (NSGA, MOGA и т. д.).

Как мы видели в однокритериальных эволюционным алгоритмах, элитарность может значительно улучшать результативность оптимизации, и эта особенность также относится к многокритериальным эволюционным алгоритмам. Существует несколько способов реализации элитарности, показанных на рис. 20.8. Например, в каждом поколении мы можем находить лучших особей применительно к каждой целевой функции и обеспечивать, чтобы они сохранялись из поколения в поколение. Либо мы можем находить недоминированных особей в каждом поколении и обеспечивать, чтобы по крайней мере некоторые из них сохранились из поколения в поколение. Хотя генетический алгоритм с векторным оцениванием обычно не определяется как элитарный алгоритм, добавление элитарности в рис. 20.8 не изменит его сущности и, вероятно, улучшит его результативность.

Алгоритм, похожий на алгоритм VEGA, иногда называется генетическим алгоритмом Хаджела-Лин (Hajela-Lin genetic algorithm, HLGA). В нем используется подход на основе взвешенной суммы. Алгоритм HLGA включает весовой вектор уравнения (20.25) как часть переменной каждой особи, которая называется переменной решения [Hajela и Lin, 1997]. В данном подходе используется однокритериальная оптимизация на основе взвешенной суммы уравнения (20.25). Для достижения многообразия весов в алгоритме HLGA используется обмен информацией о приспособленности (см. раздел 8.6.3.1).

20.3.3. *Лексикографическое упорядочение*

Лексикографический упорядочение (lexicographic ordering) похоже на алгоритм VEGA, но позволяет пользователю ранжировать целевые критерии по очередности [Fourman, 1985]. Мы проводим турнирный отбор, сравнивая особей по ранжированным по приоритету целевым критериям. Вместо того чтобы использовать для каждого турнира ранжированные по приоритету целевые критерии, мы также можем использовать для них случайно отбранные целевые критерии [Kursawe, 1991]. Лексикографическое упорядочение аналогично подходу на основе поведенческой памяти для ограниченной оптимизации (см. раздел 19.2.11) в своей последовательной обработке целевых критериев.

На рис. 20.9 схематично представлен метод лексикографического упорядочения. В оригинальном методе лексикографического упорядочения внешний цикл на рис. 20.9 выполняется для $i \in [1, k]$ в приоритетном порядке целевых функций. В рандомизированном варианте внешний цикл выполняется до тех пор, пока не будет удовлетворен определяемый пользователем критерий останова, а индекс i варьируется случайно в каждом поколении. Как и в случае с алгоритмом VEGA, на рис. 20.9 мы можем реализовать ряд вариаций, включая элитарность и встраивание любого из эволюционных алгоритмов, описанных в этой книге или в других местах.

20.3.4. *Метод ϵ-ограничений*

Метод ϵ-ограничений (ϵ-constraint method) [Ritzel и соавт., 1995] минимизирует одну целевую функцию за раз, ограничивая значения других целевых функций ниже заданного порога. Сначала мы находим минимальное значение каждой целевой функции $f_i(x)$ для $i \in [1, k]$, минимизируя ее с помощью однокритериального эволюционного алго-

ритма. В результате получаем нижнюю границу для ограничений целевых функций ϵ_i:

$$\epsilon_i > \min_i f_i(x) \quad \text{для} \quad i \in [1, k]. \tag{20.34}$$

P ← случайно сгенерированная популяция
While not (критерий останова)
    Задать индекс *i* целевой функции
    Инициализировать эволюционный алгоритм популяцией *P*
    Применить эволюционный алгоритм для минимизации $f_i(x)$,
        обозначив окончательную популяцию как *P*
Next эволюционный алгоритм

Рис. 20.9. Лексикографическое упорядочение для оптимизационной задачи с *k* целевыми критериями. Целевая функция $f_i(x)$ каждую итерацию может зависеть от установленной пользователем приоритетности либо может быть отобрана случайно. Кроме того, целевая функция $f_i(x)$ может меняться из поколения в поколение в течение каждого эволюционного алгоритма

Затем мы минимизируем первый целевой критерий, ограничивая другие целевые критерии так, чтобы они были меньше некоторого порога:

$$\min_x f_1(x) \text{ такой, что } f_i(x) < \epsilon_i \quad \text{для} \quad i \in [1, k], i \neq 1. \tag{20.35}$$

Используя окончательную эволюционно-алгоритмическую популяцию, полученную в результате уравнения (20.35), в качестве исходной совокупности следующего эволюционного алгоритма, мы минимизируем следующий целевой критерий:

$$\min_x f_2(x) \text{ такой, что } f_i(x) < \epsilon_i \quad \text{для} \quad i \in [1, k], i \neq 2. \tag{20.36}$$

Мы повторяем этот процесс для всех к цели. Затем уменьшаем значения ϵ и повторяем последовательный процесс минимизации. Этот последовательный подход аналогичен лексикографическому упорядочению (см. раздел 20.3.3) и подходу на основе поведенческой памяти для ограниченной оптимизации (см. раздел 19.2.11). На рис. 20.10 приводится схематичное описание многокритериального эволюционного алгоритма с ϵ-ограничениями. Наиболее сложная часть реализации данного алгоритма – принять решение о том, как именно «назначить ϵ_i некое число больше $f_i^*(x)$» и как «уменьшить ϵ, для $i \in [1, k]$» на рис. 20.10. Как и в случае с другими многокритериальными эволюционными алго-

ритмами, на рис. 20.10 мы можем реализовать ряд вариаций, включая элитарность и встраивание любого из эволюционных алгоритмов, описанных в этой книге или в других местах.

For each целевая функция $f_i(x)$, где $i \in [1, k]$
  Применить эволюционный алгоритм, чтобы найти $f_i^*(x) = \min_x f_i(x)$.
  Назначить ϵ_i некое число больше $f_i^*(x)$.
Next целевая функция
Инициализировать эволюционно-алгоритмическую популяцию P для задачи многокритериальной оптимизации.
While not (критерий останова)
  For each целевая функция $f_i(x)$, где $i \in [1, k]$
    Инициализировать эволюционно-алгоритмическую популяцию результатом предыдущего эволюционного алгоритма.
    Применить эволюционный алгоритм для минимизации $f_i(x)$ так, чтобы $f_r(x) < \epsilon_r$ для $r \in [1, k]$, $r \neq i$
    $P \leftarrow$ окончательная эволюционно-алгоритмическая популяция
  Next целевая функция
  Уменьшить ϵ_i для $i \in [1, k]$
Next итерация

Рис. 20.10. Схематичное описание метода ϵ-ограничений для решения оптимизационной задачи с k целевыми критериями

20.3.5. *Гендерные подходы*

Гендерные подходы (gender-based approach) назначают гендерную принадлежность каждой особи на основе целевой функции, с помощью которой они должны будут оцениваться [Allenson, 1992], [Lis и Eiben, 1997]. В гендерных методах также используется вторичная популяция, именуемая архивом. Архив представляет собой коллекцию недоминированных особей, и, аналогично элитарности, он предотвращает выпадение хороших особей. Архивы также используются во многих других многокритериальных эволюционных алгоритмах, в том числе в некоторых, которые мы обсудим ниже.

В гендерном подходе для задачи k-критериальной оптимизации у нас есть k разных полов, каждый из которых соответствует разному целевому критерию. Мы создаем исходную популяцию с равным числом особей для каждого пола. Затем отбираем особей из каждого пола i на основе $f_i(x)$. Потом используем отобранных особей для рекомбинации и получаем ребенка. Мы можем присвоить ребенку индивидуальный

пол на основании целевого критерия, для которого он работает лучше всего. Можем выполнить рекомбинацию, используя любой из методов, рассмотренных в этой книге. В случае стандартного одноточечного генетико-алгоритмического скрещивания для рекомбинации мы будем использовать только двух особей. В случае многородительского скрещивания (см. раздел 8.8.5) будем использовать одну или несколько особей от каждого пола. В конце каждого поколения мы обычно сравниваем популяцию с архивом и сохраняем в архиве недоминированных особей, удаляя оттуда доминированных особей. На рис. 20.11 представлено схематичное описание гендерного метода для многокритериальной оптимизации.

N_g = желаемый размер популяции для каждого пола.
Инициализировать k эволюционно-алгоритмических популяций P_i,
где $|P_i| = N_g$ для $i \in [1, k]$.
Размер эволюционно-алгоритмической популяции $N \leftarrow kN_g$
While not (критерий останова)
For $m = 1$ **to** N
For $i = 1$ **to** k
Применить $f_i(x)$ для вероятностного отбора одного родителя из P_i
Next i
Использовать k отобранных особей для создания детской особи c_m
Next m
Назначить гендеры детям $\{c_m\}$.
Случайно мутировать детей $\{c_m\}$.
Сохранить недоминированных детей в архиве.
Удалить доминированных особей из архива.
Поменять эволюционно-алгоритмические популяции $\{P_i\}$ детьми соответствующего пола.
Next поколение

Рис. 20.11. Схематичное описание гендерного алгоритма решения оптимизационной задачи с k целевыми критериями

Мы видим ряд возможностей для модификации алгоритма на рис. 20.11. Например, в зависимости от размерности x мы можем отобрать для рекомбинации более одного родителя из каждого пола. Кроме того, строка «Назначить гендеры детям» упускает много деталей, которые необходимо определить. Следует ли нам ограничивать каждого ребенка одним полом? Нужно ли нам обеспечивать поддержку равного количества каждого пола в популяции? Подход, описанный на рис. 20.11, создает N детей, но мы, возможно, захотим создать больше,

чем это число, для того чтобы получить детскую популяцию с высоко приспособленными детьми для каждого пола.

Инструкция «Сохранить недоминированных детей в архиве» на рис. 20.11 также упускает много деталей. Должны ли мы дать архиву расти без ограничений? Следует ли нам установить верхний предел размера архива? Следует ли нам добавлять особей в архив, если они недоминированы в отношении текущей популяции либо только если они недоминированы в отношении архива? Мы обсудим в общих чертах некоторые из этих связанных с архивом вопросов в разделе 20.4.

20.4. Паретоориентированные эволюционные алгоритмы

Подходы многокритериальных эволюционных алгоритмов предыдущего раздела пытаются найти многообразное оптимальное по Парето множество решений для многокритериальной оптимизационной задачи. Однако ни в одном из них прямо не используется понятие оптимальности по Парето для вычисления относительного доминирования между особями или группами особей (за исключением того, когда особи добавляются в архив на рис. 20.11). В следующих ниже разделах рассматриваются подходы, в которых непосредственно используется доминирование по Парето.

- В разделе 20.4.1 рассматривается простой эволюционный многокритериальный оптимизатор (SEMO), а также диверсифицирующий эволюционный многокритериальный оптимизатор (DEMO).
- В разделе 20.4.2 анализируется ϵ-ориентированный многокритериальный эволюционный алгоритм (ϵ-MOEA).
- В разделе 20.4.3 рассматриваются генетический алгоритм с сортировкой по уровню недоминирования (NSGA) и его обновленная версия (NSGA-II).
- В разделе 20.4.4 разбирается многокритериальный генетический алгоритм (MOGA).
- В разделе 20.4.5 рассматривается генетический алгоритм с нишированием по Парето (NPGA).
- В разделе 20.4.6 анализируются эволюционный алгоритм с использованием силы Парето (SPEA) и его обновленная версия (SPEA2).
- В разделе 20.4.7 рассматривается эволюционная стратегия с архивированием по Парето (PAES).

20.4.1. *Эволюционные многокритериальные оптимизаторы*

В этом разделе рассматриваются два эволюционных многокритериальных оптимизатора: простой эволюционный многокритериальный оптимизатор (simple evolutionary multi-objective optimizer, SEMO) и диверсифицирующий эволюционный многокритериальный оптимизатор (diversity evolutionary multiobjective optimizer, DEMO). Как будет видно в данном разделе, эти алгоритмы мотивированы основными идеями эволюционного программирования (EP) и эволюционных стратегий (ES).

Простой эволюционный многокритериальный оптимизатор (SEMO)

Простой эволюционный многокритериальный оптимизатор (SEMO) был первоначально предложен для бинарной оптимизации [Laumanns и соавт., 2003], однако он легко расширяется на непрерывную оптимизацию. В алгоритме SEMO мы начинаем со случайно сгенерированной популяции особей. Исходный алгоритм SEMO начинается с популяции размером в одну особь. Популяция растет по мере того, как алгоритм находит все больше и больше недоминированных решений. В каждом поколении мы мутируем одну случайно отобранную особь из популяции и создаем ребенка. Мы добавляем ребенка в популяцию, в случае если ребенок не доминирован популяцией, и мы также удаляем любых доминированных особей из популяции. Рисунок 20.12 иллюстрирует алгоритм SEMO. «Случайная мутация» на рис. 20.12 может быть любым методом мутации, обсуждавшимся в главе 5, главе 6 либо разделе 8.9.

Алгоритм SEMO обеспечивает полезную отправную точку для многокритериальной оптимизации. Мы можем модифицировать алгоритм SEMO, опираясь на идеи из других многокритериальных эволюционных алгоритмов. Например, вместо того чтобы в качестве родителя y использовать «случайно отобранных особей из P», мы можем использовать отбор пропорциональной приспособленности. Кроме того, можем генерировать y из нескольких родителей либо периодически подрезать популяцию, как в алгоритме SPEA2 (см. раздел 20.4.6).

Инициализировать популяцию кандидатных решений $P = \{x_j\}$.
Вычислить стоимость $f_i(x_j)$ для каждого целевого критерия $i \in [1, k]$ и
    для каждой особи $x_j \in P$.
While not (критерий останова)
  $y \leftarrow$ мутация случайно отобранных особей из P

If над *y* не доминирует ни одна особь в *P* **then**
$P \leftarrow \{P, y\}$
Удалить всех особей из *P*, над которыми доминирует *y*.
End if
Next поколение

Рис. 20.12. Схематичное описание простого эволюционного многокритериального оптимизатора (SEMO)

Диверсифицирующий эволюционный многокритериальный оптимизатор (DEMO)

Одной из проблем алгоритма SEMO является неограниченный рост его популяции. Мы можем справиться с этой проблемой, используя в инструкции проверки на включение *y* в популяцию *P* ϵ-доминирование вместо доминирования. В результате этого получим диверсифицирующий эволюционный многокритериальный оптимизатор (DEMO). В алгоритме DEMO используется тот же самый алгоритм, что и на рис. 20.12, за исключением того, что в нем в качестве критерия для включения *y* в *P* используется ϵ-доминирование [Horoba и Neumann, 2010]. Благодаря этому повышается уровень для включения детской особи *y* в текущую популяцию; мы включаем особь *y* в популяцию только в том случае, если над ней не ϵ-доминирует ни одна другая особь в *P*. Поскольку ϵ-доминирование является более слабым типом доминирования, чем доминирование по Парето (см. раздел 20.1), критерий включения *y* в популяцию является более строгим в алгоритме DEMO, чем в алгоритме SEMO. Алгоритм DEMO, в сущности, делит целевое пространство на гипербоксы и не дает популяции содержать более одной особи на гипербокс. В алгоритме DEMO мы обычно используем аддитивное ϵ-доминирование.

На рис. 20.13 показана концепция алгоритма DEMO для двухкритериальной оптимизационной задачи. Мы не включим ребенка y_1 в популяцию, потому что над ним ϵ-доминирует особь, которая находится в том же гипербоксе. Однако мы включим y_2 в популяцию, потому что над ним не ϵ-доминирует ни один из текущих членов популяции. Обратите внимание на то, что рис. 20.13 не совсем корректен, потому что гипербоксы алгоритма DEMO определены относительно текущей популяции. Тем не менее рисунок дает концептуальную иллюстрацию использования ϵ-доминирования в алгоритме DEMO. Хотя на рис. 20.13 показано, что $\epsilon_1 = \epsilon_2$ (то есть боксы квадратные), пользователь может выбрать другое значение ϵ_i для каждого целевого индекса $i \in [1, k]$ на основе желаемой точности решения по каждому целевому критерию.

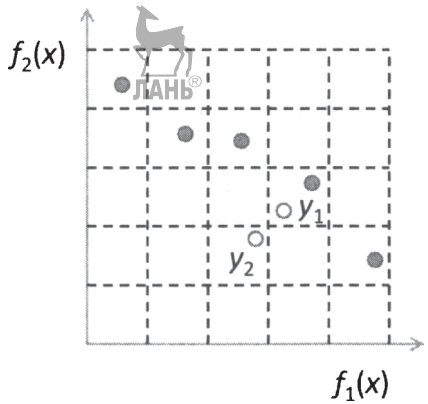


Рис. 20.13. ϵ-доминирование в диверсифицирующем эволюционном многокритериальном оптимизаторе (DEMO). Текущая популяция обозначена сплошными кружками, дети – незаполненными кружками. Мы добавляем ребенка в популяцию только в том случае, если над ним не ϵ-доминирует ни один из текущей популяции. На этом рисунке мы не добавим в популяцию y_1, но добавим y_2

20.4.2. *ϵ-ориентированный многокритериальный эволюционный алгоритм (ϵ-MOEA)*

В ϵ-ориентированном многокритериальном эволюционном алгоритме (ϵ-MOEA) используется идея ϵ-доминирования аналогично ϵ-доминированию, которое используется в алгоритме DEMO, как описано выше [Deb и соавт., 2005]. Алгоритм ϵ-MOEA включает популяцию и архив. В каждом поколении мы отбираем одну особь из популяции и одну особь из архива и с целью получения ребенка используем некий метод рекомбинации.

Если этот ребенок доминирует над особью в популяции, то мы заменяем доминированную особь на ребенка. Если этот ребенок доминирует более чем над одной особью, то мы заменяем случайно отобранную особь.

Далее сравниваем ребенка с архивом. Это сравнение может привести к четырем ситуациям. (1) Если над ребенком доминирует какая-либо из архивированных особей, то ребенок в архив не помещается. (2) Если ребенок доминирует над любым числом архивированных особей, то ребенок добавляется в архив, и доминированные особи удаляются из архива.

Если ни одно из этих двух условий не выполняется, то мы вычисляем ϵ-бокс $B(x)$ ребенка x:

$$B_j(x) = \lfloor f_j(x)/\epsilon_j \rfloor \tag{20.37}$$

для $j \in [1, k]$, где k – это число целевых критериев, ϵ_j – желаемая разрешающая спсобность j-й цели и $\lfloor \cdot \rfloor$ возвращает наибольшее целое число, меньшее или равное его аргументу. Это подводит нас к третьей ситуации, которая может возникнуть в результате сравнения ребенка с архивом. (3) Если ребенок x находится в том же ϵ-боксе, что и архивированная особь α, то ребенок заменяет α, если ребенок ближе к началу пространства целевой функции:

$$x \text{ заменяет } \alpha, \text{ если } \left[B(x) = B(\alpha),\ \text{и}\ \sum_{j=1}^{k} f_j^2(x) < \sum_{j=1}^{k} f_j^2(\alpha)\right]. \tag{20.38}$$

Вышеуказанное условие предполагает нормализацию целевых функций так, чтобы величины значений каждой целевой функции были соизмеримы друг с другом.

Уравнение (20.38) также использует евклидову норму для измерения расстояния, хотя мы можем использовать любую другую норму. (4) Если ни одно из трех предыдущих условий не выполняется, то мы добавляем ребенка в архив, в результате увеличивая размер архива на одну особь.

Рисунок 20.14 иллюстрирует эти четыре возможности. Обратите внимание, что описанная здесь логика гарантирует, что в каждом ϵ-боксе архива находится не более одной особи. Хотя архив может расти из поколения в поколение, его размер ограничен этой логикой. На рис. 20.14(a) показан случай, когда над ребенком доминирует один или несколько архивированных особей; в этом случае ребенок в архив не добавляется. На рис. 20.14(b) показан случай, когда ребенок доминирует над одной или несколькими архивированными особями; в этом случае ребенок заменяет доминированных особей. Рисунок 20.14(c) иллюстрирует случай, когда ребенок и архив недоминированы по отношению друг к другу и ребенок находится в том же ϵ-боксе, что и одна из архивированных особей; в этом случае ребенок заменяет архивированную особь, которая находится в том же ϵ-боксе, если ребенок находится ближе к исходной точке пространства целевой функции. Наконец, рис. 20.14(d) иллюстрирует случай, когда ребенок и архив недоминированы по отношению друг к другу и ребенок не делит ϵ-бокс ни с одной из архивированных особей; в этом случае ребенок добавляется в архив, в результате увеличивая размер архива на единицу.

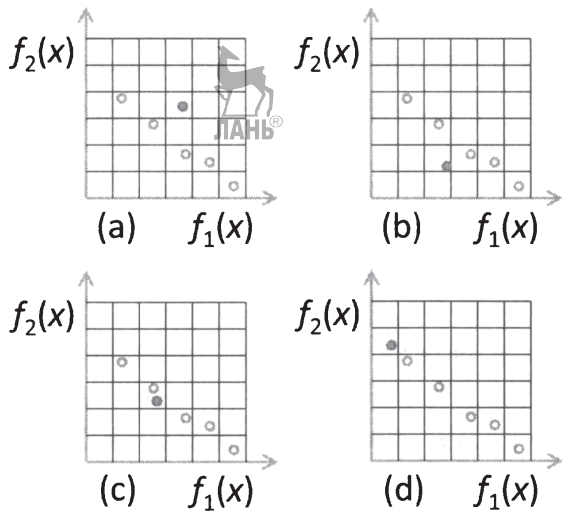


Рис. 20.14. Логика алгоритма ϵ-MOEA для добавления ребенка в архив. Пустые круги – это архивированные особи, сплошной круг – детская особь, и решетка в пространстве целевой функции определяет е-боксы. В случае (a) ребенок не добавляется в архив. В случае (b) ребенок заменяет особей, над которыми он доминирует, уменьшая на этом рисунке размер архива на единицу. В случае (c) ребенок заменяет особь, находящуюся в том же ϵ-боксе, при условии что ребенок находится ближе к исходной точке плоскости f_1/f_2. В случае (d) ребенок добавляется в архив, и размер архива увеличивается на единицу

На рис. 20.15 приводится схематичное описание ϵ-ориентированного многокритериального эволюционного алгоритма (ϵ-MOEA). Мы можем поэкспериментировать с несколькими вариациями этого алгоритма. Например, обычно мы используем турнирный отбор с турниром размером 2 для отбора родителя x из P; однако мы можем применить любой другой метод отбора. Обычно мы случайно отбираем родителя α из архива A; опять же, мы можем применить любой другой метод отбора. Метод рекомбинации, который мы используем для создания ребенка, может быть любым методом из раздела 8.8. Если мы применяем рекомбинацию с несколькими родителями, то нам нужно решить, сколько родителей отбирать из P и сколько отбирать из A.

Инициализировать популяцию кандидатных решений $P = \{x_j\}$ для $j \in [1, N]$.
Скопировать недоминированных особей из P в архив A.
While not (критерий останова)
  Отобрать одного родителя x из P и одного родителя α из A.

Создать ребенка c путем рекомбинации x и α
$D_P \leftarrow \{x \in P : c$ доминирует над $x\}$
If $D_P \neq \emptyset$ **then**
Заменить случайную $x \in D_P$ на c
End if
$D_A \leftarrow \{\alpha \in A : c$ доминирует над $\alpha\}$
If $D_A \neq \emptyset$ **then**
Добавить c в A
Удалить D_A из A
else if уравнение (20.38) удовлетворено **then**
Добавить c в A
Удалить α из A
else if (c доминирован относительно A) и ($B(c) \neq B(\alpha)$ для всех α) **then**
Добавить c в A
End if
Next поколение

Рис. 20.15. Приведенный выше псевдокод описывает алгоритм ϵ-MOEA для решения оптимизационной задачи с k целевыми критериями

20.4.3. *Генетический алгоритм с сортировкой по уровню недоминирования (NSGA)*

Генетический алгоритм с сортировкой по уровню недоминирования (nondominated sorting genetic algorithm, NSGA) был предложен в публикации [Srinivas и Deb, 1994] и основан на идеях, изложенных в публикации [Goldberg, 1989a]. Алгоритм NSGA назначает стоимость каждой особи, основывая свое решение на том, насколько сильно она доминирует. Сначала мы копируем всех особей во временную популяцию T. Затем находим всех недоминированных особей в T; этим особям, которые мы обозначаем как множество B, присваивается наименьшее значение стоимости. Далее мы удаляем B из T, отыскиваем всех недоминированных особей в сокращенном множестве T и назначаем им значение стоимости 2. Мы повторяем этот процесс до тех пор, пока всем особям не будет назначено значение стоимости на основе их уровня доминирования. Затем используем значения стоимости $\phi(\cdot)$ на рис. 20.16 для выполнения отбора и рекомбинируем особей в P, используя желаемый эволюционный алгоритм и любой желаемый метод рекомбинации. Наконец, мы мутируем детскую популяцию, заменяем родителей детьми и переходим к следующему поколению.

Рисунок 20.16 показывает, что мы начинаем с популяции из N особей, обычно генерируемых случайно. В каждом поколении вычисляем значение всех K значений стоимостной функции для всех N особей. Мы копируем особей во временную популяцию T. Назначаем всем недоминированным особям стоимостное значение 1. Удаляем всех этих особей из T, находим всех недоминированных особей в уменьшенном множестве T и назначаем им стоимостное значение 2. Мы повторяем этот процесс до тех пор, пока всем особям не будет назначено стоимостное значение, опираясь на их уровень доминирования. Затем используем стоимостные значения $\phi(\cdot)$ на рис. 20.16 для выполнения отбора и рекомбинируем особей в P, используя любой желаемый эволюционный алгоритм и любой желаемый метод рекомбинации. Наконец, мы мутируем детскую популяцию, заменяем родителей детьми и переходим к следующему поколению.

Инициализировать популяцию кандидатных решений $P = \{x_j\}$ для $j \in [1, N]$.
While not (критерий останова)
  Временная популяция $T \leftarrow P$.
  Уровень недоминирования $c \leftarrow 1$.
  While $|T| > 0$
    $B \leftarrow$ недоминированные особи в T.
    Стоимость $\phi(x) \leftarrow c$ для всех $x \in B$.
    Удалить B из T.
    $c \leftarrow c + 1$
  Next уровень недоминирования
  $C \leftarrow N$ детей, созданных в результате рекомбинации особей в P.
  Вероятностно мутировать детей в C
  $P \leftarrow C$
Next поколение

Рис. 20.16. Схематическое описание генетического алгоритма с сортировкой по уровню недоминирования (NSGA) для решения оптимизационной задачи с k целевыми критериями. Мы используем значения $\phi(x_j)$ функции стоимости для отбора родителей с целью рекомбинации

Алгоритм NSGA иногда критикуют за его неэффективность из-за внутреннего цикла для оценивания стоимости $\phi(\cdot)$ на рис. 20.16. Однако оценивания функций $f_i(\cdot)$ в реальных задачах составляют основную вычислительную нагрузку, и цикл недоминирования в алгоритме NSGA добавляет тривиальный объем вычислительных издержек.

Алгоритм NSGA-II

Алгоритм NSGA-II является модификацией алгоритма NSGA [Deb и соавт., 2002a]. В алгоритме NSGA-II вычисляется стоимость особи x, принимая во внимание не только особей, которые доминируют над ней, но и особей, над которыми она доминирует. Для каждой особи мы также вычисляем расстояние скученности, находя расстояние до ближайших особей вдоль каждой размерности целевой функции. Мы используем расстояние скученности, чтобы изменить приспособленность каждой особи. В алгоритме NSGA-II для реализации элитарности архив не используется, а используется эволюционная стратегия $(\mu + \lambda)$ (см. главу 6).

В алгоритме NSGA устанавливается расстояние скученности каждой особи x равным ее среднему расстоянию до ближайших соседей вдоль каждой размерности целевой функции. Например, предположим, что в алгоритме NSGA имеется N особей. Далее предположим, что у особи x есть вектор целевой функции

$$f(x) = [f_1(x), \ldots, f_k(x)]. \tag{20.39}$$

Для каждой размерности целевой функции мы находим ближайшее большее значение и ближайшее меньшее значение в популяции следующим образом:

$$f_i^-(y) = \max_y [f_i(y) \text{ такая, что } f_i(y) < f_i(x)]$$

$$f_i^+(y) = \min_y [f_i(y) \text{ такая, что } f_i(y) > f_i(x)] \tag{20.40}$$

Затем мы вычисляем расстояние скученности x как

$$d(x) = \sum_{i=1}^{k} f_i^+(y) - f_i^-(y)). \tag{20.41}$$

Особи, которые находятся в более скученных участках пространства целевой функции, склонны иметь меньшее расстояние скученности. Особи в предельных значениях пространства целевой функции имеют бесконечное расстояние скученности:

$$d(x) = \infty \text{ для } x \in \left\{ \arg\min_y f_i(y) \cup \arg\max_y f_i(y) \text{ для всех } i \in [1, k] \right\}. \tag{20.42}$$

Расстояние скученности соответствует половине периметра самого большого гиперкуба, именуемого в публикации [Deb и соавт., 2000] кубоидом, границы которого не выходят за пределы координат пространства целевой функции ближайших соседей x в каждой размерности. На рис. 20.17 показан гиперкуб в пространстве двумерных целевых функ-

ций, представляющем собой прямоугольник[2]. На рис. 20.17 ближайшие соседи x в направлении f_1 – это A и C, ближайшие соседи x в направлении f_2 – это A и B.

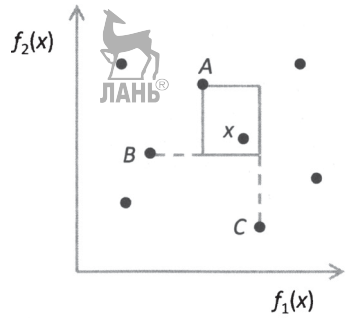


Рис. 20.17. Иллюстрация расчета расстояния скученности в NSGA-II. Расстояние скученности x получается как половина периметра самого большого гиперкуба (являющегося прямоугольником в двумерном пространстве), границы которого не выходят за пределы координат пространства целевой функции ближайших соседей x

Теперь, когда каждая особь в популяции имеет расстояние скученности, мы используем его в качестве вторичного параметра сортировки для получения ранга каждой особи. Как и в алгоритме NSGA на рис. 20.16, мы ранжируем каждую особь на основе его уровня недоминирования, но мы также включаем более мелкозернистый уровень ранжирования на основе расстояния скученности. То есть x ранжируется лучше y, если $\phi(x) < \phi(y)$, либо если $\phi(x) = \phi(y)$ и $d(x) > d(y)$. В отличие от алгоритма NSGA на рис. 20.16, в котором для отбора родителей с целью рекомбинации используется $\phi(x)$, в алгоритме NSGA-II вместо этого для отбора родителей с целью рекомбинации используются описанные выше ранги.

20.4.4. *Многокритериальный генетический алгоритм (MOGA)*

Многокритериальный генетический алгоритм (multi-objective genetic algorithm, MOGA) был представлен в публикации [Fonseca и Fleming, 1993]. Как и алгоритм NSGA, он назначает значения стоимости на основе доминирования, но алгоритм MOGA подходит к назначению стоимости с противоположного направления. В отличие от алгоритма NSGA, в ко-

[2] Вопреки публикации [Deb et al., 2000], гиперкуб не является самым большим гиперкубом, который охватывает x без включения каких-либо других точек. Как видно по рис. 20.17, это определение даст другой гиперкуб и, как правило, приведет к отличающейся результативности алгоритма NSGA-II.

тором стоимость особи x назначается на основе того, сколько уровней особей должно быть удалено из популяции, прежде чем x станет недоминированной, в алгоритме MOGA стоимость особи x назначается на основе того, сколько особей над ней доминирует. Мы назначаем одинаковую стоимость всем недоминированным особям. Для каждой доминированной особи x мы назначаем ее стоимость на основе того, сколько особей над ней доминирует, а также на основе того, сколько особей находится рядом с ней. Подобно использованию расстояния скученности в алгоритме NSGA-II, данный подход способствует многообразию популяции.

В алгоритме MOGA x ранжируется лучше y (то есть $\phi(x) < \phi(y)$, если над ней доминирует меньшее число особей в популяции P (то есть $d(x) < d(y)$)[3] либо если над ней доминирует одинаковое число особей, и возле x имеется меньше особей, чем возле y в пространстве целевой функции (то есть $s(x) < s(y)$). Такой подход к ранжированию можно выразить следующим образом:

$$\begin{aligned} d(x) &= |x' \in P : x' \text{ доминирует } x| \\ s(x) &= |x' \in P : ||f(x) - f(x')|| < \sigma| \\ \phi(x) < \phi(y) \text{ если } &\{d(x) < d(y), \text{ или } d(x) = d(y) \text{ и } s(x) < s(y)\} \end{aligned} \qquad (20.43)$$

где σ – это определяемый пользователем параметр обмена и $||\cdot||$ – метрический показатель расстояния. Обмен также может быть реализован автоматически, поэтому пользователю определять параметр обмена не требуется [Ahn и Ramakrishna, 2007]. Рисунок 20.18 дает схематичное описание алгоритма MOGA. Как и в случае с другими многокритериальными эволюционными алгоритмами, описанными в этой главе, мы можем реализовать ряд вариаций на рис. 20.18, таких как разные методы рекомбинации и разные подходы к обеспечению элитарности.

Инициализировать популяцию кандидатных решений $P = \{x_j\}$ для $j \in [1, N]$.
While not (критерий останова)
  Применить уравнение (20.43) для отыскания ранга $\phi(x_j)$ для каждого $x_j \in P$.
  $C \leftarrow N$ детей, созданных в результате рекомбинации особей в P
  Вероятностно мутировать детей в C
  $P \leftarrow C$
Next поколение

Рис. 20.18. Схематичное описание многокритериального генетического алгоритма MOGA) для решения оптимизационной задачи с k целевыми критериями. Мы используем ранги $\phi(x_j)$ для отбора родителей с целью рекомбинации

[3] Обратите внимание на изменение терминологии, в отличие от алгоритма NSGA-II: $d(x)$ – это расстояние скученности в алгоритме NSGA-II, но уровень доминирования в алгоритме MOGA.

20.4.5. *Генетический алгоритм с нишированием по Парето (NPGA)*

Генетический алгоритм с нишированием по Парето (niched pareto genetic algorithm NPGA) был предложен в публикации [Horn и соавт., 1994]. Данный алгоритм похож на алгоритмы NSGA и MOGA в том, что в нем стоимость назначается на основе доминирования. Алгоритм NPGA является попыткой уменьшить вычислительные усилия алгоритмов NSGA и MOGA. Мы случайно отбираем двух особей из популяции, x_1 и x_2. Затем случайно отбираем популяционное подмножество S, которое обычно составляет около 10 % популяции. Если над одной из особей x_1 или x_2 доминирует одна из особей S, а над другой нет, то недоминированная особь, обозначенная как r, выигрывает турнир и отбирается для рекомбинации. Если над обеими особями x_1 и x_2 доминирует, по крайней мере, одна особь в S, либо ни над одной из двух особей не доминирует ни одна особь в S, то для выбора победителя турнира мы используем обмен информацией о приспособленности, то есть особь, которая находится в наименее скученном участке пространства целевой функции, выигрывает турнир. Этот процесс отбора можно описать следующим образом:

$$
\begin{aligned}
d_i &= |y \in S : y \succ x_i| \text{ для } i \in [1, 2] \\
s_i &= \text{расстояние скученности } x_i \text{ для } i \in [1, 2] \\
r &= \begin{cases} x_1 & \text{если: } \begin{cases} (d_1 = 0) \text{ и } (d_2 > 0), \text{ или} \\ (d_1 > 0) \text{ и } (d_2 > 0) \text{ и } (s_1 < s_2), \text{ или} \\ (d_1 = 0) \text{ и } (d_2 = 0) \text{ и } (s_1 < s_2) \end{cases} \\ x_2 & \text{в противном случае} \end{cases}
\end{aligned}
\tag{20.44}
$$

где d_i – это число особей, которые доминируют над x_i, s_i – расстояние скученности для особи x_i, r – особь (x_1 либо x_2), которую мы в итоге отбираем для рекомбинации. Расстояние скученности s может быть вычислено с помощью метода из раздела 8.6.3.1, либо оно может использовать уравнение (20.43), либо может быть любым другим вычислением, которое квантифицирует скученность для особей x_1 и x_2. Расстояние скученности меньше для особей, которые находятся в более скученных участках поискового пространства или пространства целевой функции. Использование расстояния скученности в алгоритме NPGA поощряет многообразие; как и другие подобные алгоритмы, это делает его особо подходящим для мультимодальных задач или задач, в которых пользователь заинтересован в поиске хороших потенциальных решений в широко разделенных участках пространства функции или поискового пространства. Обратите внимание, что расстояние скученности уравне-

ния (20.44) может быть вычислено либо в пространстве функции, либо в поисковом пространстве, в зависимости от приоритетов пользователя.

Рисунок 20.19 дает схематичное описание алгоритма NPGA. Алгоритм NPGA с его случайно отбираемым подмножеством S популяции в уравнении (20.44) был первым многокритериальным эволюционным алгоритмом, который экономил вычислительные усилия за счет сокращения числа особей, участвующих в процессе ранжирования [Coello Coello, 2009]. Как и в случае других многокритериальных эволюционных алгоритмов в этой главе, мы можем кастомизировать рис. 20.19, включив элитарность, архив, разные стратегии рекомбинации либо разные стратегии отбора.

Инициализировать популяцию кандидатных решений $P = \{x_j\}$ для $j \in [1, N]$.
While not (критерий останова)
  $R \leftarrow \emptyset$
  While $|R| < N$
    Случайно отобрать двух особей x_1 и x_2 из P.
    Случайно отобрать популяционное подмножество $S \subset P$.
    Применить уравнение (20.44) для отбора r и $\{x_1, x_2\}$.
    $R \leftarrow \{R, r\}$
  End while
  Рекомбинировать особей в R для получения N детей.
  Случайно мутировать детей.
  $P \leftarrow$ детей
Next поколение

Рис. 20.19. Схематичное описание генетического алгоритма с нишированием по Парето (NPGA) для решения оптимизационной задачи с k целевыми критериями

20.4.6. *Эволюционный алгоритм с расчетом силы Парето (SPEA)*

Эволюционный алгоритм с расчетом силы Парето (strength pareto evolutionary algorithm, SPEA) был первым многокритериальным эволюционным алгоритмом, в котором элитарность использовалась явным образом [Zitzler и Thiele, 1999], [Zitzler и соавт., 2004]. Разумеется, любой из ранее обсуждавшихся многокритериальных эволюционных алгоритмов может быть реализован с помощью элитарности, но по какой-то причине большинство из них не включало элитарность при первоначальном внедрении. Элитарность обычно является здравым смыслом как в однокритериальных, так и в многокритериальных эво-

люционных алгоритмах. Кроме того, элитарность теоретически необходима для того, чтобы гарантировать сходимость в многокритериальных эволюционных алгоритмах [Rudolph и Agapie, 2000]. Однако если в многокритериальном эволюционном алгоритме используется подход, основанный на предпочтениях пользователя, и предпочтения меняются с течением времени, то элитарность может привести к снижению результативности [Zitzler и соавт., 2000]. Это похоже на недостатки применения элитарности, присущие для задач динамической оптимизации, где функции приспособленности варьируются во времени (см. главу 21).

Алгоритм SPEA работает, поддерживая в архиве всех недоминированных особей, которые обнаруживаются в процессе самообучения. Всякий раз, когда мы находим недоминированную особь, мы копируем ее в архив. Присваиваем значение силы $S(\alpha)$ каждой архивированной особи a на основе того, над сколькими особями в популяции α доминирует:

$$S(\alpha) = \frac{|x \in \{P\} \text{ такая, что } \alpha \succ x|}{N+1} \quad \text{для всех } \alpha \in A, \tag{20.45}$$

где P – это множество кандидатных решений, N – размер популяции P и A – архивное множество. Обратите внимание, что $S(\alpha) \in [0, 1)$. Для каждой особи x в P мы находим множество $\alpha(x)$ всех архивированных особей, которые доминируют над ней. Затем мы вычисляем сырую стоимость x, обозначаемую как $R(x)$, как сумму сил особей в $\alpha(x)$:

$$\begin{aligned} R(x) &= 1 + \sum_{y \in \alpha(x)} S(y), \quad \text{для всех } x \in P, \\ \text{где } \alpha(x) &= \{y \in A : y \succ x\}. \end{aligned} \tag{20.46}$$

Добавление единицы в приведенное выше уравнение гарантирует, что $R(x) \geq 1$, что, в свою очередь, гарантирует, что $R(x) > S(\alpha)$ для всех $x \in P$ и $\alpha \in A$. Заметим, что если x имеет низкую сырую стоимость, то x является высокорезультативной особью[4].

Рисунок 20.20 иллюстрирует расчеты силы и сырой стоимости для архива размером $|A| = 3$ и популяции размером $|P| = 6$. На рис. 20.20 показаны значения силы точек фронта Парето как нормализованное число особей, над которыми значения силы доминируют. На рисунке также показана сырая стоимость каждой доминированной точки как 1 плюс сумма сил точек фронта Парето, которые над ней доминируют. Обра-

[4] В литературе по алгоритму SPEA $R(x)$ часто упоминается как «сырая приспособленность», но мы используем термин «сырая стоимость», чтобы не противоречить интуитивному пониманию, что низкая стоимость хороша, а низкая приспособленность плоха.

тите внимание, что сырая стоимость становится больше, по мере того как особи отдаляются от фронта Парето (то есть по мере того, как точки фронта Парето все больше доминируют над особями). Кроме того, обратите внимание на доминированную особь в левом верхнем углу рисунка с сырой стоимостью 9/7 и сравните ее с двумя доминированными особями в правом нижнем углу со значениями сырой стоимости 10/7. Особи в правом нижнем углу имеют более высокую сырую стоимость, потому что они находятся в более скученном участке пространства целевой функции. Поскольку они находятся в скученном участке, сила точки фронта Парето, которая над ними доминирует, больше, что приводит к повышению их сырой стоимости.

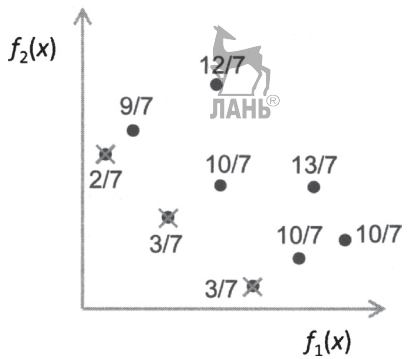


Рис. 20.20. Иллюстрация расчетов силы и сырой стоимости алгоритмом SPEA для задачи двухкритериальной минимизации. Особи фронта Парето отмечены символом *x*, а их значения силы показаны рядом с ними. Особи, не принадлежащие фронту Парето, показаны в виде заполненных кружков, а их сырые стоимостные значения показаны рядом с ними

Как уже упоминалось выше, в каждом поколении все недоминированные особи в $\{P, A\}$ добавляются в архив A. Однако это может привести к неограниченному росту архива. Алгоритм SPEA обрабатывает эту потенциальную проблему с помощью метода кластеризации [Zitzler и Thiele, 1999]. Если в архиве есть $|A|$ особей, то мы начнем с определения каждой особи как кластера. Затем мы объединяем два ближайших кластера в один, тем самым уменьшая число кластеров A на единицу. Повторяем этот процесс до тех пор, пока архив не будет содержать N_A кластеров, то есть желаемый размер архива. Наконец, мы сохраняем только одну точку из каждого кластера, обычно ближайшую к центру кластера.

SPEA2

Улучшенная версия алгоритма SPEA, обозначаемая как SPEA2, предлагает некоторые улучшения оригинального алгоритма [Zitzler и соавт., 2001]. Во-первых, мы назначаем значение силы $S(\alpha)$ не только особям в архиве, но и особям в популяции:

$$S(\alpha) = |x \in \{P, A\} \text{ такая, что } \alpha \succ x| \text{ для всех } \alpha \in \{P, A\}. \quad (20.47)$$

Мы также видим из сравнения этого уравнения с уравнением (20.45), что мы не нормализуем значения силы.

Во-вторых, мы вычисляем сырую стоимость каждой особи в P немного по-другому, суммируя силы доминирующих особей как в популяции, так и в архиве:

$$R(x) = \sum_{y \in \alpha(x)} S(y), \text{ для всех } x \in P,$$
$$\text{где } \alpha(x) = \{y \in \{P, A\} : y \succ x\}. \quad (20.48)$$

Мы также видим из сравнения этого уравнения с уравнением (20.46), что мы не добавляем единицы при расчете сырой стоимости.

В-третьих, мы модифицируем сырую стоимость каждой $x \in P$ на основе того, сколько особей находится рядом; то есть штрафуем стоимость особей, которые находятся рядом со многими другими особями в пространстве целевой функции. Мы делаем это, находя расстояние между $f(x)$ и $f(y)$, для всех $x \in P$ и для всех $y \in \{P, A\}$ – такое, что $y \neq x$. Этот метрический показатель расстояния может быть любой векторной нормой, которую пользователь считает подходящей, хотя мы обычно используем евклидову норму. Для каждой $x \in P$ мы сортируем расстояния между ним и каждой $y \in \{P, A\}$ в возрастающем порядке, поэтому у нас есть упорядоченный список расстояний для каждой x с элементами ($|P| + |A|$). Затем мы отбираем j-й элемент в списке расстояний, в результате получая расстояние между x и ее j-м ближайшим соседом, обозначаемым как $\sigma_j(x)$. Для отбора j мы можем использовать разные стратегии; например, некоторые исследователи имели хороший успех с $j = y\sqrt{|P| + |A|}$, а другие просто устанавливали $j = 1$ [Zitzler и соавт., 2004]. Мы определяем плотность x как

$$D(x) = \frac{1}{\sigma_j(x) + \gamma}, \quad (20.49)$$

где выбираем константу γ в знаменателе, для того чтобы гарантировать, что $D(x) < 1$. В оригинальной статье по алгоритму SPEA2 предлагается

$\gamma = 2$ [Zitzler и соавт., 2001]. Наконец, мы получаем модифицированную стоимость особи x, добавляя в плотность сырую стоимость:

$$C(x) = R(x) + D(x). \tag{20.50}$$

Поскольку все недоминированные особи имеют значение сырой стоимости, равное 0, как видно из уравнения (20.48), и поскольку $D(x) < 1$ для всех x, мы видим, что все недоминированные особи имеют стоимость $C(x) < 1$.

Четвертая модификация алгоритма SPEA2 касается управления размером архива. В алгоритме SPEA нет нижней границы размера архива, но в алгоритме SPEA2 значение размера архива поддерживается постоянным. Если в какой-то момент во время работы алгоритма SPEA2 размер архива становится слишком малым, мы добавляем в архив самых низкостоимостных особей из популяции, даже если они доминированные, до тех пор, пока размер архива не достигнет желаемого значения. В алгоритме SPEA используется метод кластеризации для сокращения размера архива, в случае если он становится слишком большим, но в алгоритме SPEA2 используется другой подход. В алгоритме SPEA2, если размер архива становится слишком большим, мы удаляем особей, находя расстояние от каждой $x \in A$ до ближайшего соседа в пространстве целевых функций:

$$D_{\min}(x) = \min_y \left[\sum_{i=1}^{k} (f_i(x) - f_i(y))^2, \text{ где } y \in A \text{ и } y \neq x \right], \text{ для } x \in A. \tag{20.51}$$

В результате получаем $|A|$ значений $D_{\min}(x)$, где $|A|$ – это размер архива. Далее мы используем D для обозначения множества особей, которые имеют наименьшее значение $D_{\min}(x)$):

$$D = \{x : D_{\min}(x) \leq D_{\min}(y) \text{ для всех } y \in A\}. \tag{20.52}$$

D будет иметь в нем, по крайней мере, две особи, поскольку расстояние между любыми двумя особями y и x является таким же, как и расстояние между y и x. Среди всех особей Д мы находим особь, обозначаемую как $x_{\min}$, которая является ближайший к любой архивированной особи не в D:

$$x_{\min} = \arg\min_x \left[\min_y \sum_{i=1}^{k} (f_i(x) - f_i(y))^2, \text{ где } y \in A \text{ и } y \neq D \right]. \tag{20.53}$$

Мы удаляем $x_{\min}$ из архива, что сокращает размер архива на единицу. Если $|A|$ слишком велик, то повторяем уравнения (20.51)–(20.53), удаляя

одну особь во время каждой итерации до тех пор, пока архив не достигнет желаемого размера.

Пятая, и последняя, модификация алгоритма SPEA2 заключается в том, что в отборе и рекомбинации с целью создания популяции в следующем поколении участвуют только члены A. Литература включает несколько вариаций и модификаций алгоритмов SPEA и SPEA2, но на рис. 20.21 приводится схематичное описание базового алгоритма.

N = размер популяции
N_A = максимальный размер архива
Инициализировать популяцию кандидатными решениями $P = \{x_j\}$ для $j \in [1, N]$.
Инициализировать архив A как пустое множество.
While not (критерий останова)
  Скопировать недоминированных особей из P в A:
    $A \leftarrow \{A \cup \{x \in P : \nexists\{y \in \{P, A\} : y \succeq x\}\}$
  Удалить доминированных особей из A.
  While $|A| > N_A$
    Применить метод кластеризации (SPEA) либо
      уравнения (20.51)–(20.53) (SPEA2) для удаления особей из A.
  End while
  If SPEA2 **then**
    While $|A| < N_A$
      Добавить самую низкостоимостную недублирующую особь из P в A:
        $A \leftarrow \{A,$ (arg $\min_x C(x)$ – такую, что $x \in P, x \notin A\}$
    End while
  End if
  Применить уравнение (20.46) (SPEA) либо уравнение (20.50) (SPEA2)
    для расчета стоимости каждой особи в P.
  Отобрать родителей из $\{P, A\}$ (SPEA) либо из A (SPEA2).
  Применить метод рекомбинации для создания детей C из родителей.
  Вероятностно мутировать детскую популяцию C.
  Применить метод замены для замены особей в P на особей в C.
Next поколение

Рис. 20.21. Схематичное описание эволюционного алгоритма с расчетом силы Парето (SPEA и SPEA2)

Рисунок 20.21 включает принципы алгоритма SPEA, но упускает многие детали, оставляя их для изобретательности исследователя. Здесь мы проясним несколько моментов относительно рис. 20.21 и упомянем несколько возможностей для модификаций.

- Читателю необходимо выбрать размер популяции N и размер архива N_A. Как правило, $N_A < N$.
- Инструкция «Скопировать недоминированных особей из P в A» указывает на то, что все особи $x \in P$ необходимо сравнивать со всеми особями $y \in \{P, A\}$. Любая особь x, над которой не доминирует ни одна особь y, копируется в A. Однако такая инструкция оставляет без ответа вопрос о том, следует ли удалять недоминированных особей из P. Поскольку недоминированные особи находятся в A, возможно, не имеет смысла хранить их в качестве дубликатов в P. Но тут сразу возникает следующий вопрос о размере P. Если мы удалим недоминированных особей из P, то нужно ли нам заменить их другими особями? Мы можем оставлять P в сокращенном состоянии и всегда создавать N детей независимо от размера P, либо можем заменять недоминированных особей, которых мы удаляем из P, некоторыми случайно созданными особями.
- Цикл «While $|A| > N_A$» удаляет особей из скученных участков пространства целевой функции, в случае если архив слишком большой. Для этого алгоритмы SPEA и SPEA2 имеют свои собственные методы, и читатель, несомненно, может найти другие методы, с которыми можно поэкспериментировать.
- Цикл «While $|A| < N_A$» добавляет в архив низкостоимостных особей, если он слишком малый, но только для алгоритма SPEA2. В алгоритме SPEA этот шаг опущен, то есть $|A|$ не имеет нижней границы.
- Инструкция «Отобрать родителей из $\{P, A\}$ (SPEA) либо из A (SPEA2)» оставляет много места для проявления гибкости. Для этого шага мы можем применить любой тип отбора (см. раздел 8.7). Успех алгоритмов SPEA и SPEA2 сильно зависит от реализации этой инструкции.
- Инструкция «Применить метод рекомбинации для создания детей C из родителей» также оставляет много места для проявления гибкости. Для создания детей мы можем применить любой описанный в этой книге эволюционный алгоритм и любой тип рекомбинации (см. раздел 8.8). Как и в случае отбора, успех алгоритмов SPEA и SPEA2 сильно зависит от реализации рекомбинации.
- Инструкция «Применить метод замены для замены особей в P на особей в C» может быть реализована несколькими разными способами. Если $|C| = |P|$, то мы можем просто заменить P на C. Если $|C| < |P|$, то можем отобрать лучших N особей из $C \cup P$ для замены

P, либо мы можем отобрать N особей из $C \cup P$ с помощью пропорционального по приспособленности алгоритма. Если $|C| > |P|$, то можем отобрать лучших N особей из C либо лучших N особей из $C \cup P$, либо можем отобрать N особей из C или $C \cup P$ с помощью пропорционального по приспособленности алгоритма.

■ Пример 20.7

Данный пример иллюстрирует подрезание архива алгоритма SPEA2, как описано в уравнениях (20.51)–(20.53). На рис. 20.22 показан архив недоминированных особей в двумерном пространстве целевых функций. Предположим, что нам нужно сократить размер архива с восьми особей до пяти. Сначала мы находим особей, которые ближе всего друг к другу, – на рисунке это особи f и g. Мы удаляем g, потому что она ближе к своему следующему ближайшему соседу h, чем f к своему следующему ближайшему соседу e. Теперь, когда мы удалили g, находим следующие две особи, которые ближе всего друг к другу, – на рисунке это особи d и e. Мы удаляем d, потому что она ближе к следующему ближайшему соседу c, чем e к своему следующему ближайшему соседу f. После удаления d мы находим следующие две особи, которые ближе всего друг к другу, – это a и b. Мы удаляем b, потому что она ближе к своему следующему ближайшему соседу c, чем a к своему следующему ближайшему соседу c. Теперь мы сократили размер архива до пяти особей, как и хотели. Обратите внимание, что этот метод всегда сохраняет предельных особей в архиве (в данном примере a и h).

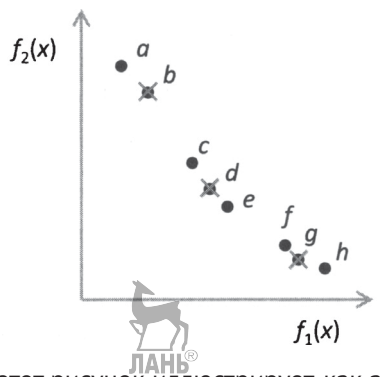


Рис. 20.22. Пример 20.7: этот рисунок иллюстрирует, как архив недоминированных решений сводится к меньшему числу особей в алгоритме SPEA2. На этом рисунке, взятом из публикации [Zitzler и соавт., 2004], размер архива уменьшается с восьми до пяти особей путем удаления (g), (d) и (b) в указанном порядке

■

■ Пример 20.8

Подход алгоритма SPEA2 для ограничения размера архива вполне может привести к ухудшению аппроксимации фронта Парето. На рис. 20.23 показан пример того, как это может произойти. На рис. 20.23(a) показана аппроксимация фронта Парето, содержащая четыре точки. Если мы хотим сохранить размер архива равным трем, то отбрасываем точку *c*, поскольку она является самой скученной особью. Однако в более позднем поколении эволюционный алгоритм вполне может найти недоминированное решение *e*, которое он добавляет в архив, как показано на рис. 20.23(b). Теперь, поскольку *b* является наиболее скученной особью на рис. 20.23(b), алгоритм SPEA2 удаляет *b* из архива, сохраняя там *e*. Мы видим, оглядываясь назад, что должны были сохранить в популяции особь *c*, поскольку она доминирует над новой архивной точкой *e*. Это говорит о том, что хотя подход алгоритма SPEA2 на основе расстояния может быть хорошим методом для подрезания множества особей, возможно, будет лучше никогда не отбрасывать недоминированных особей [Zitzler и соавт., 2004].

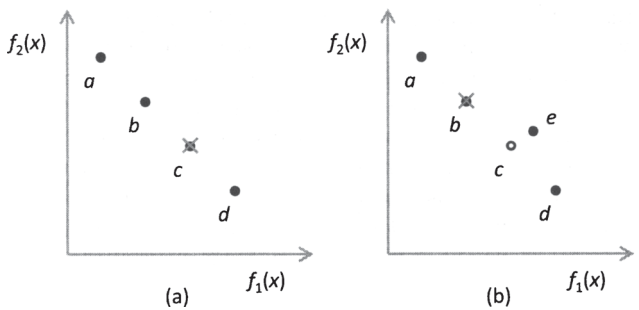


Рис. 20.23. Пример 20.8: этот рисунок иллюстрирует, как процесс подрезания архива в алгоритме SPEA2 вполне может непреднамеренно повредить аппроксимацию фронта Парето. Особь *c* удаляется из архива из-за его переполненности, и особь *e* добавляется в архив позже, несмотря на то что над ней могла доминировать особь *c*, если бы *c* была сохранена

■

В алгоритмах SPEA, SPEA2 и любом другом многокритериальном эволюционном алгоритме, который включает в себя архив, мы можем использовать архив несколькими разными способами. Во-первых, мы можем использовать его просто для хранения недоминированных решений; этот подход обеспечивает самую высокую степень сегрегации между популяцией многокритериального эволюционного алгоритма

и ее архивом. Во-вторых, мы можем копировать отдельных особей из архива в детскую популяцию в конце каждого поколения; такой подход допускает некоторое взаимодействие между популяцией и архивом. В-третьих, мы можем дать архиву участвовать в процессе отбора, с тем чтобы в рекомбинации участвовала не только популяция, но и архив; этот подход, который используется в алгоритмах SPEA и SPEA2, обеспечивает наивысшую степень взаимодействия между популяцией и архивом.

20.4.7. *Эволюционная стратегия с архивированием по Парето (PAES)*

Эволюционно-стратегический алгоритм с архивированием по Парето (Pareto archived evolution strategy, PAES) был представлен в публикации [Knowles и Corne, 2001] и мотивирован стратегией эволюции (1 + 1) (см. главу 6). В каждом поколении одна особь производит одного ребенка, используя мутацию. Всякий раз, когда родитель создает ребенка, мы добавляем его в архив, если над ним не доминирует ни одна особь из архива. Если размер архива превышает пороговое значение, то мы подрезаем архив, удаляя особь с наименьшим расстоянием скученности (то есть особь, которая находится в наиболее скученном участке поискового пространства или пространства целевых функций). В оригинальном алгоритме PAES для вычисления расстояния скученности используются решетки в пространстве целевых функций. Они аналогичны ϵ-боксам алгоритма ϵ-MOEA (см. раздел 20.4.2). Это также похоже на кластерный подход, описанный в публикации [Parks и Miller, 1998], который не добавляет особей в архив, если они недостаточно отличаются от особей, находящихся в настоящее время в архиве. На рис. 20.24 приведено схематичное описание общего алгоритма PAES.

N_A – верхняя граница размера архива
Случайно сгенерировать популяцию кандидатных решений $P = \{x_j\}$ для $j \in [1, N]$.
Инициализировать архив A как пустое множество.
While not (критерий останова)
  Отобрать родителя x из P.
  Мутировать x.
  If над x не доминирует ни одна особь в A **then**
    Добавить x в A: A $\leftarrow \{A \cup x\}$
  End if
  If $|A| > N_A$ **then**

Вычислить расстояние скученности $s(\alpha)$ для всех $\alpha \in A$
$\alpha_{min} \leftarrow \arg\min_{\alpha} s(\alpha)$
Удалить α_{min} из A.
End if
Next поколение

Рис. 20.24. Схематичное описание эволюционной стратегии с архивированием по Парето (PAES). $s(\alpha)$ – это расстояние скученности для α, которое мало для особей в скученных участках поискового пространства или пространства целевых функций

20.5. Многокритериальная биогеографическая оптимизация

В данном разделе показано, как биогеографическая оптимизация (BBO), которая обсуждается в главе 14, может быть объединена с некоторыми из ранее обсуждавшихся в этой главе подходов многокритериальных эволюционных алгоритмов. Сочетание подхода BBO с разными подходами, принятыми в многокритериальных эволюционных алгоритмах, приводит к нескольким многокритериальным биогеографическим оптимизационным алгоритмам (multi-objective biogeography-based optimization, MOBBO). Позже мы продемонстрируем сравнительное исследование этих алгоритмов MOBBO на нескольких стандартных многокритериальных эталонных задачах. Данный раздел может послужить шаблоном для расширения любого другого эволюционного алгоритма на многокритериальную оптимизацию.

20.5.1. *Биогеографическая оптимизация с векторным оцениванием*

В этом разделе мы обсудим вопрос объединения алгоритма биогеографической оптимизации (BBO) с генетическим алгоритмом, использующим векторное оценивание (VEGA) (раздел 20.3.2). Напомним, что на рис. 20.8 представлен алгоритм VEGA для задачи k-критериальной оптимизации. Поскольку алгоритм BBO основывается на миграции, мы предлагаем основывать иммиграцию в многокритериальной биогеографической оптимизации на значении k_i-й целевой функции каждой особи, где k_i – это индекс случайной целевой функции в i-м миграционном испытании. Затем предлагаем основывать эмиграцию на значении k_e-й целевой функции каждой особи, где k_e – это тоже индекс случай-

ной целевой функции. В результате получится алгоритм биогеографической оптимизации с векторным оцениванием (vector evaluated BBO, VEBBO) рис. 20.25.

Инициализировать популяцию кандидатных решений $P = \{x_j\}$ для $j \in [1, N]$.
While not (критерий останова)
  Вычислить стоимость $f_i(x_j)$ для каждого целевого критерия i
    и для каждой особи $x_j \in P$.
  $r_{ji} \leftarrow$ ранг x_j относительно i-й целевой функции, $j \in [1, N]$, $i \in [1, k]$
  Скорости иммиграции $\lambda_{ji} \leftarrow r_{ji}\Sigma_{q=1}^{N} r_{qi}$ для $j \in [1, N]$, $i \in [1, k]$
  Скорости эмиграции $\mu_{ji} \leftarrow 1 - \lambda_{ji}$ для $j \in [1, N]]$, $i \in [1, k]$
  For each особь x_j, где $j \in [1, N]$
    For each независимая переменная $s \in [1, n]$
      $k_i \leftarrow$ rand(1, k) = равномерно распределенное целое
      $r \leftarrow$ rand(0, 1)
      If $r < x_{j,k_j}$ **then** выполнить иммиграцию
        $k_e \leftarrow$ rand(1, k) = равномерно распределенное целое
        Вероятностно отобрать эмигранта x_e, где
          $\Pr(x_e = x_m) = \mu_{m,k_e} / \Sigma_{q=1}^{N} \mu_{q,k_e}$ для $m \in [1, N]$
        $x_j(s) \leftarrow x_e(s)$
      End иммиграция
    Next независимая переменная
  Next особь x_j
  Вероятностно мутировать популяцию P.
Next поколение
$P_s \leftarrow$ недоминированные элементы популяции P

Рис. 20.25. Схематичное описание алгоритма биогеографической оптимизации с векторным оцениванием (VEBBO) для решения n-мерной оптимизационной задачи с k целевыми критериями. В каждом поколении лучшая особь x_b по отношению к i-му целевому значению имеет ранг $r_{bi} = 1$, и худшая особь x_w имеет ранг $r_{wi} = N$

20.5.2. *Алгоритм BBO с сортировкой по уровню недоминирования*

Теперь мы обсудим, как объединить алгоритм биогеографической оптимизации (BBO) с генетическим алгоритмом, использующий сортировку по уровню недоминирования (NSGA) (см. раздел 20.4.3). Напомним, что на рис. 20.16 представлен алгоритм NSGA. Для того чтобы модифицировать рис. 20.16 с учетом алгоритма BBO, нам нужно толь-

ко изменить инструкцию рекомбинации «$C \leftarrow N$ детей, созданных в результате рекомбинации...» на операцию миграции алгоритма BBO. В результате получится алгоритм биогеографической оптимизации с сортировкой по уровню недоминирования (nondominated sorting BBO, NSBBO), показанный на рис. 20.26.

```
Скорости иммиграции λ_j ← φ(x_j)/Σ_{q=1}^{N} φ(x_q) для j ∈ [1, N]
Скорости эмиграции μ_j ← 1 – λ_j для j ∈ [1, N]
For each особь x_j, где j ∈ [1, N]
  For each независимая переменная s ∈ [1, n]
    r ← rand(0, 1)
    If r < λ_j then
      Вероятностно отобрать эмигранта x_e, где
      Pr(x_e = x_m) = μ_m/Σ_{q=1}^{N} μ_q для m ∈ [1, N]
      x_j(s) ← x_e(s)
    End иммиграция
  Next независимая переменная
Next особь x_j
Детская популяция C ← {x_j}
```

Рис. 20.26. Схематичное описание алгоритма биогеографической оптимизации с сортировкой по уровню недоминирования (NSBBO) для решения оптимизационной задачи с n независимыми переменными, k целевыми критериями и популяцией размером N. Данный псевдокод заменяет строку «$C \leftarrow N$ детей, созданных в результате рекомбинации» на рис. 20.16

20.5.3. *Алгоритм BBO с нишированием по Парето*

В этом разделе предлагается простой способ объединения алгоритма биогеографической оптимизации (BBO) с генетическим алгоритмом, использующим ниширование по Парето (NPGA) (см. раздел 20.4.5). Напомним, что рис. 20.19 представляет алгоритм NPGA. Аналогично алгоритму NSBBO предыдущего раздела, для того чтобы модифицировать рис. 20.19 с учетом алгоритма BBO, нам нужно только изменить инструкцию рекомбинации «Рекомбинировать особей в R» на операцию миграции алгоритма BBO. Поскольку алгоритм NPGA уже отбирает особей в R на основе недоминирования, мы можем для отбора операций миграции просто применить равные скорости миграции для каждой особи в R. В результате получится алгоритм биогеографической оптимизации с нишированием по Парето (niched Pareto BBO, NPBBO) рис. 20.27.

```
For each особь x_j ∈ R, где j ∈ [1, N]
  For each независимая переменная s ∈ [1, n]
    r ← rand(0, 1)
    If r < 1/N then
      Вероятностно отобрать эмигранта x_e, где
      Pr(x_e = x_m) = 1/N для m ∈ [1, N]
      x_j(s) ← x_e(s)
    End иммиграция
  Next независимая переменная
Next особь x_j
Детская популяция ← {x_j}
```

Рис. 20.27. Схематичное описание миграционной части алгоритма биогеографической оптимизации с нишированием по Парето (NPBBO) для решения оптимизационной задачи с n независимыми переменными, k целевыми критериями и популяцией размером N. Данный псевдокод заменяет строку «Рекомбинировать особей в R для получения N детей» на рис. 20.19

В этом месте следует отметить, что мы могли бы объединить алгоритмы многокритериальной биогеографической оптимизации (MOBBO), обсуждаемые в этом разделе, со всеми возможными вариациями алгоритма BBO, обсуждаемыми в главе 14. Например, вместо иммиграционных BBO мы могли бы использовать эмиграционные алгоритмы BBO, нелинейные миграционные кривые. Мы могли бы использовать смешанную миграцию и миграционные группы независимых переменных, а не по одной за раз (см. раздел 14.5). Мы могли бы использовать для миграции временную популяцию, чтобы не менять ни одну эмигрирующую особь до тех пор, пока все миграции не будут завершены. В общем случае все расширения, вариации и гибридизации алгоритма BBO, обсуждаемые в литературе, могут быть объединены со всеми подходами к многокритериальной биогеографической оптимизации, обсуждаемыми в этом разделе, в результате получив целый ряд алгоритмов MOBBO. Мы могли бы еще приложить такие же усилия, используя любой из других эволюционных алгоритмов, обсуждаемых в этой книге. Особенно плодотворным направлением исследований могло бы стать распространение на многокритериальную оптимизацию многих новых эволюционных алгоритмов, которые были представлены совсем недавно, в том числе обсуждались в главе 17.

20.5.4. *Алгоритм BBO с расчетом силы Парето*

В этом разделе предлагается метод объединения алгоритма биогеографической оптимизации (BBO) с эволюционно-стратегическим алгоритмом, использующим расчет силы Парето (SPEA), или с алгоритмом SPEA2 (см. раздел 20.4.6). Напомним, что на рис. 20.21 представлены алгоритмы SPEA и SPEA2. Для того чтобы модифицировать рис. 20.21 с учетом алгоритма BBO, нам нужно изменить инструкцию «Отобрать родителей» и инструкцию «Применить метод рекомбинации». Мы можем сделать это путем расчета скоростей миграции с сырой стоимостью уравнения (20.46) для алгоритма SPEA либо с модифицированной стоимостью уравнения (20.50) для алгоритма SPEA2. Затем можем реализовать миграцию алгоритма BBO с использованием этих скоростей. В этом разделе мы используем подход алгоритма SPEA, в котором родители могут быть отобраны как из популяции P, так и из архива A. В результате получится алгоритм биогеографической оптимизации с расчетом силы Парето (strength Pareto BBO, SPBBO) рис. 20.28.

Применить уравнение (20.45) для расчета силы $S(\alpha)$ каждой особи $\alpha \in A$
Рассчитать стоимость $R(\alpha) \leftarrow 1 - S(\alpha)$ для каждой $\alpha \in A$
Применить уравнение (20.46) для расчета стоимости $R(x)$ каждой особи $x \in P$
Скорости иммиграции: $\lambda_j \leftarrow R(x_j)/\Sigma_{q=1}^{|P|} R(x_q)$ для всех $x_j \in P$
Скорости эмиграции: $\mu_j \leftarrow R(x_j)/\Sigma_{q=1}^{|P|+|A|} R(x_q)$ для всех $x_j \in \{P, A\}$
For each особь $x_j \in F$
  For each независимая переменная $s \in [1, n]$
    $r \leftarrow$ rand(0, 1)
    If $r < \lambda_j$ **then**
      Вероятностно отобрать эмигранта x_e, где
      $\Pr(x_e = x_m) = \mu_m / \Sigma_{q=1}^{|P|+|A|}$ для $x_m \in \{P, A\}$
      $x_j(s) \leftarrow x_e(s)$
    End иммиграция
  Next независимая переменная
Next особь x_j

Рис. 20.28. Схематичное описание миграционной части алгоритма биогеографической оптимизации с расчетом силы Парето (SPBBO) для решения оптимизационной задачи с n независимыми переменными. Данный псевдокод заменяет шесть строк, начиная со строки «Применить уравнение (20.46)» и заканчивая строкой «Применить метод замены» на рис. 20.21. Обратите внимание, что иммиграция происходит у особей в F, в то время как эмиграция происходит у особей в $P \cup A$

20.5.5. *Симуляции многокритериальной биогеографической оптимизации*

Здесь мы представляем результаты симуляции для алгоритмов MOBBO, представленных в предыдущих подразделах. Для каждого алгоритма используем популяцию размером 100 и лимит поколений 1000. Применяем скорость мутации 1 % в расчете на независимую переменную на поколение и равномерную мутацию, центрированную в середине поисковой области (см. раздел 8.9). Проверяем популяцию на наличие дубликатов каждые 100 поколений и заменяем любые дубликаты, которые мы находим, случайно сгенерированными особями (см. раздел 8.6.1).

Мы встраиваем элитарность в алгоритмы VEBBO, NSBBO и NPBBO путем исследования популяции в каждом поколении на наличие недоминированных особей. Если мы находим недоминированных особей, то заменяем худших особей в популяции (в терминах уровня недоминирования, как показано на рис. 20.16) на двух случайно отобранных недоминированных особей из предыдущего поколения.

Мы не используем элитарность в алгоритме SPBBO, потому что алгоритм SPBBO хранит недоминированных особей в архиве (см. рис. 20.28). Когда в алгоритме SPBBO мы перемещаем недоминированных особей из популяции P в архив A, то заменяем этих особей в P на случайно сгенерированных особей, поэтому $|P|$ остается неизменно равным 100. Мы не используем нижнюю границу $|A|$, но ограничиваем максимальное значение $|A|$ равным 100 с помощью простого алгоритма кластеризации, как описано в разделе 20.4.6.

Мы тестируем четыре алгоритма мультикритериальной биогеографической оптимизации на нескольких неограниченных мультикритериальных эталонах приложения C.3, каждый из которых имеет 10 размерностей и отобран из-за их разнообразия. Мы используем задачу U01 из-за ее выпуклого фронта Парето, задачу U04 из-за ее вогнутого фронта Парето, задачу U06 из-за ее прерывистого фронта Парето и задачу U10, потому что у нее три целевых критерия (другие эталоны, которые мы тестируем, имеют только два целевых критерия).

Мы оцениваем результативность работы алгоритмов двумя метрическими показателями: гиперобъемом с опорной точкой S' уравнения (20.22) и нормализованным гиперобъемом с опорной точкой S_n' уравнения (20.23). Эти метрические показатели не учитывают многообразия, но учитывают близость аппроксимированного фронта Парето к истинному фронту Парето. Гиперобъем с опорной точкой S' также

принимает во внимание число точек в аппроксимированном фронте Парето.

В табл. 20.1 приведены результаты, усредненно по 10 симуляциям Монте-Карло. Мы видим, что в целом алгоритм SPBBO работает лучше всех. Однако в случае прерывистой функции задачи U06 алгоритм VEBBO работает лучше всего в условиях общего гиперобъема, тогда как алгоритм NSBBO работает лучше всего в условиях нормализованного гиперобъема. То есть алгоритм VEBBO находит лучшее сочетание качества и числа фронта Парето, в то время как алгоритм NSBBO находит лучшее качество. В условиях более сложной функции задачи U10, в отличие от алгоритма SPBBO, который находит лучший общий гиперобъем, алгоритм NSBBO находит более качественные точки фрона Парето. Таким образом, мы можем сказать, что алгоритм SPBBO обычно работает лучше всего из-за своего архива, но нет никаких гарантий, что он будет работать лучше в условиях любой конкретной задачи.

Таблица 20.1. Результаты алгоритмов многокритериальной биогеографической оптимизации (BBO) на четырех 10-мерных эталонных функциях. Таблица показывает относительный гиперобъем и нормализованный относительный гиперобъем с использованием линейной миграции биогеографической оптимизации (первое число в каждой паре) и с использованием синусоидальной миграции (второе число в каждой паре). Лучшая результативность для каждого эталона в части относительного гиперобъема и нормализованного относительного гиперобъема показана **жирным** шрифтом. См. раздел 14.4.1, где обсуждается линейная миграция в сопоставлении с синусоидальной миграцией в алгоритме биогеографической оптимизации

		U01	U04	U06	U1O
VEBBO	Гипер	(8.02, 8.93)	(2.73, 2.51)	(**63.53**, 59.99)	(299.68, 444.24)
	Норм	(1.28, 1.98)	(0.19, 0.18)	(21.18, 19.95)	(76.72, 103.07)
NSBBO	Гипер	(5.76, 8.01)	(3.27, 3.51)	(56.51, 48.44)	(373.51, 599.38)
	Норм	(1.26, 1.69)	(0.21, **0.22**)	(**21.94**, 20.09)	(101.40, **118.89**)
NPBBO	Гипер	(9.23, 18.78)	(2.95, 3.05)	(57.92, 48.31)	(419.96, 577.28)
	Норм	(1.18, 2.02)	(0.15, 0.15)	(20.01, 20.09)	(58.65, 57.33)
SPBBO	Гипер	(13.87, **33.10**)	(4.48, **4.60**)	(14.82, 18.33)	(934.29, **3884.32**)
	Норм	(0.90, **2.24**)	(**0.22**, **0.22**)	(3.47, 4.08)	(90.65, 108.08)

20.6. Заключение

Эта глава не претендует на полное изложение предмета многокритериальных эволюционных алгоритмов и лишь слегка касается одних из самых популярных многокритериальных эволюционных алгоритмов и связанных с ними идей. Был предложен ряд других многокритериальных эволюционных алгоритмов, и новые алгоритмы постоянно появляются в литературе. Среди книг по многокритериальным эволюционным алгоритмам следует выделить такие издания, как [Sakawa, 2002], [Collette и Siarry, 2004], [Coello Coello и соавт., 2007], [Deb, 2009] и [Tan и соавт., 2010]. Кроме того, для многокритериальных эволюционных алгоритмов становятся популярными роевые подходы, такие как оптимизация на основе роя частиц (см. главу 11) [Banks и соавт., 2008].

В публикации [Coello Coello, 2006] дан интересный, высокоуровневый исторический взгляд на многокритериальные эволюционные алгоритмы. Самый ранний технический обзор данных алгоритмов приводится в работе [Fonseca и Fleming, 1995], а дополнительный обзор дается в публикациях [Coello Coello, 1999], [Van Veldhuizen и Lamont, 2000], [Zitzler и соавт., 2004] и [Konak и соавт., 2006]. Хотя некоторые из этих публикаций быстро устаревают из-за быстрого расширения исследований в области многокритериальных эволюционных алгоритмов, все они по-прежнему очень полезны и полны глубоких идей по фундаментальным вопросам, связанным с многокритериальными эволюционными алгоритмами.

В этой главе мы увидели, что многообразие является в многокритериальных эволюционных алгоритмах важным фактором. Некоторые подходы к увеличению многообразия включают обмен информацией о приспособленности, решетки, кластеризацию, скученность, энтропию и ограничение спаривания [Fonseca и Fleming, 1995]. Многообразие может быть важным соображением и в однокритериальных эволюционных алгоритмах, о чем говорится в разделе 8.6. Все диверсифицирующие механизмы упомянутого раздела могут быть применены к обсуждаемым в этой главе многокритериальным эволюционным алгоритмам.

Некоторые важные темы для будущих исследований многокритериальных эволюционных алгоритмов включают следующее:

- автоматическая онлайновая адаптация регулировочных параметров многокритериальных эволюционных алгоритмов;
- гибридизация многокритериальных эволюционных алгоритмов со стратегиями локального поиска;

- многокритериальные эволюционные алгоритмы, которые могут обеспечивать хорошую результативность с малым числом оцениваний функций;
- многокритериальные эволюционные алгоритмы для ряда целевых критериев (более двух или трех);
- включение пользовательских предпочтений в многокритериальные эволюционные алгоритмы;
- концептуально новые подходы к проектированию многокритериальных эволюционных алгоритмов, которые не опираются на стандартные методы ранжирования по Парето;
- теория многокритериальных эволюционных алгоритмов и математические модели.

Мы обсудим некоторые из этих тем в следующих далее абзацах.

Вторая перечисленная выше тема – встраивание стратегий локального поиска в многокритериальные эволюционные алгоритмы – является очень важной. В частности, многокритериальные эволюционные алгоритмы могут быть гибридизированы с алгоритмами, основанными на производных (или другими методами локального поиска) с целью точной регулировки результатов оптимизации. Такие алгоритмы называются меметическими алгоритмами, поскольку они предполагают (по крайней мере, неявно) использование в гибридизированном алгоритме специфичной для конкретной задачи информации. Меметические стратегии, по всей видимости, часто используются в однокритериальной оптимизации [Ong и соавт., 2007], но они до сих пор мало использовались в многокритериальных оптимизационных задачах, хотя есть несколько исключений [Jaszkiewicz and Zielniewicz, 2006].

Проблема дорогостоящих оцениваний функций приспособленности для многокритериальных оптимизационных задач важна, потому что эти типы функций приспособленности часто возникают в реальных задачах, а также потому, что многокритериальные оптимизационные задачи нередко требуют гораздо больше оцениваний функций приспособленности, чем однокритериальные задачи. В разделе 21.1 обсуждаются дорогостоящие функции приспособленности в целом, однако также существуют эволюционно-алгоритмические исследования, которые специально предназначены для многокритериальных оптимизационных задач с дорогостоящими функциями приспособленности [Chafekar и соавт., 2005], [Eskandari и Geiger, 2008], [Knowles, 2005], [Santana-Quintero и соавт., 2010]. В публикации [Goh и Tan, 2007] обсуждаются

многокритериальные эволюционные алгоритмы для задач с оцениванием шумных функций приспособленности.

Разработка многокритериальных эволюционных алгоритмов для многочисленных целевых критериев (10 или более) также является важной областью будущих исследований. В данной области было опубликовано несколько результатов, но гораздо более сложной задачей не обязательно является аппроксимация множества Парето, а скорее то, как помочь лицам, принимающим решения, выбирать решение на основе аппроксимации множества Парето многокритериальным эволюционным алгоритмом. В некоторых исследованиях в области многокритериальных задач подчеркивается их особая сложность [Fleming и соавт., 2005], но в других исследованиях показывается, что найти хорошую аппроксимацию Парето для задач с многочисленными целевыми критериями на самом деле легче [Schütze и соавт., 2011]. По размышлении, это имеет интуитивный смысл, потому что чем противоречивее целевые критерии, тем вероятнее, что какое-то случайное кандидатное решение даст хорошую результативность, по крайней мере по одному из этих целевых критериев. В публикации [Van Veldhuizen и Lamont, 2000] показано, что чем больше целевых критериев мы добавляем в многокритериальную оптимизационную задачу, тем больше становится множество Парето.

Однако хотя аппроксимацию множества Парето, возможно, найти проще с помощью большего числа целевых критериев, она также потребует большего числа кандидатных решений. Например, если мы предположим, что для двухкритериальной задачи 10 кандидатных решений могут дать хорошую аппроксимацию множества Парето, то в двухкритериальном эволюционном алгоритме нам, вероятно, понадобится, по крайней мере, 100 особей. Это означает, что для k-критериальной оптимизационной задачи нам вполне может понадобиться 10^k особей, то есть нам потребуется, в частности, 100 000 особей для относительно небольшой пятикритериальной оптимизационной задачи. Таким образом, проблема с многокритериальными задачами заключается не в теоретической сложности аппроксимации множества Парето, а в практических трудностях вычислительных усилий и аппроксимации многомерных поверхностей только с несколькими точками. В публикации [Schiitze и соавт., 2011] дается хороший обзор текущих исследований в области многокритериальных задач.

Все это подводит нас к вопросу о предпочтениях пользователей. Когда мы решаем многокритериальную оптимизационную задачу, то часто имеем предопределенные предпочтения. Например, мы можем придавать большее значение определенным целевым критериям, чем

другим, или можем придавать большее значение определенным сочетаниям целевых критериев. Пользователь не всегда заинтересован в получении всего множества Парето. Если бы мы могли каким-то образом встраивать пользовательские предпочтения в многокритериальный эволюционный алгоритм, то могли бы направлять эволюцию в предпочитаемый пользователем участок поискового пространства или пространства целевых критериев. Тогда мы сможем уменьшить размер популяции многокритериального эволюционного алгоритма для решения многих многокритериальных задач, поскольку нам не нужно будет аппроксимировать все множество Парето. Другой подход к решению многокритериальных задач – просто сокращать число целевых критериев, поскольку такие задачи часто имеют целевые критерии, которые коррелируют друг с другом [Lopez Jaimes и соавт., 2009].

Аппроксимация множества Парето достаточно сложна, но даже если эволюционный алгоритм может получить хорошую аппроксимацию, то каким образом лицо, принимающее решения, сможет выбирать из большого множества потенциальных решений? Некоторые из обсуждаемых в этой главе многокритериальных эволюционных алгоритмов встраивают предпочтения пользователей (см. раздел 20.3.1), но мы не рассматривали эту тему в явном виде. Первая попытка встроить предпочтения пользователей в многокритериальный эволюционный алгоритм была предложена в публикации [Tanaka и Tanino, 1992]. С тех пор был предложен ряд других подходов; обратитесь к работе [Thiele и соавт., 2009], в которой дан хороший обзор.

Теоретические результаты для многокритериальных эволюционных алгоритмов редки, и поэтому в этой области есть много места для внесения существенного вклада. В публикации [Rudolph и Agapie, 2000] предоставляется предварительная марковская модель для многокритериальных эволюционных алгоритмов; несколько других ученых выполнили теоретическое исследование многокритериальных эволюционных алгоритмов [Zitzler и соавт., 2010], но, по сравнению с однокритериальными эволюционными алгоритмами, теоретические исследования многокритериальных эволюционных алгоритмов редки.

Наконец, читатель, который заинтересован в дополнительных результатах и исследованиях в области многокритериальных эволюционных алгоритмов, должен взять на заметку, что Карлос Коэльо Коэльо поддерживает исчерпывающую и полезную библиографию статей, связанных с многокритериальной эволюционной оптимизацией, в вебе. По состоянию на август 2012 г. его библиография содержала 4861 ссылку [Coello Coello, 2012b].

Задачи

Письменные упражнения

20.1. При наличии множества точек и задачи многокритериальной оптимизации всегда ли верно, что одна точка доминирует над другими?

20.2. Существует ли множество Парето у каждой многокритериальной оптимизационной задачи?

20.3. На рис. 20.1 показан эскиз фронта Парето для многокритериальной оптимизационной задачи, в которой мы хотим минимизировать оба целевых критерия. Нарисуйте и объясните образец выпуклого фронта Парето для многокритериальной оптимизационной задачи, в которой мы хотим: (a) минимизировать f_1 и максимизировать f_2, (b) максимизировать f_1 и минимизировать f_2, (c) максимизировать и f_1, и f_2.

20.4. Рассмотрим следующие ниже точки и значения целевых функций для многокритериальной минимизационной задачи:

$$
\begin{aligned}
f_1(x^{(1)}) &= 1, \qquad f_2(x^{(1)}) = 1,\\
f_1(x^{(2)}) &= 1, \qquad f_2(x^{(2)}) = 2,\\
f_1(x^{(3)}) &= 2, \qquad f_2(x^{(3)}) = 1,\\
f_1(x^{(4)}) &= 2, \qquad f_2(x^{(4)}) = 2.
\end{aligned}
$$

a) Какая точка доминирует над всеми остальными?
b) Над какой точкой доминирует $x^{(2)}$> и $x^{(3)}$?
c) Какая точка является недоминированной?
d) Какая точка является оптимальной по Парето?

20.5. Рассмотрим пункты задачи 20.4. Для каких значений ϵ точки $x^{(2)}$, $x^{(3)}$ и $x^{(4)}$ аддитивно ϵ-доминируют над $x^{(1)}$? В случае каких значений ϵ они мультипликативно ϵ-доминируют над $x^{(1)}$?

20.6. Приведите пример двух точек в двумерной двухкритериальной минимизационной задаче таким образом, чтобы одна точка не мультипликативно ϵ-доминировала над другой точкой при любом значении ϵ.

20.7. Приведите пример двух фронтов Парето P_1 и P_2, оба имеющих одинаковое число точек, для которых объединение P_2 гиперобъемов больше, чем то же самое для P_1, но пересечение P_2 гиперобъемов меньше, чем то же самое для P_1.

20.8. Рассмотрим следующую аппроксимацию четырехточечного фронта Парето $f(x)$ и аппроксимацию трехточечного фронта Парето $f(y)$ двухкритериальной минимизационной задачи:

$$f_1(x^{(1)}) = 3, \quad f_2(x^{(1)}) = 4, \quad f_1(y^{(1)}) = 1, \quad f_2(y^{(1)}) = 3,$$
$$f_1(x^{(2)}) = 3, \quad f_2(x^{(2)}) = 3, \quad f_1(y^{(2)}) = 4, \quad f_2(y^{(2)}) = 3,$$
$$f_1(x^{(3)}) = 2, \quad f_2(x^{(3)}) = 2, \quad f_1(y^{(3)}) = 4, \quad f_2(y^{(3)}) = 1,$$
$$f_1(x^{(4)}) = 5, \quad f_2(x^{(4)}) = 2.$$

Каков охват x относительно y? Каков охват y относительно x? В соответствии с этими относительными значениями охвата какая аппроксимация фронта Парето лучше?

20.9. Почему мы должны исходить из того, что все целевые критерии неотрицательны в методе агрегирования произведений уравнения (20.26)?

20.10. Объясните разницу между элитарностью и архивом.

20.11. Каково наибольшее число особей, которые могут храниться в архиве алгоритма ϵ-MOEA с двумя целевыми критериями, где $f_i \in [0, f_{i,\max}]$ для $i \in [1, 2]$? Как будет в случае архива с тремя целевыми критериями?

20.12. Рисунок 20.17 иллюстрирует прямоугольник, который мы можем использовать для расчета скученности в алгоритме NSGA-II. Предположим, что вместо этого мы используем прямоугольник, самый большой из тех, которые охватывают x, не включая другие точки. Нарисуйте этот прямоугольник.

20.13. Рассмотрим две точки x и y в двумерном пространстве целевых функций. Предположим, расстояние скученности $d_1(x)$ алгоритма NSGA-II вычисляется с помощью ближайших соседей особи x в каждой размерности, а расстояние скученности $d_2(x)$ вычисляется по величине прямоугольника, охватывающего x без каких-либо других точек. Верно ли, что $d_1(x)) < d_1(y)$ значит, что $d_2(x) < d_2(y)$?

20.14. Рассмотрим рис. 20.29, который иллюстрирует особей в популяции и архиве для двухкритериальной максимизационной задачи с использованием алгоритма SPEA [Zitzler и Thiele, 1999]. Каковы значения сырой стоимости каждой особи в популяции и значения силы каждой особи в архиве?

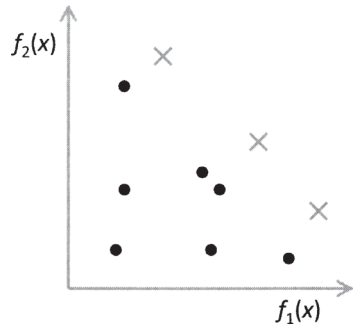


Рис. 20.29. Задача 20.14: кружки – это особи в популяции, а крестики – особи в архиве

20.15. Предположим, что у нас есть три особи в двухкритериальной минимизационной задаче:

$$f_1(x_1) = 3, \qquad f_2(x_1) = 4,$$
$$f_1(x_2) = 4, \qquad f_2(x_2) = 3,$$
$$f_1(x_3) = 2, \qquad f_2(x_3) = 2.$$

Каковы ранги r_{ij} алгоритма VEBBO на рис. 20.25?

20.16. Сколько ссылок приведено на веб-сайте Карлоса Коэльо Коэльо в списке «List of References on Evolutionary Multiobjective Optimization» (Список литературы по эволюционной многокритериальной оптимизации)?

Компьютерные упражнения

20.17. Примените исчерпывающий поиск с разрешающей способностью поисковой области, равной 0.01, для того чтобы найти множество Парето и фронт Парето для задачи

$$\min[\cos(x_1 + x_2), \sin(x_1 - x_2)],$$

где поисковая область для каждой размерности равна $[0,7\ \pi]$.

20.18. Примените подход на основе взвешенной суммы, для того чтобы свести два целевых критерия задачи 20.17 к одному. При каких значениях весов $w1$ и w_2 решение однокритериальной задачи равно фронту Парето двухкритериальной задачи?

20.19. Выполните гендерный эволюционный алгоритм на рис. 20.11 на многокритериальной задаче. Протестируйте результативность на

основе следующих вариаций: (а) разрешить только одного родителя на субпопуляцию в сопосоставлении с разрешением двух родителей на субпопуляцию; (б) разрешить каждому ребенку быть членом только одной субпопуляции в сопосоставлении с разрешением каждому ребенку быть членом более чем одной субпопуляции; (в) заменять дублирующих особей в каждом поколении в сопосоставлении с отсутствием замен дублирующих особей.

Глава 21

Дорогостоящие, шумные и динамические функции приспособленности

Эволюционным алгоритмам часто приходится решать оптимизационные задачи в присутствии широкого спектра неопределенностей.

– Яосю Цзинь и Юрген Branke [Jin and Branke, 2005]

Любой, кто работал над эволюционными алгоритмами до 1970 года, опередил свое время. В те ранние годы около десятка человек внесли фундаментальный вклад в эволюционные алгоритмы, и каждого из них можно было бы с достаточным основанием назвать «родоначальником эволюционных алгоритмов» либо, по крайней мере, одним из родоначальников[1]. Однако вся эта ранняя работа над эволюционными алгоритмами в некоторой степени провалилась из-за нехватки компьютерных ресурсов. Эволюционные алгоритмы в 1960-х годах должны были быть очень маленькими и простыми, для того чтобы их можно было выполнять за разумное время. Вычислительная мощность в 1960-х годах была просто-напросто недостаточна для проведения исследований или далеко идущих практических экспериментов с эволюционными алгоритмами.

В 1970-е годы вычислительные ресурсы стали доступнее и мощнее, а к 1980-м годам эволюционно-алгоритмические исследования вышли

[1] Насколько мне известно, в те почти доисторические времена не было женщин, работавших над эволюционными алгоритмами. Тот факт, что эволюционные алгоритмы имеют несколько отцов-основателей и не имеют «матерей», представляется вполне естественным ввиду гибкости их биологических основ.

из депрессивного состояния и стали активной областью исследований. Поиск в INSPEC, компьютеризированной базе данных научных статей в области компьютеров и техники, показывает ровно одну публикацию по генетическим алгоритмам в 1970-х годах, 37 публикаций в 1980-х годах, 7924 публикации в 1990-е годы и 35 440 публикаций в первом десятилетии XXI века[2]. Вычислительная техника сегодня достаточно мощная и позволяет любому желающему писать на настольном ПК программы с использованием эволюционных алгоритмов для решения интересных и сложных задач.

ЭВМ 1960-х годов работали с тактовой частотой до 10 МГц. Типичный настольный ПК в начале XXI века имеет тактовую частоту около 10 ГГц и даже быстрее, если принять во внимание несколько ядер. Мы стали свидетелями увеличения вычислительной мощности на три порядка за период с 1960 по 2010 год, тем не менее эволюционным алгоритмам нередко по-прежнему требуется несколько дней для полного их завершения. Это одна из причин, почему в эволюционно-алгоритмических исследованиях большое внимание уделяется распараллеливанию. Однако распараллеливание влечет за собой свой собственный ряд специфичных для конкретной задачи проблем.

Краткий обзор главы

В данной главе обсуждается вопрос сокращения вычислительной стоимости эволюционных алгоритмов. В реальном мире оценивания функции стоимости могут быть очень дорогостоящими. Эталонные функции, которые мы использовали до настоящего момента в этой книге, просты, и мы можем вычислять их в считанные миллисекунды. Но в реальном мире оценивание функции стоимости может занимать несколько дней, и в таких случаях мы не можем позволить себе выполнять эволюционный алгоритм, который требует тысячи оцениваний функции. В разделе 21.1 обсуждаются способы работы с дорогостоящими функциями стоимости.

Связанные с этим проблемы, с которыми мы сталкиваемся в реальном мире, включают в себя оценивания варьирующихся со временем функций стоимости и шумные функции стоимости. Функции стоимости могут изменяться со временем ввиду динамического и часто непредсказуемого характера нашего мира, поэтому в разделе 21.2 обсуждаются способы решения динамических оптимизационных задач.

[2] INSPEC – это большая, но не исчерпывающая исследовательская база данных. Поэтому приведенные здесь цифры дают пониженную оценку количества публикаций.

Наконец, функции стоимости часто зашумлены из-за отсутствия прецизионности, которая присутствует во многих задачах, либо из-за присущей двусмысленности в определении качества кандидатного решения, и поэтому в разделе 21.3 обсуждаются способы обработки шумных оптимизационных задач.

21.1. Дорогостоящие функции приспособленности

Во многих реальных задачах одно оценивание приспособленности может потребовать вычислений или экспериментов, которые занимают минуты, часы, дни или даже больше. Здесь мы обсудим способы сокращения времени оценивания приспособленности, позволяющие сделать эволюционные алгоритмы вычислительно менее требовательными.

Любой, кто работал с эволюционными алгоритмами с целью создания реальных приложений, знает из первых рук, что оценивание функции приспособленности является наиболее трудоемким аспектом алгоритма. Это не всегда относится к эталонам или академическим задачам, но это почти всегда относится к реальным задачам. Дж. Коза заходит так далеко, что утверждает, что вычислительные усилия, необходимые для расчета функции приспособленности «как правило, настолько велики, что они едва ли окупятся, чтобы рассматривать их на предмет всех других аспектов их выполнения» [Koza, 1992, приложение H]. Мы находим аналогичное утверждение в публикации [Banzhaf и соавт., 1998, раздел 11.1]: «почти все время, потребляемое генетическим алгоритмом, расходуется на оценивания приспособленности». Реальные задачи часто связаны с функциями приспособленности, которые содержат одну или несколько следующих характеристик [Knowles, 2005].

- Для оценивания одной функции приспособленности требуются минуты, часы или дни. Это особенно верно для функций приспособленности, которые должны оцениваться экспериментально, а не с помощью симуляции. Многие из самых ранних эволюционных алгоритмов были реализованы в экспериментальных системах из-за отсутствия ресурсов симулирования [Rechenberg, 1998], [Rechenberg, 1973].
- Оценивания функций приспособленности не могут быть распараллелены. Это особенно актуально для функций приспособленности, которые необходимо оценивать экспериментально и с ограниченными ресурсами. Например, для некоторых оптими-

зационных задач требуются уникальные экспериментальные инсталляции, требующие интенсивного взаимодействия с человеком или дорогостоящие с финансовой точки зрения. Иногда нам нужны человеческие эксперты для оценивания приспособленности кандидатных решений. Некоторые функции приспособленности нельзя квантифицировать, и вместо этого они требуют субъективных оцениваний со стороны экпертов. Это имеет место, например, во время генерирования алгоритмов, которые создают музыку или художественные объекты [Nierhaus, 2010].

- Число оцениваний функций приспособленности ограничено по времени или некоторым другим показателям. Это касается задач, которые должны быть решены к определенному сроку, задач, для которых эволюционный алгоритм должен работать в режиме реального времени, или функций приспособленности, которые должны быть оценены экспериментально профессионалами с уникальными навыками и с полной занятостью.

Некоторые способы сокращения вычислительных усилий, необходимых для оценивания функций приспособленности, включают следующее.

- Не пересчитывать стоимость особей, которые уже были оценены. Во многих эволюционных алгоритмах некоторые особи могут выживать без изменений из поколения в поколение. Особи, которые дублируются из поколения i в поколение $i + 1$, не должны оцениваться заново в поколении $i + 1$; мы уже знаем их стоимостные значения, учитывая, что задача не является динамической.
  Эта идея может быть расширена за счет отслеживания всех векторов кандидатных решений и связанных с ними стоимостных значений, встречающихся во время работы всего эволюционного алгоритма. В каждом поколении мы храним каждую особь и ее стоимость в архиве. Если у нас есть фиксированная популяция из N особей, то после T поколений у нас будет архив с NT особей (минус дубликаты). Архив не участвует в эволюционном процессе, мы используем его только для того, чтобы избегать ненужных оцениваний стоимости. Каждый раз, когда нам нужно оценить стоимость, мы сначала просматриваем архив, чтобы увидеть, была ли эта конкретная особь оценена в прошлом. После ряда поколений архив может стать довольно большим, и поиск в архиве перед каждым оцениванием стоимости будет дорогостоящим. Но

в зависимости от задачи этот поисковый процесс вполне может быть намного дешевле, чем оценивание стоимости.

- Оценивания функций приспособленности могут быть подрезаны, если видно, что они работают очень хорошо или очень плохо [Gathercole и Ross, 1997]. Если мы находимся на полпути оценивания функции приспособленности для особи и видим, что особь показывает очень хорошие результаты, то можем преждевременно выйти из процедуры оценивания, назначить ее приспособленности высокое приближенное значение и сэкономить половину вычислительных усилий для этого оценивания. Точно так же, если мы находимся на полпути оценивания функции приспособленности и видим, что особь показывает очень слабые результаты, то можем преждевременно выйти из процедуры оценивания, назначить ее приспособленности низкое приближенное значение и снова сэкономить половину вычислительных усилий.

Если нам необходимо выполнить оценивание функции стоимости для большого множества тестовых случаев, то мы можем аппроксимировать стоимость с помощью подмножества тестовых случаев. Например, предположим, что мы хотим минимизировать $f(x) = \sum_{i=1}^{M} f_i(x)$ по x. То есть функция приспособленности представляет собой композицию из нескольких подфункций приспособленности. Это часто случается, когда мы хотим оптимизировать функцию в нескольких разных операционных участках. Например, мы, возможно, захотим оптимизировать результативность отслеживания роботом траектории для нескольких разных исходных условий и для нескольких разных задач. Один из способов подойти к решению этой задачи – минимизировать $f_1(x)$ с первыми T поколениями. Затем минимизировать $f_1(x) + f_2(x)$ для следующих T поколений. Потом минимизировать $f_1(x) + f_2(x) + f_3(x)$ для следующих T поколений. Мы продолжаем этот процесс до тех пор, пока, наконец, не минимизируем $\sum_{i=1}^{M} f_i(x)$ в течение поколений от $(M - 1)T$ до MT. Еще один подход – минимизировать комбинацию случайной подборки из $f_i(x)$ функций, постепенно увеличивая число экземпляров по мере увеличения числа поколений. Такой подход называется стохастическим выборочным отбором [Banzhaf и соавт., 1998, раздел 10.1.5] и аналогичен лексикографическому упорядочению (см. раздел 20.3.3) и методу ϵ-ограничений (см. раздел 20.3.4) в многокритериальной оптимизации.

Если оценивание приспособленности проводится на персональных компьютерах, то могут быть применены стандартные методы ускоре-

ния выполнения программ. Эти методы включают в себя предварительное выделение памяти под массивы, использование оптимизированных возможностей конкретных языков программирования (например, в MATLAB векторизованные операции с массивами, а не циклы), снижение прецизионности компьютера, использование таблиц преобразования для сложных функций и отключение графических и выходных операций.

В оставшейся части данного раздела мы обсудим несколько способов аппроксимирования функций приспособленности (раздел 21.1.1), рассмотрим трансформацию функции приспособленности и то, как она может повысить результативность аппроксимации (раздел 21.1.2), разберем вопрос использования аппроксимации функций приспособленности в эволюционном алгоритме (раздел 21.1.3) и то, когда применять несколько аппроксимаций функций приспособленности в эволюционном алгоритме (раздел 21.1.4), коснемся вопроса опасности чрезмерно подогнанной аппроксимации функции приспособленности (раздел 21.1.5) и оценивания качества аппроксимации функции приспособленности (раздел 21.1.6).

21.1.1. *Аппроксимирование функции приспособленности*

Мы можем создавать модели функций приспособленности с целью сокращения усилий по оцениванию функций приспособленности. Также можем применять модели функций приспособленности для повышения результативности эволюционного алгоритма, даже если вычислительные усилия не являются узким местом. Мы называем такие модели суррогатами, поверхностями отклика или метамоделями [Shi и Rasheed, 2010]. Мы используем термин «суррогат», потому что модель приспособленности может рассматриваться как временная замена для более точного оценивания приспособленности. Применяем термин «метамодель», потому что часто оценивание приспособленности само по себе является всего лишь аппроксимацией (например, симуляцией, которая моделирует физический процесс), и поэтому модель функции приспособленности является моделью пониженного порядка модели более высокого порядка.

Предположим, что у нас есть функция приспособленности $f(x)$, которую мы вычислили на M особях $\{x_i\}$. Мы можем использовать эти M значений функции приспособленности для оценивания приспособленности в любой точке поискового пространства. Мы ожидаем, что оцен-

ка приспособленности будет иметь ошибки; в конце концов, если бы оценка была идеальной, то нам ни разу не пришлось бы выполнять дополнительные оценивания. Однако даже если оценка приспособленности содержит ошибки, они вполне могут быть достаточно малыми, тем самым делая оценку полезной. Рисунок 21.1 иллюстрирует суть идеи оценивания приспособленности. Мы генерируем оценку $\hat{f}(x)$ на основе известных значений функции приспособленности $f(x_i)$:

Рис. 21.1. Оценивание функции приспособленности. Мы используем точные значения функции приспособленности $\{f(x_i)\}$, для того чтобы аппроксимировать $f(x)$ для $x \notin \{x_i\}$

Аппроксимирование функций приспособленности с целью сокращения вычислительных усилий эволюционных алгоритмов восходит к 1960-м годам [Dunham и соавт., 1963], когда вычислительных ресурсов было гораздо меньше, чем сегодня. С тех пор в качестве алгоритма оценивания на рис. 21.1 исследователи испытали целый ряд разных алгоритмов. По сути дела, мы можем испробовать любой аппроксимационный или интерполяционный алгоритм. Общепринятые подходы включают алгоритм *k*-ближайших соседей, радиальные базисные функции, нейронные сети, нечеткую логику, кластеризацию, деревья решений, полиномиальные модели, модели кригинга[3], ряды Фурье, ряды Тейлора, модели NK[4], модели гауссовых процессов и опорно-векторные машины [Jin, 2005], [Shi и Rasheed, 2010]. В данной книге мы не обсуждаем детали этих подходов, но достаточно сказать, что для эволюционно-алгоритмической аппроксимации приспособленности может быть использован практически любой аппроксимационный алгоритм.

Один из простейших алгоритмов оценивания, который мы можем использовать на рис. 21.1, заключается в аппроксимировании приспособленности особи как приспособленности ближайшего соседа, который был уже оценен. Этот подход называется имитацией приспособленности и сводится к кусочно-постоянной аппроксимации ландшафта приспособленности. На рис. 21.2 показана имитация приспособленности.

[3] Кригинг (kriging) – метод разбиения пространства на блоки фиксированного размера и расчета исследуемого параметра в каждом блоке по выбранной модели вариограммы. Назван в честь автора по фамилии Krige. – *Прим. перев.*

[4] Модель *NK* – это математическая модель, описываемая как «регулировочно-прочный» ландшафт приспособленности. «Регулировочная прочность» отражает интуитивное понимание, согласно которому общий размер ландшафта и число его локальных «холмов и долин» можно регулировать с помощью изменений двух его параметров: *N* и *K*. – *Прим. перев.*

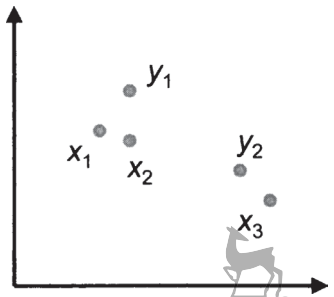


Рис. 21.2. Имитация приспособленности в двумерном поисковом пространстве. Особи y_1 и y_2 были оценены с помощью подпрограммы или эксперимента с функцией приспособленности, и поэтому их значения приспособленности точно известны. Особи x_1, x_2 и x_3 не были оценены. Мы можем аппроксимировать значения их приспособленности путем назначения каждой из них приспособленности ближайшей оцененной приспособленности, то есть $\hat{f}(x_1) = f(y_1)$ и $\hat{f}(x_2) = f(y_2)$ и $\hat{f}(x_3) = f(y_3)$

Один из аспектов рис. 21.1, который может быть важным, состоит в том, как обновлять оценку приспособленности $\hat{f}(\cdot)$ по мере того, как становятся доступными новые данные. Мы хотели бы, чтобы алгоритм оценивания приспособленности был рекурсивным, с тем чтобы мы могли обновлять $\hat{f}(\cdot)$ с минимальными усилиями всякий раз, когда становится доступной новая информация о приспособленности. Однако если мы думаем, что ландшафт приспособленности, возможно, является динамическим, тогда мы, возможно, захотим, чтобы алгоритм оценивания дисконтировал старые значения данных приспособленности при генерации $\hat{f}(\cdot)$. Когда мы обновляем аппроксимацию приспособленности, используя новые данные, мы называем это онлайновым суррогатным обновлением.

Еще один подход к аппроксимации приспособленности предполагает назначение приспособленности эволюционно-алгоритмическому ребенку на основе значений приспособленности его родителей. Это называется наследованием приспособленности [Smith и соавт., 1995]. Мы можем аппроксимировать приспособленность ребенка как среднее значение приспособленностей его родителей либо как средневзвешенное значение, которое зависит от того, насколько он похож на каждого родителя. Можем легко обобщить эту идею для эволюционных алгоритмов, которые для каждого ребенка используют любое число родителей. Мы также можем расширить эту идею, аппроксимируя приспособленность ребенка как взвешенное среднее значений приспособленности

всей популяции, опять же в зависимости от того, насколько он похож на каждую оцененную особь в популяции [Sastry и соавт., 2001]. Мы также можем использовать более сложные идеи наследования приспособленности, такие как учет корреляций между независимыми переменными и значениями приспособленности [Pelikan и Sastry, 2004]. В публикации [Ducheyne и соавт., 2003] автор пришел к выводу, что наследование приспособленности эффективно только для относительно простых задач. В частности, для многокритериальных задач наследование приспособленности эффективно только в том случае, если фронт Парето непрерывен и выпукл.

21.1.1.1. Полиномиальные модели

Кусочно-постоянная аппроксимация имитации приспособленности является хорошей отправной точкой, поскольку она показывает нам, как расширять аппроксимацию приспособленности на многочлены более высокой степени. Например, мы можем аппроксимировать приспособленность как линейную функцию

$$\hat{f}(x) = \alpha(0) + \sum_{k=1}^{n} \alpha(k)\, x(k), \tag{21.1}$$

где n – это размерность задачи и $x(k)$ – k-й элемент особи x. Этот простой пример полиномиальной модели также называется поверхностью отклика. Мы можем вычислить значения $\alpha(k)$, решив следующую задачу:

$$\min \sum_{i=1}^{M} \left(f(x_i) - \left[\alpha_0 + \sum_{k=1}^{n} \alpha(k) x_i(k) \right] \right)^2, \tag{21.2}$$

где M – это число особей, для которых у нас есть точные значения приспособленности, x_i – i-я особь, для которой у нас есть точное значение приспособленности, и $x_i(k)$ – k-й элемент особи x. Минимизация в уравнении (21.2) берется на $(n + 1)$ параметрах $\alpha(k)$ для $k \in [0, n]$. Мы можем решить уравнение (21.2) с помощью рекурсивного алгоритма наименьших квадратов [Simon, 2006, глава 3]. Благодаря этому по мере получения дополнительных значений приспособленности ($M = 1$, $M = 2$ и т. д.) для обновления решения уравнения (21.2) требуются лишь минимальные вычислительные усилия.

Мы можем написать модель, которая будет точнее линейной модели уравнения (21.1), следующим образом:

$$\hat{f}(x) = \alpha(0) + \sum_{k=1}^{n} \alpha(k)\, x(k) + \sum_{j,k=1}^{n} \alpha(j, k)\, x(j)\, x(k). \tag{21.3}$$

Это квадратичная модель с параметрами $(n^2 + n + 1)$. Она по-прежнему может быть решена рекурсивными наименьшими квадратами, потому что линейна по отношению к параметрам модели $\alpha(k)$ и $\alpha(j, k)$. После того как мы разберемся в идее полиномиального моделирования, мы можем поэкспериментировать с разными модельными формами, такими как

$$\hat{f}(x) = \alpha(0) + \sum_{k=1}^{n} \alpha(k)\, g(x(k)) + \sum_{j,k=1}^{n} \alpha(j, k) h(x(j), x(k)), \tag{21.4}$$

где $g(\cdot)$ и $h(\cdot)$ могут быть любыми функциями, линейными или нелинейными. Например, если у нас есть основания полагать, что для нашей конкретной оптимизационной задачи приспособленность вполне может быть хорошо представлена тригонометрическими функциями, то для $g(\cdot)$ мы можем применить синусоидальные и косинусоидальные члены.

Для аппроксимации модели приспособленности мы, возможно, захотим использовать методы, отличные от метода наименьших квадратов. Например, вместо того чтобы найти модель, которая решает уравнение (21.2), мы вполне можем предпочесть отыскать модель, которая решает

$$\min_{\{\alpha(k)\}} \max_{i} \left| f(x_i) - \left[\alpha_0 + \sum_{k=1}^{n} \alpha(k) x_i(k) \right] \right|, \tag{21.5}$$

где минимизация снова берется на $(n + 1)$ параметрах $\alpha(k)$. На рис. 21.3 показана разница между минимизацией суммы квадратов ошибок оценивания и минимизацией максимальной ошибки оценивания. Критерий наименьших квадратов уравнения (21.2) привлекателен тем, что его легко решить аналитически, но минимаксный критерий вполне может быть робастнее, поскольку он находит аппроксимацию, которая в результате дает наименьшую ошибку для худшего случая. Минимаксная аппроксимация принесет в жертву аппроксимационные ошибки в простых для подгонки участках поискового пространства, для того чтобы сократить аппроксимационные ошибки в более сложных участках поискового пространства.

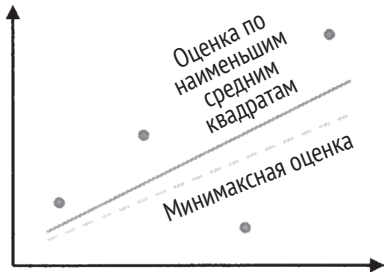


Рис. 21.3. Прямолинейная аппроксимция на основе наименьших средних квадратов в сопоставлении с минимаксной аппроксимацией. Аппроксимация на основе наименьших средних квадратов может быть решена аналитически, но минимаксная аппроксимция вполне может быть робастнее

21.1.1.2. Проектирование и анализ компьютерных экспериментов

Проектирование и анализ компьютерных экспериментов (design and analysis of computer experiments, DACE) является стохастическим аппроксимационным методом, который включает в себя диагностические тесты для измерения качества аппроксимации [Jones и соавт., 1998]. С учетом M оцениваний функции приспособленности $f(x_i)$ для n-мерных векторов x мы будем считать, что функция приспособленности может быть аппроксимирована как

$$f(x) = \mu + \epsilon(x), \tag{21.6}$$

где μ – это константа (но не обязательно среднее значение оцениваемых функций приспособленности $f(x_i)$) и $\epsilon(x)$ – поправочный член. Метод DACE исходит из того, что поправочный член $\epsilon(x)$ является гауссовым со средним значением μ и дисперсией σ^2 для всех x, то есть функция плотности вероятности (PDF) $f(x)$ равна

$$\text{PDF}(f(x)) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[\frac{-(f(x)-\mu)^2}{2\sigma^2}\right]. \tag{21.7}$$

Вместе с тем метод DACE также принимает важное допущение, что члены $\epsilon(x)$ *не являются* независимыми для разных значений x; то есть поправочные члены должны быть одинаковыми для похожих значений x. При этом метод DACE исходит из того, что коэффициент корреляции ρ_{ij} между $f(x_i)$ и $f(x_j)$ может быть выражен следующим образом:

$$d_{ij} = \sum_{k=1}^{n} \theta_k \left| x_i(k) - x_j(k) \right|^{p_k}$$

$$\rho_{ij} = \text{Corr}\,(f(x_i), f(x_j)) = \exp(-d_{ij}) \quad (21.8)$$

где $x_i(k)$ – это k-й элемент i-го кандидатного решения, $p_k \in [1, 2]$, и $\theta_k \geq 0$ – модельные параметры и $d_{ij} \geq 0$ – метрический показатель расстояния. Мы видим, что для малых d_{ij} векторы x_i и x_j имеют корреляцию, близкую к 1. Для больших d_{ij} векторы x_i и x_j имеют корреляцию, близкую к 0. При наличии M оцениваний функции приспособленности мы собираем их в векторе и параметризуем так, как показано в уравнении (21.6):

$$f(x) = [f(x_1) \ldots f(x_M)]^T$$
$$= \mu 1_M + [\epsilon(x_1) \ldots \epsilon(x_M)]^T, \quad (21.9)$$

где 1_M – это M-элементный вектор-столбец, в котором каждый элемент равен 1. Обратите внимание, что мы используем запись $f(x)$ для представления приспособленности одного кандидатного решения x, а также для представления M-элементного вектора, содержащего M приспособленностей $\{x_i\}$; смысл должен быть ясен из контекста. Гауссова функция плотности вероятности M функций приспособленности уравнения (21.9) тогда задается как

$$\text{PDF}(f(x)) = \frac{1}{(2\pi)^{M/2} \mid C \mid^{1/2}} \exp\left[\frac{(f(x) - \mu 1_M)^T C^{-1} (f(x) - \mu 1_M)}{2} \right], \quad (21.10)$$

где C – это ковариационная матрица $f(x)$. Напомним, что ковариация C_{ij} между двумя случайными величинами $f(x_i)$ и $f(x_o)$, имеющими одинаковую дисперсию σ^2, задается следующим образом [Simon, 2006, глава 2]:

$$C_{ij} = \rho_{ij} \sigma^2. \quad (21.11)$$

Следовательно, уравнение (21.10) можно записать в виде:

$$\text{PDF}(f(x)) = \frac{1}{(2\pi)^{M/2} \sigma^M \mid R \mid^{1/2}} \exp\left[-\frac{(f(x) - \mu 1_M)^T R^{-1} (f(x) - \mu 1_M)}{2\sigma^2} \right], \quad (21.12)$$

где R – это корреляционная матрица; элемент в i-й строке и j-м столбце R равен ρ_{ij}.

С учетом множества кандидатных решений $\{x_i\}$ и вектора оцениваний приспособленности $f(x)$ мы можем найти значения μ и σ, которые обеспечивают лучшее соответствие между измеренным значением $f(x)$ и принятой параметрической формой $f(x)$. Уравнение (21.12) дает функ-

цию плотности вероятности PDF от $f(x)$, которая пропорциональна вероятности получения конкретного $f(x)$ с учетом его допущенной случайной природы. Следовательно, чтобы найти лучшее соответствие между допущенной параметрической формой $f(x)$ и измеренными значениями $f(x)$, нам требуется найти значения μ и σ, максимизирующие PDF($f(x)$) в уравнении (21.12). Сначала рассмотрим максимизацию по μ. Мы можем максимизировать PDF($f(x)$) по μ, минимизируя отрицательный экспоненциальный аргумент по μ. Взяв частную производную отрицательного экспоненциального аргумента по μ и приравняв ее к нулю, в результате получим:

$$\frac{\partial(f(x)-\mu 1_M)^T R^{-1}(f(x)-\mu 1_M)}{\partial\mu}=0, \tag{21.13}$$

где мы игнорируем член $2\sigma^2$ в знаменателе уравнения (21.12), так как он не зависит от μ. Решение уравнения (21.13) дает

$$-2f^T(x)R^{-1}1_M+2\mu 1_M^T R^{-1}1_M=0$$

$$\mu=\frac{f^T(x)R^{-1}1_M}{1_M^T R^{-1}1_M}. \tag{21.14}$$

Взяв частную производную уравнения (21.12) по σ^2, получаем:

$$\frac{\partial \mathrm{PDF}(f(x))}{\partial\sigma^2}=\frac{1}{2\sigma^2}\left[\frac{(f(x)-\mu 1_M)^T R^{-1}(f(x)-\mu 1_M)}{\sigma^2}-M\right]\mathrm{PDF}(f(x)). \tag{21.15}$$

Приравнивание вышеуказанного уравнения к нулю дает

$$\sigma^2=\frac{(f(x)-\mu 1_M)^T R^{-1}(f(x)-\mu 1_M)}{M}. \tag{21.16}$$

Уравнения (21.14) и (21.16) дают оптимальные значения μ и σ для аппроксимации функции приспособленности с помощью метода DACE.

Теперь рассмотрим наши M оцениваний функции приспособленности $f(x)$ кандидатных решений $\{x_i\}$. Предположим, что функции приспособленности коррелируют, как показано в уравнении (21.8). Предположим, что мы получаем еще одно оценивание функции приспособленности $f(x^*)$ для еще одного кандидатного решения x^*. Мы увеличиваем вектор оцениваний функции приспособленности, получив $(M+1)$-элементный вектор:

$$\tilde{f}(x) = [\, f^T(x) \quad f(x^*)\,]^T. \tag{21.17}$$

Мы можем написать новую корреляционную матрицу

$$\tilde{R} = \begin{bmatrix} R & r \\ r^T & 1 \end{bmatrix}, \tag{21.18}$$

где r – это вектор корреляций между оцениваниями функции приспособленности $f(x)$ и дополнительным оцениванием функции приспособленности $f(x^*)$. Мы хотим максимизировать функцию плотности PDF уравнения (21.12) по $f(x^*)$, в результате получив оценку формы уравнения (21.6), которая ближе всего соответствует новым данным $f(x^*)$. Такая оценка называется оценкой максимального правдоподобия. Мы можем максимизировать PDF уравнения (21.12) путем максимизации

$$(\tilde{f}(x) - \mu 1_M)^T \tilde{R}^{-1} (\tilde{f}(x) - \mu 1_M) = \begin{bmatrix} f(x) - \mu 1_M \\ f(x^*) - \mu \end{bmatrix}^T \begin{bmatrix} R & r \\ r^T & 1 \end{bmatrix}^{-1} \begin{bmatrix} f(x) - \mu 1_M \\ f(x^*) - \mu \end{bmatrix} \tag{21.19}$$

по $f(x^*)$. Можем использовать результаты из деривации леммы инверсии матрицы [Simon, 2006, глава 1], показав, что

$$\tilde{R}^{-1} = \begin{bmatrix} R & r \\ r^T & 1 \end{bmatrix}^{-1} = \frac{1}{1 - r^T R^{-1} r} \begin{bmatrix} R^{-1} + R^{-1} r r^T R^{-1} & -R^{-1} r \\ -r^T R^{-1} & 1 \end{bmatrix}. \tag{21.20}$$

Подставив это выражение в уравнение (21.19), получаем:

$$\frac{(f(x^*) - \mu)^2 - 2 r^T R^{-1} (f(x) - \mu 1_M)(f(x^*) - \mu)}{1 - r^T R^{-1} r} + \text{члены без } f(x^*). \tag{21.21}$$

Поскольку мы хотим максимизировать это выражение по $f(x^*)$, мы берем производную по $f(x^*)$ и приравниваем ее к нулю, получив:

$$\frac{2(f(x^*) - \mu) - 2 r^T R^{-1} (f(x) - \mu 1_M)}{1 - r^T R^{-1} r} = 0. \tag{21.22}$$

Решение для $f(x^*)$ дает

$$f(x^*) = \mu + r^T R^{-1} (f(x) - \mu 1_M). \tag{21.23}$$

Это уравнение показывает, как мы можем применить существующую модель для аппроксимирования значения приспособленности новой точки x^*. Среднеквадратическая ошибка аппроксимации введена в публикации [Jones и соавт., 1998] как

$$s^2(x^*) = \sigma^2 \left[1 - r^T R^{-1} r + \frac{(1 - 1_M^T R^{-1} r)^2}{1_M^T R^{-1} 1_M} \right]. \tag{21.24}$$

Пара строк алгебраических выражений легко показывает, что $s(x) = 0$ в выборочных точках данных (см. задачу 21.3) [Jones и соавт., 1998].

Мы можем использовать среднеквадратическую ошибку с целью определения подходящих выборочных точек для дополнительных оцениваний приспособленности. В поисковом пространстве есть два участка, где мы, возможно, будем особенно заинтересованы получить дополнительные оценивания приспособленности. Во-первых, нас вполне может заинтересовать выборочный отбор (то есть вычисление $f(x)$) вблизи минимума аппроксимации приспособленности в надежде найти более подходящее решение оптимизационной задачи. Во-вторых, нас вполне может особенно заинтересовать выборочный отбор в участках, где $s(x)$ является большим, потому что в этих участках поискового пространства у нас много неопределенности. Эта идея показана на рис. 21.4. Выборочный отбор дополнительных значений приспособленности вблизи минимума аппроксимации является эксплуатационной стратегией, поскольку она предусматривает поиск в участках, где у нас уже есть хорошие результаты. Выборочный отбор дополнительных значений приспособленности в участках с высокой среднеквадратической ошибкой является разведывательной стратегией, так как она предусматривает поиск в участках, где у нас мало информации о функции приспособленности.

Взятие выборочных точек с целью повышения точности моделирования называется активным самообучением. Активное самообучение обычно означает, что мы берем выборочные точки в обучающемся алгоритме для оптимизации некой стоимостной функции. В описанном выше сценарии DACE мы можем взять выборочные точки для уменьшения максимальной среднеквадратической ошибки. Методы тренировки нейронных сетей часто включают в свой состав активное самообучение [Settles, 2010].

Мы можем получить дополнительную точность функции приспособленности, оценив оптимальные значения $\{p_k\}$ и $\{\theta_k\}$ из уравнения (21.8). Мы делаем это, подставляя уравнения (21.14) и (21.16) в уравнение (21.12), в результате получая выражение функции плотности PDF($f(x)$), которое зависит только от $\{p_k\}$ и $\{\theta_k\}$. Затем мы максимизируем это выражение по $\{p_k\}$ и $\{\theta_k\}$, в результате получая оценку оптимальных коэффициентов корреляции ρ_{ij}, которые являются элементами R. Затем мы используем это значение R в уравнениях (21.14) и (21.16) для получения лучших оценок для μ и σ. Всякий раз, когда мы получаем новое кандидатное решение x^*, мы используем уравнение (21.23) для аппроксимации его приспособленности.

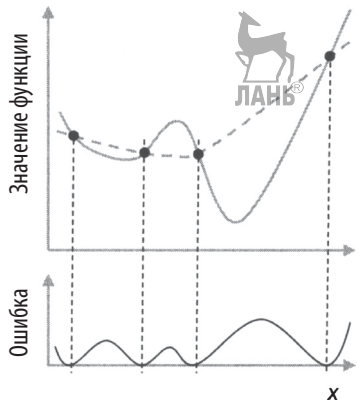


Рис. 21.4. Сплошная линия на верхнем рисунке представляет функцию, и пунктирная линия представляет ее аппроксимацию. Кривая на нижнем рисунке представляет среднеквадратическую ошибку уравнения (21.24). У нас вполне может возникнуть соблазн улучшить нашу аппроксимацию путем отбора дополнительных значений функций вблизи минимума аппроксимации, но в этом случае мы должны отбирать там, где ошибка является самой высокой, потому что именно там происходит минимум функции

■ Пример 21.1

В данном примере мы используем метод DACE для оценивания двумерной эталонной функции Бранина

$$f(x) = (x(2) - (5/(4\pi^2))\, x(1)^2 + 5x(1)/\pi - 6)^2 + 10(1 - 1/(8\pi))\cos(x(1)) + 10, \qquad (21.25)$$

где $x(1)$ и $x(2)$ – это два компонента кандидатного решения ($n = 2$). Область функции равна $x(1) \in [-5, 10]$ и $x(2) \in [0, 15]$. Сначала мы должны решить, какие выборочные точки использовать. Здесь мы произвольно решаем использовать 25 выборочных точек, равномерно распределенных в двумерной поисковой области ($M = 25$). Затем используем функцию `fmincon` языка MATLAB для максимизации уравнения (21.12) по $\{p_k\}$ и $\{\theta_k\}$, которая дает

$$\begin{aligned} p_1 &= 1.6194, & p_2 &= 2 \\ \theta_1 &= 0.020816, & \theta_2 &= 0.00018011 \end{aligned} \qquad (21.26)$$

Далее мы используем уравнение (21.23) для аппроксимации $f(x)$ на мелкой решетке. Рисунок 21.5 показывает результаты. Мы видим, что аппроксимация довольно хорошо захватывает принципиальную форму функции Бранина, и самое главное, она захватывает мультимодальную характеристику данной функции.

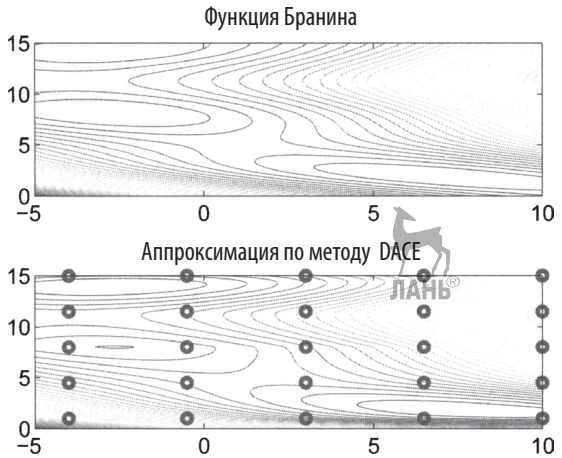


Рис. 21.5. Результаты примера 21.1. Верхний рисунок показывает контурный график функции Бранина. На нижнем рисунке показаны 25 равномерно распределенных выборочных точек и аппроксимация функции Бранина на основе метода DACE

■

В примере 21.1 используется равномерный выборочный отбор, однако другие методы отбора вполне могут дать более качественные результаты аппроксимации. Одним из популярных методов является отбор на основе латинского гиперкуба. Этот метод делит область на интервалы в каждой размерности, а затем размещает выборочные точки так, что каждый интервал в каждой размерности содержит только одну выборочную точку. Такой подход к выборочному отбору может иногда лучше захватывать непредсказуемый, неизвестный характер функции, чем равномерный выборочный отбор. На рис. 21.6 показана разница между равномерным выборочным отбором и отбором на основе латинского гиперкуба.

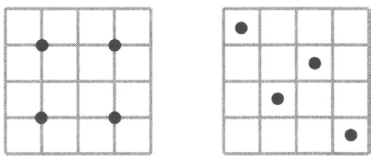


Рис. 21.6. На рисунке слева показан равномерный выборочный отбор четырех точек в поисковой области. На рисунке справа показан выборочный отбор на основе латинского гиперкуба. Обратите внимание, что в каждой строке есть только одна точка, и в каждом столбце есть только одна точка. Такую характеристику имеет несколько разных расположений точек, и поэтому выборочный отбор на основе латинского гиперкуба не уникален

■ Пример 21.2

Мы используем метод DACE с выборочным отбором на основе латинского гиперкуба для оценивания двумерной функции Бранина в примере 21.1. Мы произвольно решаем использовать 21 выборочную точку ($M = 21$). Затем применяем функцию MATLAB `fmincon` для максимизации уравнения (21.12) по $\{p_k\}$ и $\{\theta_k\}$, получив

$$\begin{aligned} p_1 &= 1, & p_2 &= 2 \\ \theta_1 &= 0.028227, & \theta_2 &= 0.0013912 \end{aligned} \tag{21.27}$$

Далее мы применяем уравнение (21.23) для аппроксимации $f(x)$ на мелкой решетке. На рис. 21.7 показаны результаты. Сравнение рис. 21.5 и 21.7 показывает, что выборочный отор на основе латинского гиперкуба дает более качественную аппроксимацию, чем равномерный выборочный отбор. По сути дела, среднеквадратическая аппроксимационная ошибка (точнее, средний корень квадратный из нее) аппроксимации по методу DACE в примере 21.1, в котором используется равномерный выборочный отбор, составляет 24.9, в то время как среднеквадратическая аппроксимация с выборочным отбором на основе латинского гиперкуба – 14.3. Даже при меньшем числе выборочных точек выборочный отбор на основе латинского гиперкуба дает аппроксимационную ошибку, которая почти на 50 % лучше, чем при равномерном выборочном отборе. Мы видим, что метод выборочного отбора может оказать существенное влияние на результаты аппроксимации по методу DACE. Кроме того, метод, который мы используем для максимизации уравнения (21.12) с целью нахождения оптимальных значений $\{p_k\}$ и $\{\theta_k\}$, может существенно повлиять на метод DACE, хотя мы не приводим здесь никаких примеров.

■

Метод DACE является обобщением алгоритма кригинга. Алгоритм кригинга – это аппроксимационный метод, названный в честь Даниэля Герхардуса Криге (Daniel Gerhardus Krige), который разработал его для геологических применений [Krige, 1951]. Хотя он назван в честь человека, слово «кригинг» обычно пишется со строчной буквы. Кригинг совпадает с методом DACE, за исключением того, что в кригинге уравнение (21.8) заменяется на

$$\begin{aligned} d_{ij} &= \sum_{k=1}^{n} \theta_k \left| x_i(k) - x_j(k) \right|^2 \\ \rho_{ij} &= \mathrm{Corr}\left(f(x_i), f(x_j) \right) = \exp(-d_{ij}) \end{aligned} \tag{21.28}$$

То есть p_k в уравнении (21.8) заменяется константой 2 [Chung и соавт., 2003).

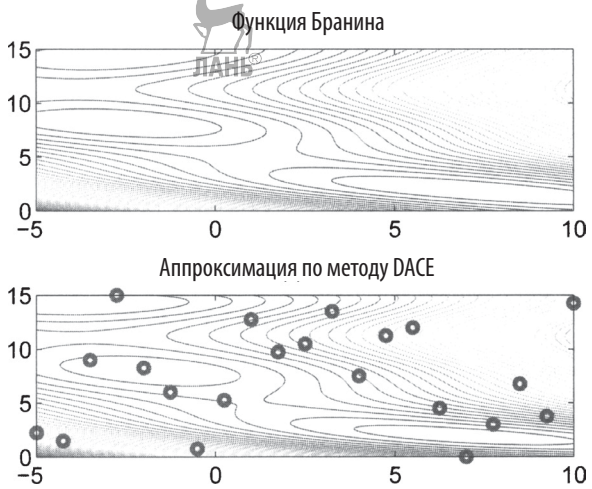


Рис. 21.7. Результаты примера 21.2. Верхний рисунок показывает контурный график функции Бранина. На нижнем рисунке показаны 21 выборочная точка, полученная с использованием выборочного отбора на основе латинского гиперкуба, и аппроксимация функции Бранина на основе метода DACE

21.1.2. *Аппроксимирование трансформированных функций*

Иногда методы аппкроксимации приспособленности не работают хорошо. Например, уравнение (21.23) метода DACE требует обратной матрицы, которая может не существовать. Если обратной матрицы не существует, то мы можем применить псевдообратную матрицу [Golan, 2007]. Однако главное здесь заключается в том, что базисные функции, которые мы используем для аппроксимирования функции приспособленности, могут не подходить для формы этой функции приспособленности. Например, если для аппроксимирования функции с неправильным поведением и острыми краями мы используем ряд Фурье, то мы не можем ожидать хорошей аппроксимационной результативности во всех точках области функции. В таких случаях мы можем трансформировать исходную функцию приспособленности, а затем найти аппроксимацию трансформированной функции. Например, предположим, что мы оценили функцию приспособленности в M выборочных точках $\{x_i\}$. Если аппроксимация не работает хорошо, то можем попробовать транс-

формировать образцы функции приспособленности $f(x_i)$, взяв их натуральный логарифм:

$$L(x_i) = \log(f(x_i)). \tag{21.29}$$

Затем мы находим аппроксимацию для $L(x)$ с помощью выборочных точек $L(x_i)$. Обозначим аппроксимацию как $\hat{L}(x)$. Потом инвертируем трансформацию, чтобы найти аппроксимацию для исходной функции:

$$\hat{f}(x) = \exp(\hat{L}(x)). \tag{21.30}$$

■ Пример 21.3

Мы применяем метод DACE раздела 21.1.1.2 на функции Гольдштейна-Прайса [Floudas и Pardalos, 1990]:

$$\begin{aligned} a &= 1 + (x(1) + x(2) + 1)^2 \times \\ &\quad (19 - 14x(1) + 3x(1)2 - 14x(2) + 6x(1)\mathrm{x}(2) + 3x(2)^2) \\ \mathrm{b} &= 30 + (2\mathrm{x}(1) - 3x(2))^2 \times \\ &\quad (18 - 32x(1) + 12x(1)^2 + 48x(2) - 36x(1)x(2) + 27x(2)^2) \\ f(x) &= ab \end{aligned} \tag{21.31}$$

где $x(1)$ и $x(2)$ находятся в области $[-2, 2]$. Эта функция очень плоская вблизи минимума, что встречается при $x^*(1) = 0$ и $x^*(2) = 1$. Минимум функции равен $f^* = 3$. Плоский участок отвергает аппроксимацию DACE, потому что в плоском участке выборочные точки сильно коррелированы, а значит, некоторые столбцы в R состоят почти полностью из них, следовательно, R почти сингулярна. Мы вполне могли бы преодолеть эту проблему, используя псевдообратную матрицу R вместо регулярной обратной матрицы R. Вместо этого мы в данном примере решаем взять натуральный логарифм от выборочных точек, как показано в уравнении (21.29). Это значительно изменяет форму функции; он разводит порознь малые, но близкие друг к другу значения $f(x)$ и сводит вместе большие значения $f(x)$, тем самым сжимая общий диапазон функции, при этом отделяя похожие значения функции. Затем мы применяем метод DACE для аппроксимирования $L(x)$, а потом вычисляем аппроксимацию исходной функции, как показано в уравнении (21.30). Эта простая модификация процесса аппроксимирования дает нам приличную аппроксимацию, как показано на рис. 21.8. Аппроксимация не выглядит великолепной, но, по крайней мере, она приводит к обратимой матрице R, а также захватывает большой плоский участок в середине области.

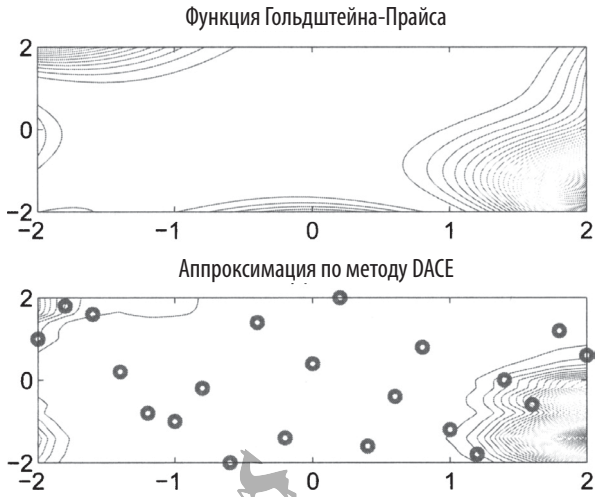


Рис. 21.8. Результаты примера 21.3. На верхнем рисунке показан контурный график функции Гольдштейна-Прайса. На нижнем рисунке показаны 21 выборочная точка, полученная с помощью выборочного отбора на основе латинского гиперкуба, и аппроксимация функции на основе метода DACE

■

21.1.3. *Как применять аппроксимации приспособленности в эволюционных алгоритмах*

Имея в распоряжении алгоритм аппроксимации приспособленности, у нас есть несколько вариантов его использования в эволюционном алгоритме [Jin, 2005]. Во-первых, мы можем просто заменить фиксированную долю R оцениваний приспособленности аппроксимациями приспособленности. Исходя из того, что в эволюционном алгоритме оценивание функции приспособленности является доминирующим вычислительным усилием, это позволит уменьшить вычислительные усилия с E до $(1 - r)E$. Однако с этой идеей не следует заходить слишком далеко. Если мы заменим слишком много оцениваний приспособленности на аппроксимации, то эволюционному алгоритму потребуется больше времени для схождения, и наша попытка сэкономить на вычислениях вполне может оказаться контрпродуктивной. В самом крайнем случае, если мы заменим все оценивания приспособленности на аппроксимации, то $r = 1$. В этом случае вычислительные усилия действительно могут быть примерно равны нулю, но эволюционный алгоритм никогда не сойдется к полезному результату.

Еще одним вариантом является создание дополнительных детей в каждом поколении и использование их приближенных значений приспособленности, для того чтобы решить, какие из них сохранять для следующего поколения. Мы называем эту идею эволюционным контролем, или модельным менеджментом. Если мы оцениваем отдельных особей с помощью точной функции приспособленности и других особей с помощью приближенной функции приспособленности, то называем это индивидуальным эволюционным контролем [Shi и Rasheed, 2010]. Для того чтобы решать, какие особи оценивать точно, а какие приближенно, мы можем применять различные методы. Например, можем принимать случайное решение, какой тип оценивания использовать по каждой особи. Каждое поколение мы также можем использовать точные оценивания приспособленности только для особей с хорошей приближенной приспособленностью. Особи, чью приспособленность мы оцениваем точной функцией приспособленности, называются контролируемыми особями.

Если мы оцениваем всех особей в определенном поколении с помощью точной функции приспособленности, а всех особей в определенных поколениях с помощью примерной функции приспособленности, то имеем поколенческий эволюционный контроль. Для того чтобы решать, какие поколения оценивать точно, а какие приближенно, мы можем применять различные методы. Например, можем детерминистически решать оценивать с помощью точной функции приспособленности только каждое k-е поколение, где k – это определяемый пользователем контрольный параметр. Кроме того, можем произвольно решать, какой вид оценивания использовать в каждом поколении. Мы можем также применять приближенные оценивания приспособленности до тех пор, пока не обнаружим схождение (например, лучшая особь не улучшилась определенное число поколений подряд, или стандартное отклонение популяции упало ниже некоторого порога), а затем оценивать следующее поколение с помощью точной функции приспособленности. После этого мы вернемся к приближенным оцениваниям приспособленности. Поколения, в которых мы оцениваем всех особей с помощью точной функции приспособленности, называются контролируемыми поколениями.

Исследователи предложили несколько разновидностей эволюционного контроля, включая гибридный эвалюционный алгоритм на основе динамической приближенной приспособленности (dynamic approximate fitness based hybrid EA, DAFHEA) [Bhattacharya, 2008], который проиллюстрирован на рис. 21.9.

N = размер популяции
$N_c \leftarrow 5N$
Создать N_c случайных особей
Оценить приспособленность N_c особей
Применить N_c значений приспособленности для создания аппроксимаций $\hat{f}(\cdot)$
Оставить лучших N особей для исходной популяции
While not (критерий останова)
    Применить эволюционный алгоритм для создания N_c детей
    Применить $\hat{f}(\cdot)$ для аппроксимирования приспособленности детей
    Сохранить N лучших детей (согласно $\hat{f}(\cdot)$) для следующего поколения
    If настало время для следующей аппроксимации **then**
        Вычислить приспособленность $f(x_i)$ для каждой особи x_i
        Применить значения приспособленности для обновления $\hat{f}(\cdot)$
    End if
Next поколение

Рис. 21.9. Схематичное описание гибридного эвалюционного алгоритма на основе динамической приближенной приспособленности (DAFHEA)

В алгоритме DAFHEA обычно реализуется элитарность, хотя на рис. 21.9 это не показано. Мы можем попробовать применить несколько вариаций рис. 21.9. Например, можем попробовать значения для N_c, которые отличаются от $5N$. Мы можем попробовать разные алгоритмы аппроксимации приспособленности, хотя в оригинальном алгоритме DAFHEA использовались опорно-векторные машины. Можем попробовать разные методы, помогающие определить время для новой аппроксимации. Несколькими общими критериями для этого решения является фиксированное число поколений или создание новой аппроксимации, когда эволюционный алгоритм удовлетворяет некоторым критериям сходимости.

Кроме того, для принятия решения, когда следует генерировать новую аппроксимацию, мы можем использовать доверительные участки [Betts, 2009]. Методы доверительных участков основаны на сравнении приближенных значений приспособленности с фактическими значениями. Если приближенные значения приспособленности близки к фактическим значениям, то аппроксимация хороша, поэтому мы можем увеличить время между генерированием новых аппроксимаций. Однако если приближенные значения приспособленности не близки к фактическим значениям, то аппроксимация слабая, поэтому нам нужно уменьшить время между генерированием новых аппроксимаций. Предположим, что G – это число поколений между вычислениями новой

аппроксимации приспособленности. Каждое поколение у нас есть N_c детей на рис. 21.9. Мы вычисляем точные значения приспособленности N_e детей и сравниваем их приближенные значения приспособленности с их точными значениями приспособленности. Если среднеквадратическая разница превышает заданный порог T^+, то мы уменьшаем G. Если она опускается ниже заданного порога T^-, то мы увеличиваем G. Мы используем $T^+ > T^-$ (строгое неравенство), чтобы предотвратить вибрирование в G. Нам нужно отрегулировать значения T^+ и T^-, для того чтобы получить хороший компромисс между уменьшенным вычислительным усилием (большим G) и достаточно точными аппроксимациями приспособленности (малым G). В этом подходе нам также нужно отрегулировать N_e.

Рисунок 21.9 носит довольно общий характер. Чтобы быть конкретнее, мы можем использовать аппроксимацию приспособленности, для того чтобы решать, какие N из N_c исходных особей оставлять в исходной популяции. Такой подход называется информированной инициализацией, и в этом случае мы, возможно, захотим основывать аппроксимацию приспособленности на чем-то другом, помимо точных оцениваний приспособленности N_c исходных особей. Например, мы можем сконструировать алгоритм аппроксимации приспособленности в автономном режиме до начала исполнения эволюционного алгоритма.

Мы можем также использовать аппроксимации приспособленности на рис. 21.9 только для скрещивания (если нашим базовым эволюционным алгоритмом является генетический алгоритм), или только для миграции (если нашим базовым эволюционным алгоритмом является алгоритм биогеографической оптимизации), или только для алгоритма рекомбинации, который зависит от применяемого нами эволюционного алгоритма. В этом случае мы проводим рекомбинацию (скрещивание, либо миграцию, либо какой-либо другой специфичный для эволюционного алгоритма метод рекомбинации), для того чтобы создать ряд детей (более N), но мы сохраняем только лучших N детей, основываясь на их приближенных значениях приспособленности. Это называется, в частности, информированным скрещиванием либо информированной миграцией.

Мы также можем использовать приближение приспособленности на рис. 21.9 только для мутации в конкретном применяемом нами эволюционном алгоритме. В этом случае мы создаем ряд (более N) мутированных версий N детей, но сохраняем только лучшие N версий, основываясь на их приближенных значениях приспособленности. Это называется информированной мутацией и аналогично идее в разде-

ле 16.4 реализации оппозиционного самообучения только для наименее приспособленных особей в популяции.

Мы можем попробовать различные алгоритмы обновления $\hat{f}(\cdot)$ в конце алгоритма DAFHEA в зависимости от того, сколько информации мы хотим сохранить от предыдущей аппроксимации. Это может зависеть от того, насколько динамичной мы считаем функцию приспособленности. Мы можем использовать несколько моделей аппроксимации приспособленности и переключаться между моделями в зависимости от того, насколько они точны. Это похоже на мультимодельную аппроксимацию, как будет описано в следующем разделе. Однако в следующем разделе мы сосредоточимся на намеренном объединении в нашем эволюционном алгоритме моделей с низкой и высокой точностью. Здесь мы можем попытаться применить несколько моделей в попытке найти модель с наивысшей точностью.

Наконец, мы можем немного модифицировать рис. 21.9, применив аппроксимацию приспособленности только для того, чтобы решать, каких детей оставлять для следующего поколения. Этот подход называется подходом на основе информированного оператора (informed operator approach) [Rasheed и Hirsh, 2000], который проиллюстрирован на рис. 21.10.

N = размер популяции
$N_c \leftarrow 5N$
Создать N_c случайных особей.
Оценить приспособленность N_c особей.
Применить N_c значений приспособленности для создания аппроксимации приспособленности $\hat{f}(\cdot)$
Оставить лучших N особей для исходной популяции.
While not (критерий останова)
    Применить эволюционный алгоритм для создания N_c детей.
    Применить $\hat{f}(\cdot)$ для аппроксимации приспособленности детей.
    Сохранить N лучших детей (согласно $\hat{f}(\cdot)$) для следующего поколения.
    Вычислить приспособленность $f(x_i)$ для каждого ребенка x_i
    Применить значения приспособленности для обновления $\hat{f}(\cdot)$
Next поколение

Рис. 21.10. Схематичное описание алгоритма информированного оператора

21.1.4. *Множественные модели*

Мы можем выполнять приближенные оценивания функции стоимости в ранних поколениях и более точные оценивания в последующих по-

колениях. Это похоже на идею использования подмножества тестовых случаев для оценивания стоимостной функции, о которой мы говорили выше, но здесь мы используем то же число тестовых случаев при выполнении менее точных оцениваний в течение ранних эволюционно-алгоритмических поколений. В качестве конкретного примера предположим, что оценивание стоимостной функции подразумевает решение уравнения Риккати:

$$P = FPF^T - FPH^T(HPH^T + R)^{-1}HPF^T + Q. \quad (21.32)$$

Этот тип уравнения часто возникает в задачах управления и оценивания и поэтому, вероятно, будет появляться и в эволюционных алгоритмах, которые пытаются оптимизировать контроллер или эстиматор [Simon, 2006]. С учетом известных квадратных матриц F, Q и R и, возможно, неквадратной матрицы H нам нужно решить уравнение для квадратной матрицы P. Результативность алгоритма управления или оценивания часто пропорциональна следу P. Решатель уравнения Риккати может быть вычислительно дорогостоящим, но для получения оценок решения мы можем применять приближенные методы [Emre и Knowles, 1987]. В ранних эволюционно-алгоритмических поколениях для решения уравнения (21.32) мы можем использовать грубые аппроксимации, а в последующих поколениях – более точные.

Рисунок 21.11 иллюстрирует этот процесс. Эволюционный алгоритм с низкоточной моделью приспособленности работает в течение T_1 поколений. После T_1 поколений эволюционно-алгоритмическая популяция используется для инициализации следующего эволюционного алгоритма, в котором используется среднеточная модель, работающая для поколений T_2. После этого эволюционный алгоритм завершается, и его окончательная популяция используется для инициализации завершающего эволюционного алгоритма, в котором используется высокоточная модель, работающая для поколений T_3. Это можно расширить на столько модельных уровней точности, на сколько потребуется. При таком подходе мы должны проявлять особую осторожность в том, чтобы каждый эволюционный алгоритм инициализировался с многообразной популяцией. Мы можем добиться этого, приняв меры к тому, чтобы каждая эволюционно-алгоритмическая популяция была достаточно многообразной, прежде чем она перейдет на следующий самый высокий уровень аппроксимации функции приспособленности. Мы также можем сделать это путем мигрирования только нескольких особей из более низкоточного эволюционного алгоритма в более высокоточный эволюционный алгоритм и инициализации остальной популяции бо-

лее высокоточного эволюционного алгоритма, так чтобы обеспечивалось многообразие.

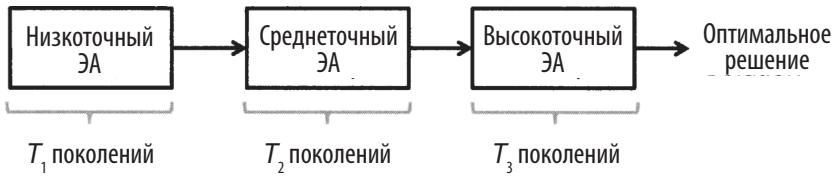


Рис. 21.11. Мультимодельная аппроксимация приспособленности. Эволюционные алгоритмы с разными уровнями аппроксимации приспособленности выполняются последовательно

Более тесно интегрированный подход к мультимодельной оптимизации параллельно выполняет эволюционные алгоритмы с разными уровнями аппроксимаций функций приспособленности. При таком подходе особи мигрируют между параллельными эволюционными алгоритмами с заданной частотой [Sefrioui и Périaux, 2000]. Данный подход, именуемый иерархическими эволюционными вычислениями, включает в себя несколько разных вариантов. Во-первых, мы можем мигрировать особей из эволюционных алгоритмов с более высокоточных аппроксимаций в эволюционные алгоритмы с более низкоточными аппроксимациями, как показано на рис. 21.12. Во-вторых, мы можем мигрировать особей туда и обратно между эволюционными алгоритмами с аналогичными уровнями приспособленности, как показано на рис. 21.13. Иерархические эволюционные алгоритмы могут использоваться с любым числом уровней точности моделей, которое потребуется.

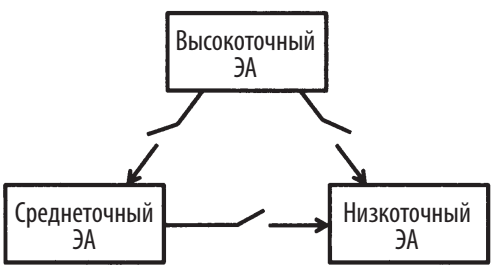


Рис. 21.12. Иерархический эволюционный алгоритм. Данная модель мигрирует особей из эволюционных алгоритмов с высокоточными аппроксимациями функции приспособленности в эволюционные алгоритмы с низкоточными аппроксимациями. Миграция происходит на определяемых пользователем частотах, как обозначено переключателями на рисунке

Рис. 21.13. Иерархический эволюционный алгоритм. Данная модель мигрирует особей между эволюционными алгоритмами с аналогичными уровнями аппроксимации функции приспособленности. Миграция происходит на определяемых пользователем частотах, как обозначено переключателями на рисунке

Еще один подход к использованию множественных моделей заключается в генерировании множественных моделей, которые могут использоваться в различных комбинациях для любой особи x. Например, предположим, что мы оценили приспособленность M особей $\{x_i\}$. Мы можем применить алгоритм кластеризации, для того чтобы разделить $\{x_i\}$ на C кластеров. Затем мы можем создать модель аппроксимации приспособленности для каждого из C кластеров. В результате получим $\hat{f}_k(x)$ для $k \in [1, C]$. Теперь, когда нам понадобится аппроксимировать приспособленность особи x, мы можем принять один из нескольких подходов. Например, мы можем аппроксимировать $f(x)$ как $\hat{f}_k(x)$, где индекс k – это кластер, ближайший к x [Chung и Alonso, 2004]. В качестве альтернативы мы можем аппроксимировать $f(x)$ как взвешенную комбинацию $\hat{f}_k(x)$, где веса в сумме составляют 1 и являются функциями удаленности x от каждого кластера.

21.1.5. *Переподгонка*

В некоторых подходах, связанных с аппроксимацией приспособленности, переподгонка может являться проблемой. Проектировщик эволюционного алгоритма всегда должен быть осторожен в использовании методов аппроксимации приспособленности. Переподгонка часто является проблемой в нейронных сетях, если инженер не делает намеренных попыток ее избежать [Krogh, 2008]. На рис. 21.14 показан пример переподгонки в простой задаче подгонки кривой под множество точек данных. Хотя полином более высокой степени на рисунке укладывается в данные лучше, чем полиномы более низких степеней, полином более высоких степеней не очень хорошо обобщает. Мы могли бы сказать, что он запоминает точки данных, не обеспечивая при этом хорошую результативность между точками данных. Часто, для того чтобы получить более высокую результативность обобщения, нам нужно принять более высокую ошибку подгонки в точках данных.

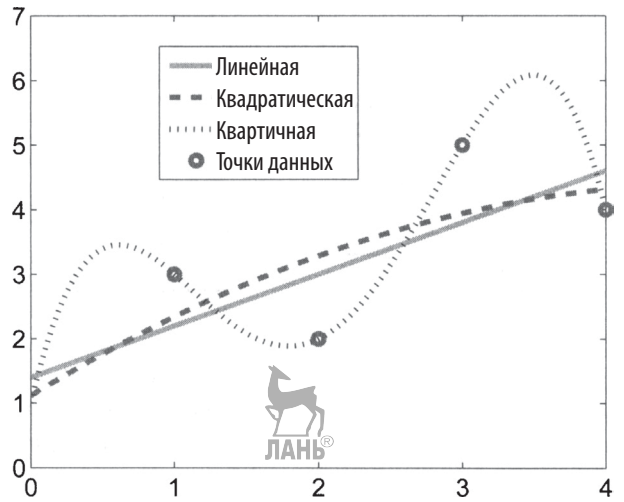


Рис. 21.14. На данном рисунке показан пример переподгонки. Линейная функция и квадратичная функция, похоже, хорошо укладываются в данные. Квадратичная функция идеально укладывается в данные, но включает большие колебания, что указывает на то, что она плохо обобщает

Переподгонку можно смягчить ансамблевыми методами. Ансамбль представляет собой множество индивидуально натренированных аппроксимаций приспособленности, предсказания которых объединяются во время оценивания значений приспособленности для ранее не встречавшихся точек в поисковом пространстве [Opitz и Maclin, 1999], [Lim и соавт., 2010].

21.1.6. *Оценивание аппроксимационных методов*

После того как у нас есть аппроксимация функции, прежде чем полагаться на нее в эволюционном алгоритме, нам нужно проверить, что она показывает хорошие результаты. Мы всегда должны начинать с проверки значений аппроксимации в выборочных точках (то есть точках, которые мы использовали для создания аппроксимации). Многие аппроксимационные методы автоматически выдают функции $\hat{f}_k(x)$, которые в точности совпадают с истинной функцией $f(x)$ в M выборочных точках $\{x_i\}$, то есть $\hat{f}_k(x) = f(x)$ для $x \in \{x_i\}$. Тем не менее мы все равно должны проверить это, чтобы убедиться, что правильно реализовали аппроксимационный алгоритм.

Одним из методов оценивания точности аппроксимационного метода является выбор нескольких дополнительных выборочных точек,

помимо тех, которые мы использовали для построения нашей аппроксимации. Скажем, мы выбираем Q дополнительных выборочных точек $\{x_i\}$, именуемых тестовыми точками, где $i \in [M + 1, M + Q]$. Затем мы оцениваем функцию и аппроксимацию в тестовых точках, для того чтобы увидеть, насколько хорошо аппроксимация работает. Мы измеряем среднеквадратическую (RMS) ошибку аппроксимации как

$$E^2_{\text{RMS}} = \frac{1}{Q} \sum_{i=M+1}^{M+Q} (f(x_i) - \hat{f}(x_i))^2 \tag{21.33}$$

и измеряем ошибку аппроксимации для худшего случая как

$$E_{\max} = \max_{i \in [M+1, M+Q]} |f(x_i) - \hat{f}(x_i)|. \tag{21.34}$$

В зависимости от наших приоритетов и нашей конкретной задачи мы можем для оценивания качества аппроксимации использовать любой из этих метрических показателей.

Еще один метод, именуемый перекрестной проверкой, или ротационным оцениванием [Geisser, 1993], позволяет оценивать качество аппроксимации без использования дополнительных выборочных точек. Как и выше, предположим, что у нас есть M выборочных точек и M значений функции $f(x_i)$. В перекрестной проверке мы вычисляем аппроксимацию, используя все выборочные точки, кроме k-й, и называем эту аппроксимацию $\hat{f}_k(x)$. Таким образом, мы вычисляем M аппроксимаций, каждая из которых не учитывает одну выборочную точку. В результате получаем M аппроксимаций $\hat{f}_k(x)$ для $k \in [1, M]$, где каждая аппроксимация использует уникальное множество $(M - 1)$ выборочных точек. Затем мы оцениваем каждую аппроксимацию в выборочной точке, которую не использовали при ее построении. То есть мы оцениваем $\hat{f}_k(x_k)$ для $k \in [1, M]$. Как и выше, мы измеряем среднеквадратическую ошибку (RMS) или ошибку аппроксимации для худшего случая как

$$\begin{aligned} E^2_{\text{RMS}} &= \frac{1}{M} \sum_{i=1}^{M} (f(x_i) - \hat{f}_i(x_i))^2 \\ E_{\max} &= \max_{i \in [1, M]} |f(x_i) - \hat{f}_i(x_i)| \end{aligned} \tag{21.35}$$

После того как мы применили перекрестную проверку, чтобы убедиться в правильности нашего подхода к аппроксимации, мы используем все M точек данных, для того чтобы найти аппроксимацию функции $\hat{f}(x)$, которую мы будем использовать в нашем эволюционном алгоритме.

Для оценивания качества метода аппроксимации приспособленности мы можем использовать другие метрические показатели, если у нас есть ресурсы для сравнения приближенных значений приспособленности с точными значениями приспособленности. Эти метрические показатели включают сравнение числа особей, отобранных для рекомбинации при использовании истинных значений приспособленности, с приближенными значениями приспособленности; ранги особей, которые неправильно отобраны при использовании приближенных значений приспособленности; корреляции между истинным и приближенным значениями приспособленности; корреляции между истинными и приближенными рангами приспособленности [Jin и соавт., 2003].

В предыдущих разделах обсуждалось несколько разных способов аппроксимирования функции приспособленности. Есть также ряд других методов, которые мы не обсуждали, и каждый метод содержит несколько вариаций и регулировочных параметров. Трудно определить «лучший» метод аппроксимации функции приспособленности. В дополнение к метрическим показателям среднеквадратической и максимальной ошибок уравнения (21.35) существует несколько критериев, которые необходимо учитывать при оценивании методов аппроксимации функции приспособленности.

- Насколько точен аппроксимационный метод для данной задачи?
- Независимо от точности аппроксимационного метода, насколько хорошо эволюционный алгоритм работает при использовании аппроксимационного метода? Обратите внимание, что относительная результативность разных эволюционных алгоритмов, возможно, будет варьироваться в зависимости от метода. Например, ЭА #1 вполне может показывать лучшую результативность с аппроксимационным методом *A*, в то время как ЭА #2 вполне может показывать лучшую результативность с аппроксимационным методом *B*.
- Насколько аппроксимационный метод уменьшает вычислительные усилия?
- Насколько сложен аппроксимационный метод? Это влияет на удобство в сопровождении, расширяемость и переносимость программного кода. Насколько легко модифицировать программный код (удобство в сопровождении)? Насколько легко добавлять новые функции или функциональные возможности (расширяемость)? Насколько легко переносить на другие вычислительные

платформы или другие оптимизационные задачи (переносимость)?

Элитарность

Наконец, отметим, что дорогостоящие функции приспособленности могут быть исключением из правила всегда использовать элитарность в наших эволюционных алгоритмах. До этого момента мы, как правило, рекомендовали элитарность, для того чтобы оставлять лучшую особь(ей) каждое поколение. Однако эволюционные алгоритмы с малым числом оцениваний функции могут быть результативнее без элитарности, чем с элитарностью [Torregosa и Kanok-Nukulchai, 2002]. Это объясняется тем, что, когда размер популяции невелик или число оцениваний невелико, разведывание становится относительно важнее, чем эксплуатация. Неэлитарные эволюционные алгоритмы могут способствовать увеличению разведывания.

21.2. Динамические функции приспособленности

Функции приспособленности часто со временем изменяются, то есть они нестационарны. Иногда они меняются, потому что среда эксперимента по оцениванию приспособленности со временем меняется. Например, если мы пытаемся отрегулировать контроллер робота, среда или задача робота может со временем измениться. Параметры робота также могут со временем меняться по мере старения датчиков и электромеханических компонентов. Иногда требования заказчика или клиента меняются, то есть люди меняют свое мнение о том, чего они хотят. Бывает, со временем меняются ограничения, потому что ресурсы расходуются и пополняются. В этом разделе обсуждается вопрос использования эволюционных алгоритмов для отслеживания динамического оптимума.

■ Пример 21.4

Данный пример *не* иллюстрирует динамический эволюционный алгоритм, а просто иллюстрирует влияние динамики в эталонной оптимизационной функции на результативность эволюционного алгоритма. Мы используем упрощенный генератор динамической эталонной функции на рис. С.24, а в качестве базисной функции применяем функцию Экли. Каждые 100 поколений (то есть $E_{\text{update}} = 100$ поколений на рис. С.24) мы вставляем в функцию динамику и используем размерность задачи 10. Мы выполняем алгоритм биогеографической оптимизации

(BBO, глава 14) с популяцией размером 50 и параметром элитарности 2 и заменяем дублирующих особей в каждом поколении случайно генерируемыми особями. На рис. 21.15 показана результативность алгоритма BBO на стационарной функции Экли и на динамической функции Экли, усредненно по 20 симуляциям Монте-Карло. Результативность идентична для первых 100 поколений. Но мы видим, что для динамической функции алгоритм BBO, по существу, должен начинать с самого начала каждые 100 поколений, потому что функция коренным образом меняется. Данный пример иллюстрирует, что эволюционным алгоритмам требуются методы, которые могут интеллектуально обрабатывать динамические изменения в стоимостной функции.

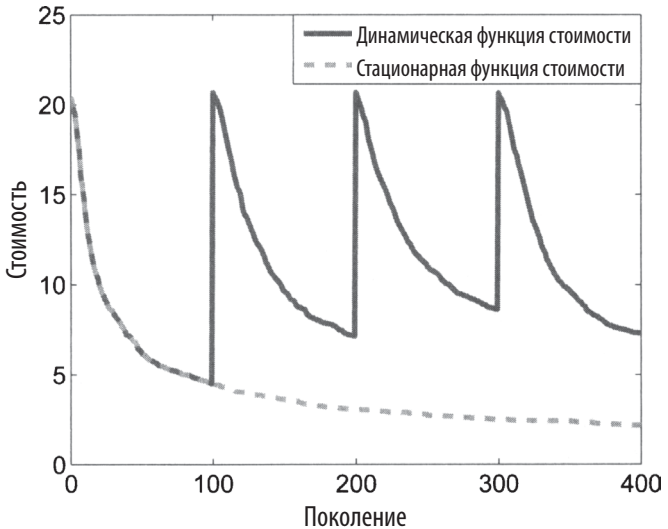


Рис. 21.15. Пример 21.4: данный рисунок показывает результативность алгоритма BBO на 10-мерных стационарных и динамических функциях Экли, усредненно по 20 симуляциям Монте-Карло. Динамика вставляется в динамическую версию эталона каждые 100 поколений, что приводит к тому, что алгоритм BBO теряет все эволюционно-поступательное развитие, которое он накопил до этого момента

■

Первой задачей в динамической оптимизации является обнаружение изменения в ландшафте функции приспособленности. Мы можем обнаруживать такое изменение, используя несколько маркерных особей и оценивая их значения приспособленности в каждом поколении. Если их значения приспособленности значительно из поколения в поколение изменяются (за пределы того, что можно было бы ожидать от

шума), то мы можем сделать вывод, что ландшафт приспособленности изменился; то есть оптимизационная задача изменилась. Но использование маркеров не является ошибкоустойчивым методом. Обнаружение изменения ландшафта не является утверждением «или-или». Ландшафт может меняться в местах расположения маркеров, оставаясь фиксированным в оптимальной точке. И наоборот, ландшафт может изменяться в оптимальной точке, оставаясь фиксированным в местах расположения маркеров. Эти трудности показаны на рис. 21.16.

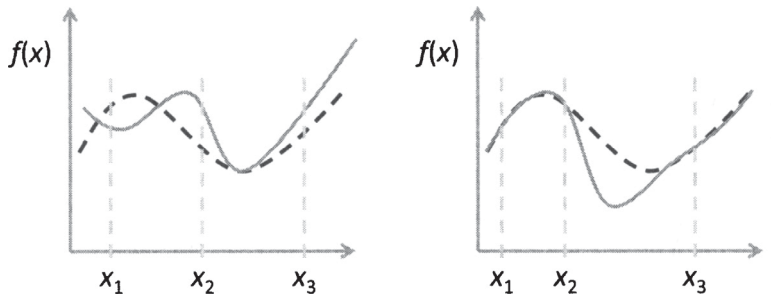


Рис. 21.16. Обнаружение изменения в динамической функции приспособленности. Пунктирная линия на каждом графике является исходной функцией приспособленности, и сплошная линия – новой функцией приспособленности. Три точки x_1, x_2 и x_3 являются маркерами, которые мы используем для обнаружения изменения функции приспособленности. Рисунок слева показывает, что, несмотря на то что приспособленность маркеров резко изменилась, оптимум не изменился. На рисунке справа показано обратное: хотя приспособленность маркеров не изменилась, оптимум сильно изменился

После того как мы обнаружим изменение в ландшафте приспособленности, мы можем принять один из нескольких подходов к отслеживанию изменения оптимума. Одна из возможностей состоит в том, чтобы полностью заменить старую популяцию на новую случайную популяцию и перезапустить процесс эволюционной оптимизации. Данная крайняя мера не использует информацию из предыдущего процесса оптимизации повторно. Это вполне может быть уместно, если ландшафт приспособленности изменился настолько резко, что предыдущая популяция не содержит никакой полезной информации о новом ландшафте. Такая ситуация, по сути, влечет за собой запуск нового эволюционного алгоритма на новой оптимизационной задаче.

Вместе с тем в большинстве практических задач существует некоторое сходство между старым ландшафтом приспособленности и новым

ландшафтом. То есть ландшафт приспособленности меняется постепенно, а не резко. В этом случае мы хотим разведывать новый ландшафт, а также эксплуатировать результаты поступательного развития эволюционного алгоритма на старом ландшафте. Например, мы можем оставить бóльшую часть старой популяции, но посеять ее несколькими новыми особями в нашей попытке разведать новый ландшафт. Мы можем также временно увеличить скорость мутации, чтобы временно увеличить разведывание. Этот подход называется гипермутацией [Cobb и Grefenstette, 1993]. Число новых особей, которые мы вводим в популяцию, величина, на которую мы увеличиваем мутацию, и число поколений, для которых мы увеличиваем мутацию, – все это управляет балансом между разведыванием и эксплуатацией. Этот баланс определяет, насколько эволюционный алгоритм адаптируется к новым ландшафтам и насколько он опирается на свои прошлые результаты.

В следующих разделах мы обсудим несколько динамических эволюционно-алгоритмических подходов, включая предсказательный эволюционный алгоритм (раздел 21.2.1), иммигрантские эволюционные алгоритмы (раздел 21.2.2) и эволюционные алгоритмы на основе памяти (раздел 21.2.3). Мы также обсудим проблему оценивания результативности эволюционных алгоритмов на динамических оптимизационных задачах (раздел 21.2.4).

21.2.1. *Предсказательный эволюционный алгоритм*

Один из подходов к динамической оптимизации заключается в сочетании техники прогнозирования с эволюционным алгоритмом. В результате получится алгоритм, именуемый предсказательным эволюционным алгоритмом (predictive EA) [Hatzakis и Wallace, 2006]. Если эволюционный алгоритм работает и его оптимум со временем меняется предсказуемым образом, то мы, очевидно, можем создать модель того, как он изменяется со временем. Затем, когда будет обнаружено изменение ландшафта, мы сможем использовать эту модель для посева новых членов популяции. Ключевым условием здесь является то, что оптимум должен меняться «предсказуемым образом». Если это условие не выполняется, то мы можем повторно посеять всю популяцию случайно инициализированной популяцией и просто перезапустить эволюционный алгоритм. Рисунок 21.17 иллюстрирует основную идею предсказательного эволюционного алгоритма, а рис. 21.18 показывает пример динамической эволюции эволюционного-алгоритмического оптимума.

Инициализировать эволюционно-алгоритмическую популяцию
$X^* \leftarrow \emptyset$
While not (критерий останова)
  Выполнить эволюционный алгоритм для T поколений либо до тех пор,
    пока не будет обнаружено изменение в ландшафте приспособленности
  Обозначить лучшую особь в эволюционном алгоритме как x*
  $X^* \leftarrow \{X^*, x^*\}$
  Экстраполировать последовательность X^* для оценивания нового оптимума $\hat{x}^*$
  Создать субпопуляцию S особей, которые находятся рядом с $\hat{x}^*$
  Заменить несколько особей в эволюционно-алгоритмической популяции с S
Next поколение

Рис. 21.17. Схематичное описание предсказательного эволюционного алгоритма для динамической оптимизации

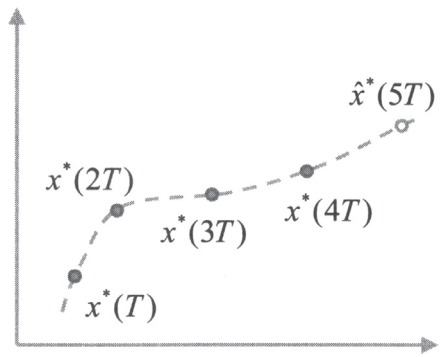


Рис. 21.18. Пример поступательного развития эволюционно-алгоритмического оптимума в двумерном пространстве и его предсказанное значение. $x^*(T)$, $x^*(2T)$, $x^*(3T)$ и $x^*(4T)$ являются эволюционно-алгоритмическими оптимумами после T, $2T$, $3T$ и $4T$ поколений, $\hat{x}^*(5T)$ – предсказанный оптимум, который используется для посева следующего эволюционного алгоритма

Некоторые детали реализации на рис. 21.17 предоставлены разработчику эволюционного алгоритма. Например, если между экстраполяциями мы будем выполнять для постоянного числа поколений T, то как выбрать значение T? Как обнаружить изменения в ландшафте приспособленности? Если мы используем маркеры, то сколько маркеров мы должны применять? Если используем слишком мало маркеров, то можем пропустить изменение. Но если используем слишком много маркеров, то можем потратить время на ненужные оценивания функции приспособленности.

Как использовать множество X^* на рис. 21.17 для оценивания нового оптимума? То есть какой экстраполяционный алгоритм мы должны

использовать? Как создать субпопуляцию S, близкую к $\hat{x}^*$? Мы можем просто установить S равной одноэлементному множеству $\{\hat{x}^*\}$. Еще одна возможность – установить S равной множеству M мутированных версий $\hat{x}^*$, где M – это определяемая пользователем константа. Еще один вариант – установить S равной детерминированному множеству из M особей в гиперкубе или гиперсфере, окружающей $\hat{x}^*$. Если мы будем отслеживать точность нашего предсказания $\hat{x}^*$ из одного исполнения эволюционного алгоритма к другому, то сможем ее использовать для определения размера гиперобъема. Если мы обнаружим, что $\hat{x}^*$ неизменно точное, то можем использовать S с небольшой мощностью и с небольшим гиперобъемом. Если же обнаружим, что $\hat{x}^*$ неизменно неточно, то мы должны увеличить мощность S и ее гиперобъем. По сути дела, мы могли бы использовать эту идею для адаптивной корректировки мощности и гиперобъема S. Наконец, мы должны решить, какие особи в старой популяции заменить на S на рис. 21.17. Распространенными вариантами являются либо замена худших особей в старой популяции, либо замена случайно отобранных особей в популяции.

21.2.2. *Иммигрантские схемы*

Возможны ситуации, когда мы не можем обнаружить изменения в функции приспособленности или когда функция приспособленности меняется почти непрерывно. В этом случае мы можем постоянно вводить в популяцию новых особей в попытке сделать эволюционный алгоритм устойчивым к изменениям ландшафта приспособленности. Такие алгоритмы называются иммигрантскими схемами (immigrant scheme) [Yu и соавт., 2009]. Существует два основных подхода к иммигрантским схемам. В прямой иммигрантской схеме для создания новых особей используются особи в популяции. Этот подход похож на стандартные алгоритмы рекомбинации и мутации, в которых для создания детской популяции используется родительская популяция. Например, прямая иммигрантская схема может мутировать или рекомбинировать элитарных особей из текущего или прошлого поколения для создания новых особей. Косвенная иммигрантская схема создает новых особей, опираясь на модель популяции. Например, для моделирования популяции мы можем применить алгоритм типа популяционного инкрементного самообучения (PBIL) (см. раздел 13.2.3), а затем создать новых особей на основе данной модели.

После того как мы решили использовать либо прямую, либо косвенную иммигрантскую схему, нам нужно решить следующее.

1. Как генерировать новых особей? Как уже упоминалось выше, мы могли бы сгенерировать множество особей X_e на основе элитарных особей. Мы также могли бы сгенерировать множество случайных особей X_r. Если мы думаем, что ландшафт приспособленности вполне может радикально меняться, то мы могли бы создать множество двойных особей X_d, то есть таких же, как противоположные особи главы 16. Наконец, мы могли бы применить комбинацию этих трех вариантов. Если будем отслеживать то, насколько хорошо каждый из этих типов особей работает, то мы могли бы адаптировать число особей X_e, X_r и X_d, которых мы вводим каждое поколение [Yu и соавт., 2009].
2. Сколько новых особей вводить в популяцию? Большинство исследователей вводит около $0.2N$ или $0.3N$ новых особей, где N – это размер популяции. Если мы сможем определить число или частоту изменений в ландшафте приспособленности, то тогда сможем соответствующим образом корректировать коэффициент замещения. Например, если обнаружим большие изменения в ландшафте приспособленности, то мы, возможно, захотим вводить больше новых особей.
3. Какие особи в популяции заменять на новые особи? Один из распространенных ответов на этот вопрос состоит в замене случайно отобранных особей. Еще один ответ – заменять худших особей. Здесь следует помнить один момент, который заключается в том, что новые особи, возможно, будут не очень приспособлены в популяции, но их приспособленность, вероятно, будет улучшаться со временем по мере изменения ландшафта. Поэтому мы, возможно, захотим отслеживать возрастной фактор, который не позволяет заменять особь до тех пор, пока она не превысит определенный возраст [Tinós и Yang, 2005].

На рис. 21.19 представлено схематичное описание иммигрантского эволюционного алгоритма для динамической оптимизации. Вместе с тем он оставляет много места для решений проектировщика эволюционного алгоритма.

1. Какие значения следует использовать для r_r, r_d и r_e? Как подразумевалось ранее, обычно используемое значение для общей доли замещения r_T составляет около 0.2 или 0.3. Чаще мы начинаем с того же числа случайных, двойных и элитарных особей-заменителей, так чтобы $r_r \approx r_d \approx r_e \approx r_T/3$.

N = размер популяции
r_r = доля случайных особей, создаваемых каждое поколение
r_e = доля элитарных особей, создаваемых каждое поколение
r_d = доля двойных особей, создаваемых каждое поколение
Создать N исходных особей $\{x_i\}$
Оценить приспособленность N особей
While not (критерий останова)
  Применить метод рекомбинации/мутации эволюционирования $\{x_i\}$
    для следующего поколения
  Оценить приспособленность особей $\{x_i\}$
  $X \leftarrow \{Nr_r$ случайно сгенерированных особей$\}$
  $X \leftarrow X \cup \{Nr_e$ мутаций элитарных особей$\}$
  $X \leftarrow X \cup \{Nr_d$ двойных особей$\}$
  Оценить приспособленность особей в X
  Заменить особей в $\{x_i\}$ на особей из X
  Адаптировать r_r, r_e и r_d на основе результативности новых особей
Next поколение

Рис. 21.19. Схематичное описание иммигрантского эволюционного алгоритма для динамической оптимизации

2. Следует ли адаптировать r_r, r_d и r_e? Мы можем адаптировать их для того, чтобы попытаться улучшить результативность эволюционного алгоритма, как показано на рис. 21.19, но это усложнит наш алгоритм. В публикации [Yu и соавт., 2009] предлагается следующая ниже схема адаптации. Каждое поколение мы оцениваем приспособленность новых особей. Если группа случайных особей работает лучше, чем группа двойных и элитарных особей, то мы выполняем следующие назначения:

$$\begin{aligned} r_d &\leftarrow \max(r_{\min}, r_d - \alpha) \\ r_e &\leftarrow \max(r_{\min}, r_e - \alpha) \\ r_r &\leftarrow r_r - r_d - r_e \end{aligned} \tag{21.36}$$

где α управляет скоростью адаптации, $r_{\min}$ определяет минимальную долю каждого типа новой особи, создаваемой каждое поколение, и константа r_T определяет общую долю новых особей, создаваемых каждое поколение. В публикации [Yu и соавт., 2009] используется $\alpha \approx 0.02$ и $r_{\min} = 0.04$. Если группа двойных особей или элитарных особей работает лучше всех, то мы соответственно переписываем уравнение (21.36), увеличив число типов высокорезультативных особей, которые создаются в течение следующего

поколения, и уменьшив число других типов. Этот адаптационный подход ставит вопрос: как решать, какой тип новых особей работает «лучше всех»? Мы можем принять решение на основе лучшей случайной особи, лучшей двойной особи и лучшей элитарной особи; либо мы можем принять решение на основе средней результативности всей группы случайных, двойных и элитарных особей.

3. Какие особи в популяции $\{x_i\}$ следует заменить на новые особи? Мы уже кратко говорили об этом. В публикации [Yu и соавт., 2009] заменяются худшие особи, но мы также можем заменить случайно отобранных особей или комбинацию худших и случайных особей. Кроме того, мы можем сгруппировать исходную популяцию $\{x_i\}$ с замещающей популяцией X и использовать стохастический отборочный механизм для выбора лучших N особей. Можем использовать любой из отборочных механизмов, описанных в разделе 8.7.1.
4. И последнее, но не менее важное: напомним, что нам нужно выбрать метод рекомбинации и мутации с целью эволюционирования $\{x_i\}$ для следующего поколения на рис. 21.19. Для рекомбинации мы можем применить любой эволюционный алгоритм и любой из методов мутации, описанных в разделе 8.9.

■ Пример 21.5

В данном примере мы оцениваем результативность алгоритма биогеографической оптимизации (BBO) на той же самой динамической функции Экли из примера 21.4. Мы можем обнаружить изменение в функции приспособленности, потому что оставляем две элитарные особи каждое поколение, поэтому лучшая стоимость популяции должна уменьшаться каждое поколение. Если лучшая стоимость увеличивается, то мы делаем вывод, что функция стоимости изменилась[5]. Мы пробуем два разных способа адаптации к изменению стоимостной функции: во-первых, мы реинициализируем популяцию случайно сгенерируемыми особями; во-вторых, используем прямую иммигрантскую схему, приведенную на рис. 21.19. На рис. 21.20 показана результативность алгоритма BBO, усредненно по 20 симуляциям Монте-Карло, для этих двух вариантов перезапуска. Мы видим, что случайный перезапуск ра-

[5] Этот план не является ошибкоустойчивым. Изменение стоимостной функции может привести к снижению лучшей стоимости в популяции. Однако если лучшая стоимость уменьшается из поколения в поколение, то мы не должны жаловаться слишком много, даже если наш алгоритм обнаружения изменения функции приспособленности оказывается безуспешным.

ботает лучше, чем иммигрантская схема. Динамическое изменение в стоимостной функции является настолько случайным (поворот вектора смещения), что замена только 30 % популяции не является достаточно радикальной, чтобы превзойти случайный перезапуск.

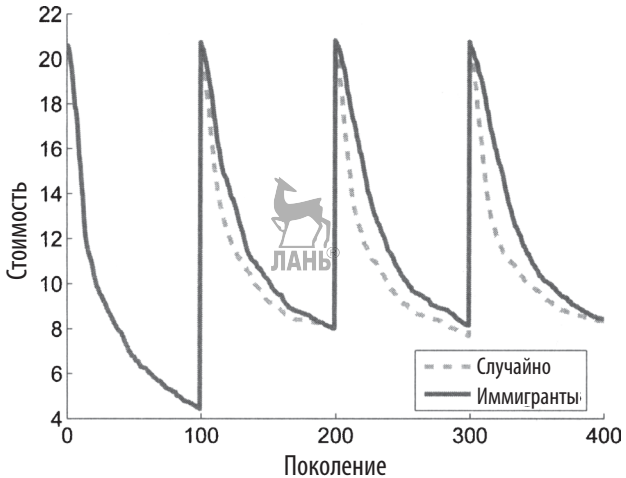


Рис. 21.20. Результаты примера 21.5. Данный рисунок показывает результативность алгоритма BBO на 10-мерной динамической функции Экли, усредненно по 20 симуляциям Монте-Карло. Когда стоимостная функция изменяется посредством поворота вектора смещения каждые 100 поколений, мы либо случайно реинициализируем всю популяцию, либо заменяем 30 % популяции на иммигрантскую схему, показанную на рис. 21.20. Поскольку динамика стоимостной функции настолько случайная, случайный перезапуск превосходит иммигрантскую схему

■

■ Пример 21.6

В данном примере мы снова оцениваем результативность алгоритма BBO для динамической функции Экли. Однако в этом примере динамическое изменение стоимостной функции состоит не из поворота вектора смещения, а из пертурбации вектора смещения на случайную величину:

$$\begin{aligned} \theta(t) &\leftarrow \theta(t-1) + 0.1(x_{\max} - x_{\min})\rho(t-1) \\ \theta(t) &\leftarrow \min(\theta(t), x_{\max} - x^*) \\ \theta(t) &\leftarrow \max(\theta(t), x_{\min} - x^*) \end{aligned} \tag{21.37}$$

где $[x_{\min}, x_{\max}]$ определяет поисковое пространство, x^* – это оптимизирующее значение несмещенной стоимостной функции, $\rho(t-1)$ – случайное число, взятое из гауссова распределения с нулевым средним и единичной дисперсией. Дополнительные сведения о вышеуказанных параметрах см. в приложении С.4. Приведенная выше последовательность назначений гарантирует, что оптимум $f(x-\theta(t))$ находится в поисковой области. Для функции Экли $x^* = 0$, но уравнение (21.37) одинаково хорошо применяется к любому другому эталону. Уравнение (21.37) показывает, что вектор смещения $\theta(t)$ изменяется в соответствии с гауссовым распределением со стандартным отклонением, равным 10 % диапазона поискового пространства. Это умеренное изменение стоимостной функции вполне может быть более реалистичным, чем менее структурированное изменение вектора смещения в примере 21.5. В данном примере мы пробуем три разных варианта адаптации к изменению стоимостной функции: (1) мы реинициализируем популяцию случайно сгенерированными особями; (2) мы используем прямую иммигрантскую схему, показанную на рис. 21.19, где r_r, r_e и r_d равны 10 % от размера популяции, и новых особей для замены худших особей в каждом поколении; (3) мы игнорируем динамику стоимостной функции и не вносим никаких изменений в популяцию. На рис. 21.21 показана результативность алгоритма BBO, усредненно по 20 симуляциям Монте-Карло для этих трех вариантов. Мы видим, что случайный перезапуск работает хуже всего. Это происходит потому, что динамическое изменение стоимостной функции имеет некоторую структуру, поэтому замена всей популяции приводит к потере информации, которая эволюционировала в течение предыдущих 100 поколений. На рис. 21.21 показано, что иммигрантская схема и вариант «игнорировать» работают примерно одинаково.

■

Как правило, мы рассчитываем, что иммигрантский подход будет работать лучше, чем подход «игнорировать». Мы вполне можем улучшить результативность иммигрантской схемы, адаптировав параметры r_r, r_e и r_d, как показано в уравнении (21.36), чего мы не делали в приведенном выше примере. Отметим также, что сравнение, показанное на рис. 21.21, не совсем справедливо, поскольку сравнение трех алгоритмов производится для одного и того же числа поколений. Всякий раз, когда функция изменяется, алгоритм случайного перезапуска требует N оцениваний функции приспособленности; всякий раз, когда функция изменяется, иммигрантский алгоритм требует $0.3N$ оцениваний функ-

ции приспособленности, и вариант «игнорировать» не требует оцениваний функции приспособленности, когда функция изменяется. Вместе с тем функция изменяется только один раз в 100 поколений. Поэтому, несмотря на то что строгое сравнение между тремя подходами должно быть сделано на основе оцениваний функций, а не поколений, в данном примере разница в числе оцениваний функций настолько близка, что мы ее игнорируем.

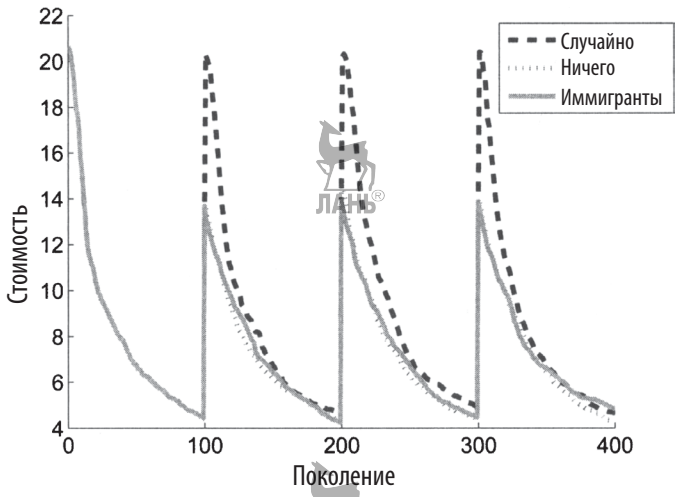


Рис. 21.21. Результаты примера 21.6. Данный рисунок показывает результативность алгоритма BBO на 10-мерной динамической функции Экли, усредненно по 20 симуляциям Монте-Карло. Когда стоимостная функция изменяется посредством умеренной пертурбации вектора смещения каждые 100 поколений, мы либо случайно реинициализируем всю популяцию, заменяем 30 % популяции иммигрантской схемой на рис. 21.19, либо просто ничего не делаем. Поскольку изменение стоимостной функции относительно структурировано, случайный перезапуск выполняется хуже всего

21.2.3. *Подходы на основе памяти*

Иногда стоимостная функция изменяется среди конечного множества функций. Например, предположим, что нам необходимо оптимизировать процесс управления на производственном предприятии. Параметры производственного процесса могут периодически меняться, по мере того как руководитель предприятия изменяет задачу или производит замену деталей машины. Эти изменения приводят к изменению стоимостной функции, но изменения не случайны. Изменения

являются детерминированными, но детали этих изменений могут быть недоступны контроллеру и оптимизационному процессу. В подобных случаях мы, возможно, захотим отслеживать предыдущие оптимальные решения и вставлять их в популяцию при обнаружении изменения в стоимостной функции.

Эксплицитные подходы на основе памяти хранят хороших особей в архиве [Woldesenbet и Yen, 2009]. При обнаружении изменения стоимостной функции особи извлекаются из архива и вставляются в популяцию. Если стоимостная функция изменится на ранее встречавшуюся функцию, то особи из архива будут хорошими решениями, и эволюционный алгоритм очень быстро сойдется к новому оптимуму. Рисунок 21.22 иллюстрирует этот подход. На рисунке представлено лишь схематичное описание базового алгоритма, и в нем не рассматриваются некоторые важные детали. Следующие ниже вопросы предоставляют широкие возможности для дополнительных исследований.

1. Сколько особей следует хранить в архиве, если изменение не зафиксировано?
2. Каким большим должен быть архив? Рисунок 21.22 не включает верхний предел его размера. На практике мы, возможно, захотим попытаться определить «операционную точку» задачи. Если операционная точка совпадает с ранее встречавшейся операционной точкой, то элитарные особи уже будут сохранены в архиве для этой операционной точки, и мы не захотим сохранять текущие элитарные особи в архиве, если они не будут лучше, чем ранее заархивированные значения. Например, предположим, что после обнаружения изменения в $f(\cdot)$ мы производим поиск в архиве и находим, что текущее элитарное множество аналогично ранее сохраненным особям. Мы вполне можем тогда сделать вывод, что задача, которую только что решил эволюционный алгоритм, совпадает с ранее решенной задачей, поэтому нет необходимости сохранять текущие элитарные особи в архиве, если они не лучше, чем архив.
3. Как обнаруживать изменение в $f(\cdot)$? Мы обсудили этот вопрос в начале раздела 21.2.
4. Когда обнаруживать изменение в $f(\cdot)$? Какие особи в популяции должны быть заменены архивными особями? Какие архивные особи должны быть включены в популяцию?

```
Создать исходную популяцию {x_i}
Архив A ← Ø
While not (критерий останова)
    Применить метод рекомбинации/мутации для эволюционирования {x_i}
        для следующего поколения
    Оценить приспособленность особей {x_i}
    Сохранить лучших особей из {x_i} в элитарном множестве E
    If обнаружено изменение в f(·) then
        Заменить несколько особей в популяции {x_i} особями из архива A
        Сохранить элитарное множество E в архиве A
    End if
Next поколение
```

Рис. 21.22. Схематичное описание эксплицитного подхода к динамической оптимизации на основе памяти. Данный алгоритм в особенности подходит для функций приспособленности, периодических со временем или равных одному из конечного множества функций приспособленности

21.2.4. *Оценивание результативности динамической оптимизации*

Оценивание результативности динамической оптимизации отличается от оценивания результативности стационарной оптимизации. При оценивании эффективности стационарной оптимизации мы обычно смотрим на популяцию последнего поколения, чтобы увидеть, насколько хорошие результаты были показаны эволюционным алгоритмом. Однако при оценивании результативности динамической оптимизации функция приспособленности изменяется из поколения в поколение. Поэтому взгляд только на последнее поколение не дает хорошего общего показателя результативности эволюционного алгоритма. Вместо этого нам нужно посмотреть на результативность во всех поколениях [Yu и соавт., 2009]. Двумя общепринятыми метрическими показателями для динамических оптимизационных задач являются средняя лучшая результативность $\bar{f}_b$ и усредненно-средняя результативность $\bar{f}_a$[6]:

$$\bar{f}_b = \frac{1}{G}\sum_{i=1}^{G} f_{i,b}$$

$$\bar{f}_a = \frac{1}{G}\sum_{i=1}^{G} f_{i,a} \qquad (21.38)$$

[6] В оригинале mean average performance. – *Прим. перев.*

где G – это число поколений, $f_{i,b}$ – лучшая приспособленность в i-м поколении, $f_{i,a}$ – средняя приспособленность в i-м поколении. Эти величины дают нам хорошие метрические показатели для сравнения результативности динамической оптимизации между разными эволюционными алгоритмами. Если мы выполним несколько симуляций Монте-Карло, то у нас будет дополнительный уровень усреднения: мы будем усреднять $\bar{f}_b$ и $\bar{f}_a$ на симуляциях Монте-Карло. Более подробно оценивание результативности обсуждается в приложении В.2.2.

Создать исходную популяцию $\{x_i\}$
Архив $A \leftarrow \emptyset$
While not (критерий останова)
  Применить метод рекомбинации/мутации для эволюционирования $\{x_i\}$
    для следующего поколения.
  Оценить приспособленность особей $\{x_i\}$
  Сохранить лучших особей из $\{x_i\}$ в элитарном множестве E.
  If обнаружено изменение в $f(\cdot)$ **then**
    Заменить несколько особей в популяции $\{x_i\}$ особями из архива A.
    Сохранить элитарное множество E в архиве A.
  End if
Next поколение

Рис. 21.22. Схематичное описание эксплицитного подхода к динамической оптимизации на основе памяти. Данный алгоритм в особенности подходит для функций приспособленности, периодических со временем или равных одному из конечного множества функций приспособленности

21.3. Шумные функции приспособленности

Оценивание функций припособленности в эволюционных алгоритмах часто сопровождается шумом. Например, при экспериментальных оцениваниях функций приспособленности шум могут вызывать неточности датчиков. Кроме того, если мы измеряем значения функции приспособленности с помощью симуляционного программного обеспечения, то шум в оцениваниях функции приспособленности могут вызывать аппроксимационные ошибки в нашем программном обеспечении. Инго Рехенберг, изобретатель эволюционной стратегии, вероятно, был первым, кто исследовал влияние шума на эволюционные алгоритмы [Rechenberg, 1973].

Оценивание шумной функции приспособленности может привести к тому, что высокая приспособленность будет ошибочно назначена низко

приспособленной особи. И наоборот, оно может привести к тому, что низкая приспособленность ошибочно будет назначена высоко приспособленной особи. Рисунок 21.23 иллюстрирует функцию плотности вероятности (PDF) двух шумных, но несмещенных функций приспособленности $f(x_1)$ и $f(x_2)$. Мы видим, что истинное значение $f(x_1)$ равно 0, а истинное значение $f(x_2)$ равно 4, но оценивания шумные. Следовательно, x_1, возможно, имеет оцененную приспособленность, которая на самом деле больше, чем у x_2. Такая ситуация приведет к неточной оценке относительных значений приспособленности x_1 и x_2, что может привести к тому, что эволюционный алгоритм отберет неправильную особь для рекомбинации. То есть шум может обмануть эволюционный алгоритм.

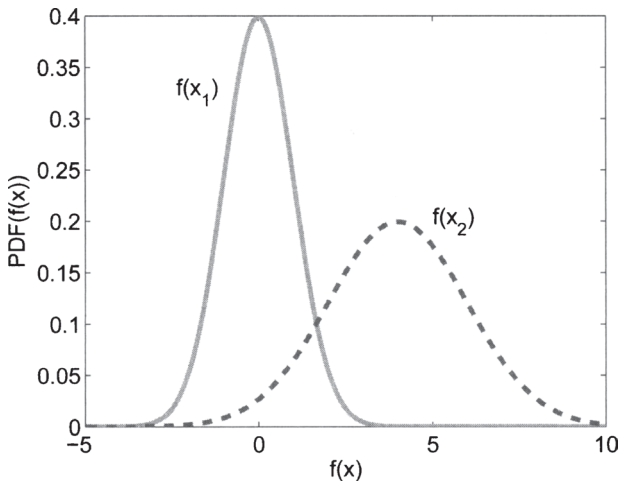


Рис. 21.23. На данном рисунке показаны функции плотности вероятности (PDF) двух функций приспособленности. x_2, имеющая истинное значение 4, больше приспособлена, чем x_1, имеющая истинное значение 0. Но в зависимости от шума, который реализуется при оценивании функции приспособленности, эволюционный алгоритм вполне может подумать, что x_1 больше приспособлена, чем x_2. Это может привести к неправильному отбору для следующего поколения

Когда у нас есть шумные оценивания функций приспособленности, мы не можем быть уверены, какое из них лучше. Предположим, что у нас есть две особи x_1 и x_2, два верных значения приспособленности $f_t(x_1)$ и $f_t(x_2)$ (к примеру, изображенные на рис. 21.23, как 0 и 4), и два оценивания шумной функции приспособленности $f(x_1)$ и $f(x_2)$. Из-за шума $f(x_1) > f(x_2)$ не обязательно означает, что $f_t(x_1) > f_t(x_2)$. Однако если мы знаем функции плотности вероятности (PDF) для $f(x_1)$ и $f(x_2)$, то мо-

жем вычислить вероятность того, что $f_t(x_1) > f_t(x_2)$, при условии что даны конкретные значения $f(x_1)$ и $f(x_2)$. Мы не будем здесь пускаться в математику, но можем выполнить вычисления, используя стандартные методы из теории вероятностей [Grinstead и Snell, 1997], [Mitzenmacher и Upfal, 2005]. Обратите внимание, что в ходе исполнения эволюционного алгоритма у нас не будет функции плотности вероятности (PDF) шумной функции приспособленности, так как мы не знаем истинной функции приспособленности (предполагаемого среднего значения шумной функции приспособленности). Однако мы вполне можем знать PDF истинной функции приспособленности. Эта ситуация аналогична той, которая показана на рис. 21.23, за исключением того, что вместо рассмотрения шумной функции приспособленности как случайной величины со средним значением, равным истинной функции приспособленности, мы можем рассматривать истинную функцию как случайную величину со средним значением, равным значению шумной оцененной функции приспособленности.

В следующих далее разделах рассматриваются три метода работы с шумными функциями приспособленности. В разделе 21.3.1 обсуждается подход на основе многократного выборочного отбора, в разделе 21.3.2 – подход к оцениванию приспособленности, и в разделе 21.3.3 – эволюционный алгоритм на основе фильтра Калмана, в котором фильтр Калмана используется для оценивания значений функции приспособленности.

21.3.1. *Многократное взятие проб*

Один из простых подходов к сокращению шума заключается в многократном взятии проб функции приспособленности. Если мы оцениваем функцию приспособленности для данной особи N раз и значения шума из N проб независимы, то дисперсия средней функции приспособленности уменьшается в N раз [Grinstead и Snell, 1997], [Mitzenmacher и Upfal, 2005]. Предположим, что оцененная приспособленность $g(x)$ кандидатного решения x задана

$$g(x) = f(x) + w, \tag{21.39}$$

где $f(x)$ – это истинная приспособленность и w – шум с нулевым средним значением и дисперсией σ^2. Это означает, что измеренное значение приспособленности $g(x)$ имеет среднее значение $f(x)$ и дисперсию σ^2. Если мы возьмем N независимых замеров $\{g_i(x)\}$, то каждый замер $g_i(x)$ имеет дисперсию σ^2, лучшая оценка истинной приспособленности равна

$$\hat{f}(x) = \frac{1}{N}\sum_{i=1}^{N} g_i(x) \tag{21.40}$$

и дисперсия $\hat{f}(x)$ равна σ^2/N. Рисунок 21.24 иллюстрирует эту идею. Среднее значение множества из N оцениваний шумной функции приспособленности в N раз точнее, чем одно оценивание.

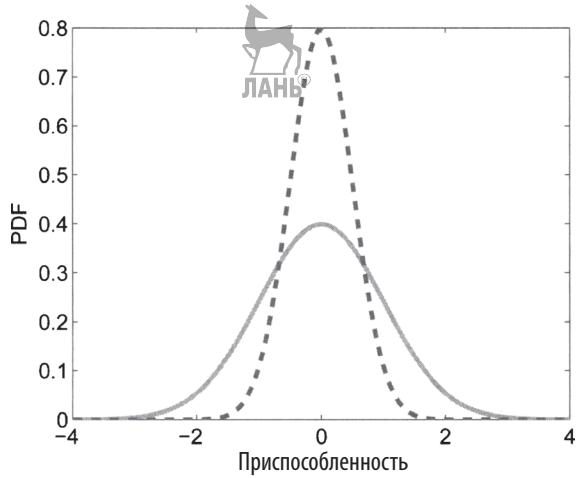


Рис. 21.24. Данный рисунок иллюстрирует стратегию многократного взятия проб для оцениваний шумной функции приспособленности. Сплошная линия показывает функцию плотности вероятности (PDF) шумной функции приспособленности. Пунктирная линия показывает PDF среднего значения четырех оцениваний функции приспособленности. Обе имеют нулевое среднее значение, но усредненное оценивание имеет дисперсию, которая составляет 1/4 от одного оценивания. Усредненное оценивание, вероятно, будет гораздо ближе к среднему значению, чем одно оценивание

Однако стратегия многократного взятия проб полностью допустима только в том случае, если шум оценивания функции приспособленности не зависит из одной пробы в другую. Например, предположим, что мы замеряем приспособленность кандидатных решений с помощью шумных измерительных приборов. Если шум измерительных приборов коррелирует с самим собой со временем из одной пробы в другую, то усреднение N проб не сокращает дисперсию в N раз. В этом случае величина, на которую сокращается дисперсия, зависит от корреляции шума из одной пробы в другую.

Если у нас есть N оцениваний приспособленности $\{g_i(x)\}$ кандидатного решения x, то мы можем найти оценку $\hat{\sigma}^2$ дисперсии σ^2 оценки приспособленности следующим образом:

$$\hat{f}(x) = \frac{1}{N}\sum_{i=1}^{N} g_i(x)$$

$$\hat{\sigma}^2 = \frac{1}{N-1}\sum_{i=1}^{N}(\hat{f}(x) - g_i(x))^2 \qquad (21.41)$$

Интуитивно кажется, что уравнение для $\hat{\sigma}^2$ должно в знаменателе иметь N вместо $(N-1)$, но член $(N-1)$ предпочтительнее, поскольку он дает несмещенную оценку дисперсии [Simon, 2006, задача 3.6]. Мы можем применить уравнение (21.41), чтобы увидеть, сколько проб шумной функции приспособленности мы должны взять для достижения желаемой дисперсии в нашей оценке значения приспособленности (см. задачу 21.7). Требуемая дисперсия определяется пользователем и зависит от конкретной задачи. По мере того как $N \to \infty$, дисперсия стремится к 0, и наша оценка значения приспособленности становится безошибочной.

Некоторые исследователи предлагают брать пробы фиксированное число раз для каждого кандидатного решения. Но при этом игнорируется возможность того, что шум оценивания приспособленности может быть неодинаковым для всех кандидатных решений [Di Pietro и соавт., 2004]. Это может быть в том случае, например, если оценивание приспособленности выполняется с помощью датчиков, шум которых пропорционален сигналу, который они замеряют [Arnold, 2002]. В этом случае уравнение (21.39) заменяется на

$$g(x) = f(x) + w(x). \qquad (21.42)$$

То есть шум оценивания приспособленности зависит от конкретного оцениваемого кандидатного решения. В этом случае взятие проб фиксированное число раз для каждого x будет неэффективным, поскольку оно приведет к разной точности для разных кандидатных решений. Но мы по-прежнему можем использовать уравнение (21.41), для того чтобы решить, сколько раз опробовать каждое кандидатное решение.

Данная стратегия многократного взятия проб может потребовать много проб для достижения желаемой дисперсии. Это вполне может быть неосуществимо для дорогостоящих функций приспособленности. Поэтому нам, возможно, потребуется скомбинировать многократное взятие проб с одним из методов аппроксимации функции приспособленности, описанных в разделе 21.1.1. Многократное взятие проб также можно сократить, выполняя его только для лучших особей в популяции. Опираясь на одно оценивание функции приспособленности, мы вполне можем не узнать, какие особи являются лучшими, но мы сможем,

по крайней мере, получить представление, и сможем зарезервировать вычислительные усилия, необходимые для многократного взятия проб, для лучших особей в популяции [Branke, 1998].

Еще один подход к определению числа многократных проб основан на идее наследования приспособленности, обсуждаемой в разделе 21.1.1 [Bui и соавт., 2005]. Предположим, что несколько родителей $\{p_i\}$ производят ребенка x. Допустим дальше, что i-й родитель имеет оцененную приспособленность $g(p_i)$ и стандартное отклонение $\sigma(p_i)$. Прежде чем оценим приспособленность ребенка, мы можем применить наследование приспособленности и получить оценки его приспособленности $\hat{f}(x)$ и его стандартного отклонения $\hat{\sigma}(x)$. Затем мы оцениваем приспособленность ребенка один раз. Если оценивание его приспособленности $g(x)$ попадает в пределы $\pm 3\hat{\sigma}(x)$ приспособленности $\hat{f}(x)$, то мы принимаем $g(x)$ как допустимую. В противном случае мы делаем вывод, что $g(x)$ очень зашумлена, и поэтому мы следуем стратегии многократного взятия проб с целью сокращения шума. Этот подход в некотором смысле противоречит здравому смыслу. Он указывает на то, что чем больше значение шума у родителей, тем больше вероятность того, что мы примем оценивание ребенка. Интуитивно мы ожидаем, что верно обратное, то есть чем шумнее значения приспособленности у родителей, тем вероятнее, что мы должны многократно взять пробы приспособленности ребенка [Syberfeldt и соавт., 2010].

Один из способов сокращения числа оцениваний приспособленности – брать пробы только несколько раз в начале эволюционного алгоритма и чаще по мере поступательного развития эволюционного алгоритма [Syberfeldt и соавт., 2010]. Это имеет смысл, потому что поиск в эволюционном алгоритме обычно относительно крупнозернистый в начале процесса оптимизации. В течение первых поколений эволюционный алгоритм пытается найти общую окрестность оптимального решения, в последующих поколениях он пытается сходиться к более точным решениям. Точные значения функции приспособленности нужны только тогда, когда эволюционный алгоритм начинает сходиться к концу процесса оптимизации. То есть прецизионность релевантна только в том случае, если точность хорошая (см. задачу 21.8) [Taylor, 1997].

Многие стратегии многократного взятия проб исходят из того, что шум приспособленности нормально распределен. Это верно для многих шумных феноменов в замерах и вычислениях, но не для всех. Если шум не является гауссовым, то многие стратегии многократного взятия проб, которые были опубликованы в научной литературе, должны быть выведены заново.

21.3.2. *Оценивание приспособленности*

Уравнение (21.41) показывает простой усредняющий метод для оценивания приспособленности на основе зашумленных проб. Однако мы также можем использовать более сложные вероятностные методы. Например, в публикации [Sano и Kita, 2002] авторы принимают допущение, что особи x_1 и x_2, близкие друг к другу в поисковом пространстве, имеют одинаковые значения приспособленности:

$$f(x_1) \sim N(f(x_2), kd). \tag{21.43}$$

То есть приспособленность x_1 является гауссовой случайной величиной со средним $f(x_2)$ и дисперсией kd, где k – это неизвестный параметр и d – расстояние между x_1 и x_2. Если оценивание приспособленности $g(x_1)$ имеет дисперсию σ^2, то

$$g(x_1) \sim N(f(x_2), kd + \sigma^2). \tag{21.44}$$

С этими допущениями для оценивания приспособленности x_1 и x_2 при заданных шумных оцениваниях x_1 и x_2 мы можем применить расчеты максимального правдоподобия. То есть для оценивания приспособленности особи мы используем не только шумные оценивания особи, но и шумные оценивания ее соседей [Branke и соавт., 2001].

21.3.3. *Эволюционный алгоритм на основе фильтра Калмана*

Эволюционный алгоритм на основе фильтра Калмана (Kalman EA) предназначен для задач с шумными и дорогостоящими оцениваниями функций приспособленности. Эволюционный алгоритм на основе фильтра Калмана, который был первоначально предложен в контексте генетических алгоритмов, представляет собой подход для отслеживания неопределенности в значениях приспособленности и распределения оцениваний приспособленности соответствующим образом [Stroud, 2001].

Фильтр Калмана является оптимальным эстиматором состояний линейной динамической системы [Simon, 2006]. Эволюционный алгоритм на основе фильтра Калмана принимает допущение, что приспособленность данной особи x постоянна. Мы далее принимаем допущение, что шум оценивания приспособленности не является функцией x. С этими допущениями для отслеживания неопределенности в каждом оценивании приспособленности мы можем применить сокращенную и упро-

щенную скалярную форму фильтра Калмана. Обозначим дисперсию одиночного оценивания функции приспособленности как R. Обозначим дисперсию оценивания приспособленности особи x после k оцениваний функции приспособленности как $P_k(x)$. Обозначим значение k-го оценивания функции приспособленности x как $g_k(x)$. Наконец, обозначаем нашу оценку приспособленности x после k оцениваний функции приспособленности как $\hat{f}_k(x)$. С помощью этой записи мы можем применить теорию фильтра Калмана, для того чтобы записать

$$\begin{aligned} \hat{f}_{k+1}(x) &= \hat{f}_k(x) + \frac{P_k(x)(g_{k+1}(x) - \hat{f}_k(x))}{P_k(x) + R} \\ P_{k+1}(x) &= \frac{P_k(x)R}{P_k(x) + R} \end{aligned} \tag{21.45}$$

для $k = 0, 1, 2, \ldots$. Мы инициализируем $P_0(x) = \infty$ для всех x, и в результате получим

$$\begin{aligned} \hat{f}_1(x) &= g_1(x) \\ P_1(x) &= R \end{aligned} \tag{21.46}$$

То есть наша оценка $\hat{f}_1(x)$ после первого оценивания функции приспособленности $g_1(x)$ просто равняется первому оцениванию. Кроме того, неопределенность $P_1(x)$ в нашей оценке приспособленности после первого оценивания просто равняется неопределенности в оценивании.

Уравнение (21.45) показывает, что всякий раз, когда мы оцениваем приспособленность x, мы модифицируем нашу оценку $\hat{f}(x)$ на основе предыдущей оценки, ее неопределенности и последнего результата оценивания функции приспособленности $g(x)$. Уравнение (21.45) показывает, что

$$\begin{aligned} \lim_{P_k(x) \to 0} \hat{f}_{k+1}(x) &= \hat{f}_k(x) \\ \lim_{P_k(x) \to \infty} \hat{f}_{k+1}(x) &= g_{k+1}(x) \end{aligned} \tag{21.47}$$

Другими словами, если мы полностью уверены в приспособленности x (то есть $P_k(x) = 0$), то дальнейшие оценивания приспособленности x не изменят нашу оценку его приспособленности. С другой стороны, если мы полностью не уверены в приспособленности x (то есть $P_k(x) \to \infty$), то установим нашу оценку его приспособленности равной следующему результату оценивания функции приспособленности.

Уравнение (21.45) также показывает, что всякий раз, когда мы оцениваем приспособленность x, наша неопределенность $P(x)$ в ее значении уменьшается (то есть наша уверенность в ее оценке увеличивается). Уравнение (21.45) показывает, что

$$\begin{aligned}
\lim_{R \to 0} \hat{f}_{k+1}(x) &= g_{k+1}(x) \\
\lim_{R \to 0} P_{k+1}(x) &= 0 \\
\lim_{R \to \infty} \hat{f}_{k+1}(x) &= \hat{f}_k(x) \\
\lim_{R \to \infty} \hat{f}_{k+1}(x) &= P_k(x)
\end{aligned} \tag{21.48}$$

Эти результаты согласуются с интуицией. Если дисперсия шума функции приспособленности R равняется 0, то оценивание функции приспособленности является идеальным, поэтому наша оценка просто равняется результату оценивания функции приспособленности, и неопределенность в нашей оценке функции приспособленности равна 0. С другой стороны, если дисперсия шума функции приспособленности R бесконечна, то шум настолько велик, что оценивания функции приспособленности не дают нам никакой информации. В этом случае дополнительные оценивания функции приспособленности не изменяют нашу оценку значения функции приспособленности, а также не сокращают нашу неопределенность в ее значении.

Эволюционный алгоритм на основе фильтра Калмана отслеживает оценку функции приспособленности $\hat{f}_k(x)$ и дисперсию $P_k(x)$ для каждой особи x из одного оценивания функции приспособленности в следующее (k = 1, 2, ...). Мы выделяем определяемую пользователем долю F имеющихся оцениваний для генерирования и оценивания новых особей. Мы инициализируем нашу оценку приспособленности и дисперсию для каждой новой особи, как описано в уравнении (21.46). Мы используем долю $(1 - F)$ имеющихся оцениваний для переоценки существующих особей. В этом случае обновляем нашу оценку приспособленности и дисперсию, как описано в уравнении (21.45). Всякий раз, когда у нас достаточно ресурсов для оценивания функции приспособленности, мы генерируем случайное число r, равномерно распределенное на [0, 1]. Если $r < F$, то выполняем эволюционно-алгоритмическую рекомбинацию и мутацию, для того чтобы сгенерировать новую особь, а затем оцениваем ее приспособленность; в противном случае мы переоцениваем существующую особь.

Когда наступает время переоценки существующей особи, мы рассматриваем два направляющих принципа. Во-первых, мы можем генерировать больше информации путем переоценки особей, чья дисперсия оценки приспособленности высока. Во-вторых, мы можем генерировать больше полезной информации, переоценивая особей, чья оцененная приспособленность высока. То есть мы не слишком заботимся о том, чтобы получить высокую прецизионность в оценке низко приспособленных особей, потому что мы, вероятно, не заинтересованы в их рекомбинации для будущих эволюционно-алгоритмических поколений. Поэтому в публикации [Stroud, 2001] предлагается следующая ниже стратегия отбора особи x_s для переоценки:

$$\begin{aligned} \overline{f} &\rightarrow \text{среднее значение значений оцененных приспособленностей популяции} \\ \sigma &\rightarrow \text{стандартное отклонение значений оцененных приспособленностей популяции} \\ x_s &\rightarrow \arg\max\{P(x) : \hat{f}(x) > \overline{f} - \sigma\} \end{aligned} \tag{21.49}$$

где мы пренебрегли нижним индексом k на $\hat{f}(x)$ и $P(x)$; мы используем самые последние обновленные значения $\hat{f}(x)$ и $P(x)$ для каждой особи x в уравнении (21.49). Данное уравнение показывает, что среди всех особей, чья оцененная приспособленность больше, чем одно стандартное отклонение вниз от среднего значения, мы отбираем для переоценки ту, которая имеет наибольшую неопределенность. Эта стратегия исходит из того, что $f(x)$ является приспособленностью – такой, что более крупная $f(x)$ лучше.

Мы видим ряд возможностей для дополнительного развития эволюционного алгоритма на основе фильтра Калмана. Например, как распространить его на задачи динамической оптимизации? Как определять оптимальную долю F на основе новой особи? Есть ли способ адаптировать F на основе результативности? Как расширить эволюционный алгоритм Калмана до более общих парадигм фильтрации, таких как H_∞-фильтрация[7] [Simon, 2006]?

21.4. Заключение

Обзоры эволюционно-алгоритмической аппроксимации приспособленности приведены в публикациях [Ong и соавт., 2004], [Jin, 2005],

[7] H_∞-фильтрация, или фильтрация H-бесконечность – это методы, применяемые в теории управления для синтеза регуляторов, достигающих стабилизации с гарантированной результативностью. См. https://en.wikipedia.org/wiki/H-infinity_methods_in_control_theory. – *Прим. перев.*

[Knowles и Nakayama, 2008] и [Shi и Rasheed, 2010]. Сборник статей, связанных с эволюционными алгоритмами, для задач с оцениванием дорогостоящих функций приспособленности имеется в публикации [Tenne и Goh, 2010]. Книги по динамическим эволюционным алгоритмам включают [Branke, 2002], [Morrison, 2004], [Yang и соавт., 2010] и [Simões, 2011]. Юрген Бранке ведет веб-сайт, посвященный эволюционным алгоритмам для динамической оптимизации [Branke, 2012].

Аппроксимация приспособленности – это не единственный способ, который помогает справляться с дорогостоящими функциями приспособленности. Мы также можем использовать распределенные grid-вычисления, что влечет за собой применение распределенных вычислительных ресурсов, которые объединяются для решения одной задачи [Melab и соавт., 2006], [Lim и соавт., 2007]. Другие экономящие время приемы для дорогостоящих функций приспособленности включают многочисленные ядра для распараллеливания с использованием одного компьютера, кластера компьютеров, облачные вычисления и другие формы распределенной обработки [Tomassini и Vanneschi, 2009], [Tomassini и Vanneschi, 2010].

Обзоры эволюционных алгоритмов в неопределенных средах приведены в публикациях [Jin и Branke, 2005] и [Nguyen и соавт., 2012]. «Неопределенная среда» в данном контексте включает в себя дорогостоящие функции приспособленности, динамические функции приспособленности и шумные функции приспособленности, которые являются основными темами данной главы. Однако в публикации [Jin и Branke, 2005] также обсуждается робастность, которая является мерой качества эволюционно-алгоритмического решения при наличии вариаций в векторе решения либо в параметрах задачи [Eiben и Smit, 2011].

Рассмотрим робастность по отношению к вариациям в параметрах. Многие функции приспособленности могут быть записаны как $f(x, p)$, где x – это вектор решения, а p – вектор параметров. Например, если мы пытаемся оптимизировать алгоритм управления роботом, то p вполне может относиться к некоторым физическим параметрам конструкции робота. Когда мы оптимизируем $f(x, p)$, «робастность» обозначает качество $f(x, p + \Delta p)$, где Δp представляет вариации параметров. В общем случае хорошее решение часто не является робастным [Keel и Bhattacharyya, 1997]. Рисунок 21.25 иллюстрирует эту ситуацию. Минимальная стоимость достигается при $x = x_1$, но при таком значении x стоимостная функция очень чувствительна к вариациям в параметрах. Мы можем получить субоптимальную стоимость при $x = x_2$, что дает ухудшенную стоимость, но гораздо более высокую робастность по отноше-

нию к вариациям в параметрах. В зависимости от ожидаемой величины вариации в параметрах x_2 вполне может быть предпочтительнее x_1.

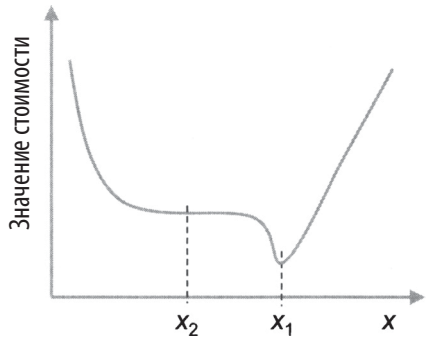


Рис. 21.25. На приведенном выше рисунке показана стоимость как функция от значения параметров p, где p^* – это номинальное значение параметров, $x = x_1$ дает лучшую стоимость при $p = p^*$, но не является робастной. $x = x_2$ дает худшую стоимость, но обеспечивает гораздо более высокую робастность по отношению к вариациям в параметрах

Мы также можем рассмотреть робастность по отношению к вариациям в векторе решения. В этом случае, когда мы оптимизируем $f(x)$, «робастность» обозначает качество $f(x + \Delta x)$, где Δx представляет вариации в векторе решения. Когда мы находим оптимальный вектор решения x, вектор решения, который мы реализуем в нашем решении, может варьироваться из-за проблем реализации, производственных неопределенностей и других проблем. Рисунок 21.26 иллюстрирует эту ситуацию. Минимальная стоимость достигается при $x = x_1$, но при этом значении x стоимостная функция очень чувствительна к вариациям в x. Мы можем получить субоптимальную стоимость при $x = x_2$, что дает ухудшенную стоимость, но гораздо более высокую робастность по отношению к вариациям в x. В зависимости от ожидаемой величины вариации x_2 вполне может быть предпочтительнее x_1. В данной главе мы не рассматривали робастность, но этот вопрос является важным в реальных задачах, и в публикациях [Jin и Branke, 2005] и [Branke, 2002, глава 8] этой теме уделено немало внимания.

По мере того как эволюционные алгоритмы применяются ко все большему числу реальных задач, исследовательские усилия в области эволюционных алгоритмов могут начать смещаться в сторону большего количества тем, обсуждаемых в этой главе, включая дорогостоящие функции приспособленности, динамические функции приспособленности и шумные функции приспособленности. Некоторые интересные

области будущих исследований включают эволюционные алгоритмы для задач, которые имеют более чем одну из этих особенностей. Например, мы обсудили алгоритмы обработки динамических функций приспособленности и алгоритмы обработки шумных функций приспособленности, но встает вопрос: как объединить эти алгоритмы для обработки функций приспособленности, которые являются динамическими и шумными? Еще один интересный вопрос: как распределять бюджет из K оцениваний приспособленности для получения лучших результатов от эволюционного алгоритма. Обратите внимание, что прилагательное «лучший» в этом контексте можно интерпретировать несколькими способами. Например, мы можем искать лучшую стратегию с точки зрения оптимизации ожидаемого результата эволюционного алгоритма, либо с точки зрения оптимизации худшего результата эволюционного алгоритма, либо с точки зрения оптимизации результата для лучшего случая минус три сигмы, либо некого их сочетания.

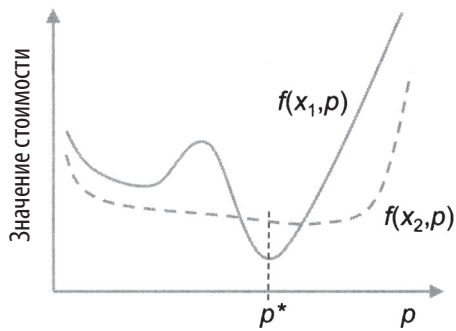


Рис. 21.26. На приведенном выше рисунке показана стоимость как функция независимой переменной x. Мы видим, что $x = x_1$ дает лучшую стоимость, но не является робастной, $x = x_2$ дает худшую стоимость, но обеспечивает гораздо более высокую робастность по отношению к вариациям в x

Задачи

Письменные упражнения

21.1. Рассмотрим одномерную функцию, состоящую из следующих трех точек (x, y): (1, 3), (2, 1) и (3, 4).

а) Какова линейная среднеквадратическая (RMS) подгонка под эту функцию? Какова среднеквадратическая и минимаксная ошибка этой аппроксимации?

b) Какова линейная минимаксная подгонка под эту функцию? Какова среднеквадратическая и минимаксная ошибка этой аппроксимации?

21.2. Выведите формулу уравнения (21.16).

21.3. В тексте говорится, что $s(x)$ в уравнении (21.24) равно нулю в выборочных точках данных. Докажите это.
(*Подсказка*: если x^* – это одна из выборочных точек данных, то r – это один из столбцов R.)

21.4. Сколько существует возможных вариантов расположения с отбором на основе латинского гиперкуба для двумерной $M \times M$-решетки?

21.5. Предположим, что шумное оценивание приспособленности x_1 расположено равномерно между –2 и 2. Предположим, что шумное оценивание приспособленности x_2 расположено равномерно между –1 и 3. Какова вероятность того, что шумное оценивание приспособленности x_1 больше, чем оценивание x_2?

21.6. Докажите следующее утверждение из раздела 21.3.1: «Если мы оцениваем функцию приспособленности для данной особи N раз и значения шума этих N проб независимы, то дисперсия средней функции приспособленности уменьшается в N раз».

21.7. Предположим, у вас есть следующие 10 оцениваний шумной функции приспособленности заданной особи:

[101, 102, 98, 97, 99, 103, 104, 101, 97, 98].

Сколько оцениваний приспособленности вам нужно в среднем, чтобы получить дисперсию 5?

21.8. В чем разница между прецизионностью и точностью? Дайте объяснение и пример.

21.9. Предположим, что мы используем эволюционный алгоритм на основе фильтра Калмана с целью оценивания приспособленности особи x. Предположим, что наша оценка приспособленности $\hat{f}(x) = 10$ с дисперсией неопределенности = 1. Предположим, что мы получаем новый замер приспособленности $g(x) = 11$ с дисперсией неопределенности = 3. Каковы новая оценка приспособленности и ее дисперсия неопределенности?

Компьютерные упражнения

21.10. Повторите пример 21.1 и 21.2 для двумерной функции Экли, при этом каждая независимая переменная должна находиться в области [–3, +3].

21.11. Примените уравнение (21.38) для оценивания результативности трех динамических эволюционно-алгоритмических подходов примера 21.6.

21.12. Напишите компьютерную программу, экспериментально подтверждающую, что среднее значение N зашумленных членов имеет дисперсию, в $1/N$ раз превышающую дисперсию каждого зашумленного члена.

21.13. Просимулируйте примеры 21.5 и 21.6, отслеживая следующие пропорции:

p_r = как часто случайные особи R являются лучшими новыми особями;

p_e = как часто мутированные элитарные особи E являются лучшими новыми особями;

p_d = как часто двойные особи D являются лучшими новыми особями,

где слово «лучший» определяется с точки зрения средней стоимости. То есть после создания новых случайных особей R, новых мутированных элитарных особей E и новых двойных особей как часто средняя стоимость R лучше средней стоимости E и средней стоимости D и т. д.? Для того чтобы сделать в разумной степени определенные выводы, используйте достаточно большое число поколений для сбора достаточного количества данных.

Часть V

Приложения

Приложение А

Несколько практических советов

Хороший совет всегда непременно будет проигнорирован, но это не причина его не давать.

– Агата Кристи

В этом приложении содержатся практические рекомендации для исследователей и практиков эволюционных алгоритмов. Что делать, если эволюционный алгоритм не работает? Как усовершенствовать работу нашего эволюционного алгоритма? Как стать успешным исследователем в области эволюционных алгоритмов? На чем мы должны сосредоточиваться в наших исследованиях? В книге «A Field Guide to Genetic Programming» («Полевой справочник по генетическому программированию») приводится много хороших практических советов [Poli и соавт., 2008, глава 13]. Эти советы специфичны для генетического программирования, но бóльшая их часть достаточно обща для того, чтобы применять их в любом другом типе эволюционных алгоритмов, и поэтому в данном приложении мы позаимствуем некоторые из этих идей.

А.1. Проверка на наличие ошибок

Многие студенты и начинающие исследователи ожидают, что их исходный код и исходный код других разработчиков заработает с первого раза. Бездефектного исходного кода не существует. Любой исходный код содержит дефекты. Мы должны понимать наш программный продукт достаточно хорошо, чтобы находить проблемы в исходном коде

либо проблемы, связанные с тем, как мы его используем. Это означает, что нам следует выполнять исходный код пошагово, по одной строке за раз, и проверять переменные с помощью отладчика. Нам нужно разбираться в том, какие значения загружаются в какие переменные и почему. Это необходимо делать, даже если исходный код выполняется в первый раз. То, что исходный код выполняется, не означает, что он работает.

Вышесказанное вовсе не означает, что стандартные программы не работают. Вышесказанное вовсе не означает, что мы должны понимать абсолютно все о каждой используемой нами компьютерной программе. Но производство программного продукта – это качественно иной характер, чем научные исследования и разработка программ. Если производственный программный продукт не работает, то мы, как правило, видим результаты неисправности, и мы можем применить изобретательность, чтобы обойти эти проблемы (например, перезагрузить компьютер или поэкспериментировать с другой последовательностью нажатий кнопок для получения желаемых результатов).

Однако если исследовательский программный продукт не работает, то нам обычно приходится исправлять программу самим. Для этого нам нужно разбираться в программном продукте. Что еще более важно, даже если программный продукт работает, то он вполне может работать неправильно. Если мы не разбираемся в программном продукте, то никогда не будем уверены, что он работает правильно, и никогда не сможем доверять нашим результатам.

В эволюционно-алгоритмических исследованиях (или в любых других инженерных исследованиях) нет коротких путей. Всегда лучше написать собственный программный продукт. Если мы хотим использовать чужое программное обеспечение, то должны изучить его исходный код и хорошо в нем разбираться. В противном случае мы, вероятно, получим то, за что платим, в особенности если программное обеспечение бесплатное.

А.2. Эволюционные алгоритмы являются стохастическими

Эволюционные алгоритмы являются стохастическими. Это означает, что при каждом прогоне они не будут давать одинаковые результаты. Если мы выполним эволюционный алгоритм 10 раз, чтобы попытаться решить задачу, и он не будет работать ни один из этих 10 раз, то мы вполне можем наивно заключить, что эволюционный алгоритм не ра-

ботает или что задача не разрешима. Однако если у эволюционного алгоритма есть 20%-ный шанс решить задачу, то шанс, что он завершится безуспешно 10 раз подряд, составляет 10 %.

И наоборот, если мы выполним его 10 раз и он закончится успешно 10 раз подряд, то мы вполне можем наивно заключить, что эволюционный алгоритм в каждом случае будет работать. Однако если у эволюционного алгоритма есть 20%-ный шанс завершиться безуспешно, то шанс, что он завершится успешно 10 раз подряд, составляет 10 %. Для того чтобы глубоко понимать стохастическую природу эволюционных алгоритмов и вытекающих из нее последствий, необходимо хорошее понимание теории вероятностей.

А.3. Небольшие изменения могут иметь большой эффект

Такие безобидные вещи, как изменение способа, которым особи дублируются, или небольшие изменения в скорости мутации, или изменение метода отбора, могут радикально изменить работу эволюционного алгоритма. Мы никогда не должны исходить из того, что небольшое изменение не имеет значения. Это означает, что когда мы получаем хорошие результаты, мы должны бережно относиться к тому, чтобы сохранять настройки эволюционного алгоритма, которые дали нам эти результаты. Более того, для того чтобы мы могли воспроизвести наши результаты, мы даже должны сохранить посев случайного числа (см. приложение В.2.3). Если мы получим хорошие результаты и начнем возиться с настройками параметров, для того чтобы улучшить результативность, то вполне можем потерять наши хорошие результаты навсегда, в случае если забудем настройки параметров, которые использовали для получения этих хороших результатов.

А.4. Большие изменения могут иметь малый эффект

В отличие от вышеизложенного, иногда эволюционный алгоритм нечувствителен к большим изменениям параметров. Хорошо, если эволюционный алгоритм работает хорошо, потому что это означает, что эволюционный алгоритм робастен к этому параметру. Но нечувствительность к изменениям параметров плоха, если эволюционный алгоритм работает плохо. Если эволюционный алгоритм работает плохо, то

это вполне может быть связано с тем, что задача для эволюционного алгоритма слишком сложна, либо потому, что задача не была сформулирована таким образом, который поддается эволюционному алгоритму. Нам не гарантируется, что эволюционный алгоритм будет давать хорошие результаты, поэтому мы не должны исходить из того, что эволюционный алгоритм непременно будет успешным, если мы просто отыщем правильные настройки параметров. Вот где следует уравновешивать нашу настойчивость с возможностью того, что мы вполне просто тратим наше время впустую. Тем не менее большинство из нас ошибается, находясь на стороне слишком малой настойчивости, нежели слишком большой.

А.5. Популяции имеют много информации

Если наш эволюционный алгоритм работает не очень хорошо, то мы должны изучить популяцию в разных поколениях. Состав популяции может дать нам много информации. Если мы видим, что популяция сходится к единственному кандидатному решению, то понимаем, что нам нужно тщательнее заняться дублирующими особями. Если мы видим, что рекомбинирующие особи не улучшаются, то понимаем, что нам нужно изменить нашу стратегию рекомбинации. Если мы отслеживаем мутации, то понимаем, мутируем ли мы особей слишком много/мало или слишком часто/редко. Если мы видим, что популяция не улучшается после первых нескольких поколений, то понимаем, что нам нужно больше заняться разведыванием и меньше – эксплуатацией. Количество подробностей о поведении эволюционного алгоритма, которые мы можем почерпнуть из изучения его популяции, ограничено только нашей изобретательностью и упорством.

А.6. Поощрение многообразия

Это связано с вышеуказанным пунктом. Мы должны отдавать себе полный отчет о наличии многообразия (или его отсутствии) в нашей популяции. Если наша популяция не будет иметь достаточного многообразия, то она, по всей видимости, не будет хорошо работать. Мы должны делать все необходимое для поощрения многообразия, не препятствуя при этом эксплуатации хороших кандидатных решений. Мы должны помнить, что, для того чтобы эволюционный алгоритм был успешным, нам нужно найти только одно или несколько хороших решений. Большее многообразие увеличивает наши шансы на успех.

А.7. Использование специфичной на конкретной задаче информации

Чем лучше мы поймем нашу задачу, тем более качественное решение сможем найти. Эволюционный алгоритм представляет собой, как правило, безмодельный оптимизатор, а значит, нам не нужно встраивать в него какую-либо специфичную для конкретной задачи информацию. Однако если мы все-таки встроим специфичную для конкретной задачи информацию, то мы почти наверняка сможем найти более качественное решение, чем могли бы в противном случае. Эволюционный алгоритм является хорошим глобальным оптимизатором, но метод локального поиска, такой как восхождение к вершине холма или градиентный спуск, может улучшить результаты эволюционного алгоритма до такой степени, что он докажет разницу между неудачей и успехом.

А.8. Сохраняйте свои результаты как можно чаще

Место на компьютерном диске дешевое. Нам следует сохранять наши настройки задачи, старые версии нашего программного продукта, окончательные и промежуточные результаты. Это означает, что мы должны организовать свою работу так, чтобы могли эффективным образом пробираться сквозь все наши сохраненные программы и данные, когда мы ищем то, что нам нужно, не тратя на это слишком много времени.

Прогоны эволюционных алгоритмов зачастую имеют большую вычислительную емкость. Это означает, что, для того чтобы получить хорошие результаты, нам, возможно, придется работать в течение нескольких дней или недель. Эффективный и организованный способ сделать это состоит в том, чтобы выполнять эволюционный алгоритм в течение дня, сохранять наши результаты, а затем снова начинать работу с новой популяцией, посеянной предыдущими результатами. Это позволяет эволюционному алгоритму пользоваться ранее обнаруженными хорошими кандидатными решениями, продолжая работать в течение длительного периода времени.

А.9. Понимание статистической значимости

Мы должны понимать статистическую значимость наших экспериментальных результатов. Это означает, что нам нужно понимать статистику и теорему об отсутствии бесплатных обедов (см. приложение В). Это

также означает, что нам нужно проверять результаты нашего эволюционного алгоритма на множестве проверочных данных. Эволюционный алгоритм можно использовать для поиска хорошего решения оптимизационной задачи, но не менее важно выполнять проверку результативности решения на данных, которые не использовались во время тренировки. Например, мы вполне можем применить эволюционный алгоритм для поиска параметров, оптимизирующих классификатор, и у нас есть некоторые тестовые данные, которые мы используем для оценивания функции приспособленности. Однако если мы на этом остановимся, то не сделаем ничего, кроме как покажем, что классификатор может запоминать тренировоные данные. Реальная проверка классификатора заключается в том, насколько хорошо он работает с данными, которые он еще не встречал. Эти данные называются проверочным множеством, или набором. Существует несколько способов разделения данных на тренировочные и проверочные. В данной книге мы эти идеи не обсуждаем, за исключением короткого обсуждения в разделе 21.1.6, но отметим, что очень важно хорошо понимать основные идеи контрольной проверки [Hastie и соавт., 2009].

А.10. Пишите хорошо

Если мы проводим лучшее эволюционно-алгоритмическое исследование в мире, но не сообщаем об этом другим, то такое исследование будет бесполезным. Исследование выполняется для коммуникации. Великий английский ученый Майкл Фарадей сказал: «Работай, завершай, публикуй» [Beveridge, 2004, page 121], подразумевая, что процесс исследования не завершен до тех пор, пока результаты не будут опубликованы. Составление технической документации – это навык, которого катастрофически не хватает сегодняшним студентам, инженерам и исследователям. Мы станем более качественными профессионалами, если сможем на письме оформлять нашу работу лучше. Научиться писать лучше – это не какой-то загадочный процесс. Мы учимся писать лучше точно так же, как учимся делать лучше что-то еще: мы изучаем чистописание и потом практикуемся в нем.

А.11. Акцент на теории

Слишком много эволюционно-алгоритмических исследований сегодня включает в себя регулировку параметров и гибридизацию эволюционных алгоритмов с другими оптимизационными алгоритмами. Учитывая

все доступные сегодня эволюционные алгоритмы, все их регулировочные параметры и все другие алгоритмы неэволюционной оптимизации, существует практически бесконечное число способов, которыми эволюционные алгоритмы могут быть модифицированы, объединены и отрегулированы для повышения результативности на эталонах. Но такого рода исследования на самом деле не продвигают современное состояние дел. Такого рода исследования носят исключительно инкрементный и недальновидный характер и не дают никаких результатов, выходящих за рамки их непосредственных результатов. Большинству эволюционных алгоритмов катастрофически не хватает теоретического обеспечения и математического анализа. Если бы в наших исследованиях мы могли уделять больше внимания теории и математике, то смогли бы начать вносить перемены в наше фундаментальное понимание эволюционных алгоритмов. Одна хорошая теоретическая работа стоит дюжины работ по регулировке параметров.

А.12. Акцент на практике

Слишком много эволюционно-алгоритмических исследований сегодня сфокусировано на эталонах. Но если мы хотим, чтобы наши исследования изменили мир к лучшему, то нам нужно сотрудничать с промышленностью и решать задачи, которые важны для практикующих инженеров и других специалистов. Наши эволюционные алгоритмы могут потратить оставшуюся часть своего существования на оптимизацию эталонов, ни разу не выходя за пределы университета. Однако эволюционные алгоритмы существуют не для того, и не для этого они были изначально изобретены [Fogel и соавт., 1966], [Fogel, 1999]. Они были изобретены для решения реальных задач и внесения вклада в жизнь общества. Эталоны имеют важное значение, но являются средством для достижения конечной цели, при этом конечной целью является разработка эволюционных алгоритмов для последующего их применения в реальном мире. Давайте не будем путать средства и цели и не будем забывать о нашей конечной цели в эволюционно-алгоритмических исследованиях.

Приложение B

Теорема об отсутствии бесплатных обедов и тестирование результативности

Вполне можно ожидать, что есть пары поисковых алгоритмов A и B – такие, что A в среднем лучше B... Одним из основных результатов этой работы является то, что такие ожидания неверны.

– Дэвид Вулперт и Уильям Макриди
[Wolpert и Macready, 1997, стр. 67]

Существует три вида лжи: ложь, наглая ложь и статистика.

– Марк Твен [Twain, 2010, стр. 228]

В настоящем приложении рассматриваются два отдельных, но связанных между собой вопроса. Эта глава является критически важной для прочтения и понимания эволюционных алгоритмов студентами и исследователями. Данный материал приведен в приложении, потому что оно не имеет прямого отношения к эволюционным алгоритмам и потому что мы должны разбираться в эволюционных алгоритмах и симуляциях, прежде чем читать это приложение, однако его отнесение к приложениям не умаляет его важности.

В разделе B.1 рассматривается теорема об отсутствии бесплатных обедов (no free lunch theorem, NFL) в интуитивно понятном, нематема-

тическом аспекте. Теорема об отсутствии бесплатных обедов говорит нам, что все алгоритмы работают одинаково хорошо при определенных условиях. В разделе В.2 рассказывается о том, как результаты эволюционного-алгоритмического симулирования нередко представляются неверно и как они могут быть правильно представлены. Раздел В.3 содержит примечание о взаимосвязи используемых в данной книге методологий и обсуждаемых в этом приложении исследовательских принципов.

В.1. Теорема об отсутствии бесплатных обедов

Теорема об отсутствии бесплатных обедов, которая была впервые формализована Дэвидом Вольпертом и Уильямом Макриди [Wolpert и Macready, 1997], просто удивительна:

> *Все алгоритмы оптимизации работают одинаково хорошо при усреднении по всем возможным задачам.*

Это значит, что в общем случае ни один оптимизационный алгоритм не лучше любого другого и ни один не хуже любого другого. Обратите внимание, что теорема об отсутствии бесплатных обедов – это больше, чем гипотеза, или общее утверждение, или эмпирическое правило; это математическая теорема. Для определенных типов задач некоторые алгоритмы будут работать лучше, чем другие. Но данная теорема должна препятствовать нам в том, чтобы мы делали необоснованные заявления о нашем любимом алгоритме; она должна побуждать нас быть скромнее в наших заявлениях.

Технически теорема об отсутствии бесплатных обедов применима только к задачам дискретной оптимизации. Вместе с тем все вещественные функции приспособленности однажды когда-нибудь дискретизируются. В конце концов, мы определяем кандидатные решения и измеряем приспособленность с помощью цифровых компьютеров. Поэтому, по существу, слово «дискретный» из теоремы об отсутствии бесплатных обедов можно удалить.

Интуитивно мы рассчитываем, что некоторые оптимизационные алгоритмы в среднем лучше, чем другие алгоритмы. Например, рассмотрим алгоритм A, являющийся алгоритмом спуска с вершины холма и аналогичный алгоритмам, описанным в разделе 2.6; и алгоритм B, который является генератором случайных чисел. Предположим, что оба

алгоритма пытаются найти минимум некой функции $f(x)$. Если $f(x) = x^2$, то спуск с вершины холма обычно намного лучше, чем случайный поиск. Вместе с тем случайный поиск иногда, чисто случайно, будет генерировать число, которое очень близко к минимуму, и в этих случаях он будет работать лучше, чем A.

Мы можем утверждать, что существует бесконечное число гладких функций, похожих на $f(x) = x^2$, и спуск с вершины холма будет довольно хорошо работать на таких функциях, тогда как случайный поиск обычно хорошо работать не будет. Однако, как показано в следующем далее примере, также существует бесконечное число нерегулярных функций, для которых случайный поиск будет работать лучше, чем спуск с вершины холма.

■ Пример 2.1

Рассмотрим дискретную функцию на рис. В.1. Эта конкретная функция имеет поисковое пространство размером 10 и была сгенерирована случайно. Мы можем реализовать алгоритм спуска с вершины холма для нахождения минимума этой функции. Мы инициализируем поиск в случайной точке поискового пространства, и алгоритм проверяет соседние значения, чтобы решить, в каком направлении продолжить свой поиск. Если алгоритм застрял в локальном, неглобальном минимуме $x \in \{4, 7, 10\}$, то он перезапускается в случайном исходном значении. Для нахождения минимума алгоритму спуска с вершины холма требуется в среднем около 19 оцениваний функции, в то время как для случайного поиска требуется в среднем ровно 10 оцениваний функции. Причина, почему спуск с вершины холма занимает так много времени, заключается в том, что для нахождения глобального минимума он должен начинаться с $x \in [1, 3]$; в противном случае он сойдется к локальному минимуму, и ему придется реинициализироваться с новой случайной отправной точкой. Любая отправная точка, отличная от $x \in [1, 3]$, приведет к тому, что алгоритм спуска с вершины холма будет тратить время в локальной окрестности, что не приведет к глобальному минимуму.

■

Существует бесконечное число нерегулярных функций, подобных показанной на рис. В.1, для которых случайный поиск будет работать лучше, чем спуск с вершины холма. Для каждой регулярной функции, для которой спуск с вершины холма работает лучше, мы можем найти соответствующую нерегулярную функцию, для которой случайный поиск работает лучше. Мы видим, что усредненно по всем возможным

функциям спуск с вершины холма и случайный поиск работают одинаково хорошо.

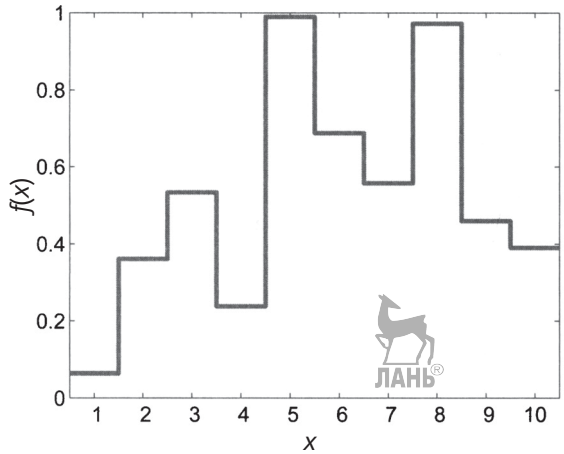


Рис. В.1. Пример 2.1: для того чтобы найти глобальный минимум при $x = 1$ для этой минимизационной задачи, алгоритм спуска с вершины холма требует в среднем 19 оцениваний функции, в то время как случайный поиск требует в среднем только 10 оцениваний функции

Неудивительно, что нерегулярную функцию, подобную функции на рис. В.1, можно минимизировать эффективнее с помощью случайного поиска, чем с помощью алгоритма спуска с вершины холма. Что более удивительно, так это то, что теорема об отсутствии бесплатных обедов утверждает, что для минимизации функции алгоритм *восхождения* к вершине холма в среднем работает так же хорошо, как и алгоритм *спуска* с вершины холма. Если мы хотим найти минимум функции, то следует ли нам применять алгоритм спуска с вершины холма или алгоритм восхождения к вершине холма? Здравый смысл подсказывает, что если мы хотим найти минимум, то нам следует использовать алгоритм спуска с вершины холма. Тем не менее теорема об отсутствии бесплатных обедов говорит нам, что алгоритм восхождения к вершине холма будет работать так же хорошо, усредненно по всем возможным функциям, как показано на следующем далее примере.

■ Пример 2.2

Рассмотрим дискретную функцию, показанную на рис. В.2. Её можно считать обманчивой функцией, потому что локальные изменения в функции стоимости приводят нас к ожиданию того, что минимум должен быть на одном из экстремальных значений x. Алгоритм спуска с верши-

ны холма использует в своем поиске локальную информацию и поэтому обычно будет застревать в $x = 1$ или $x = 11$, и затем ему придется начинать сначала. Единственный раз, когда алгоритм спуска с вершины холма добьется успеха, просходит в случае, если он инициализируется в точке $x \in [5, 7]$. С другой стороны, алгоритм восхождения к вершине холма будет всегда подниматься до $x = 5$, если он стартует в $x < 5$, и он будет всегда подниматься до $x = 7$, если стартует в $x > 7$. После того как он достигнет $x = 5$ или $x = 7$, он наткнется на глобальный минимум на следующей итерации, когда будет искать восходящее направление. Алгоритму восхождения на вершине холма для нахождения глобального минимума никогда не потребуется более 6 оцениваний функции. Для того чтобы найти глобальный минимум, алгоритму спуска с вершины холма требуется в среднем около 16 оцениваний функции, случайному поиску нужно в среднем 11 оцениваний функции, восхождению к вершине холма – в среднем только 5 оцениваний функции. Мы можем придумать бесконечное число функций с той же топологией, что и функция, показанная на рис. B.2. Несмотря на то что применение алгоритма восхождения к вершине холма для минимизации функции противоречит нашему здравому смыслу, мы видим, что восхождение к вершине холма работает так же хорошо, как и спуск с вершины холма, усредненно по всем возможным функциям.

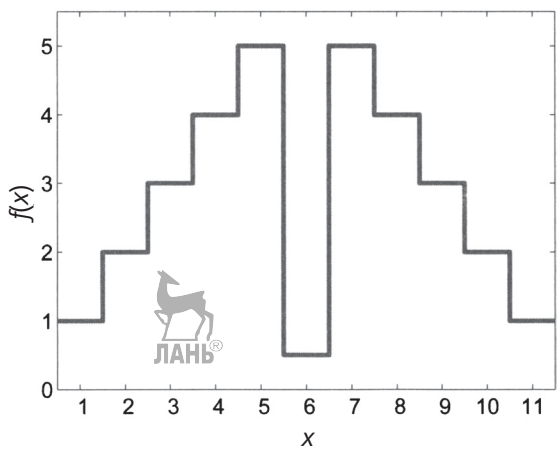


Рис. B.2. Пример 2.2: для нахождения глобального минимума в $x = 6$ для данной минимизационной задачи алгоритм спуска с вершины холма требует в среднем 16 вычислений функции, алгоритм случайного поиска – в среднем 11 вычислений функции, алгоритм восхождения к вершине холма – в среднем лишь 5 вычислений функции. В случае этой минимизационной задачи восхождение к вершине холма справляется с минимизацией лучше, чем спуск с вершины холма

■

■ Пример 2.3

Рассмотрим тонко отрегулированный генетический алгоритм, запрограммированный для решения некой сложной реальной оптимизационной задачи. Мы думаем, что генетический алгоритм должен работать лучше, чем решения, полученные в результате случайного гадания случайным человеком. Но предположим, что вы выходите на улицу и просите первого встречного выбрать число. Число, которое выбирает этот человек, является решением некой оптимизационной задачи. По сути дела, это число является решением бесконечного числа оптимизационных задач. Конкретную задачу, которую мы пытаемся решить, будем надеяться, лучше всего атаковать с помощью генетического алгоритма, чем с помощью генератора случайных чисел. Но усредненно по всем возможным задачам случайное гадание дает такой же хороший результат, что и генетический алгоритм.

■

Теорему об отсутствии бесплатных обедов можно уточнить, задав очень простые вопросы о множестве всех возможных функций, множестве всех возможных оптимизационных алгоритмов и множестве всех возможных поведений оптимизационных алгоритмов [Whitley и Watson, 2005]. Такой подход к пониманию данной теоремы иллюстрируется следующим далее примером.

■ Пример 2.4

Рассмотрим простую минимизационную задачу, имеющую область $x \in [1, 3]$. Предположим, что мы инициализируем все поисковые алгоритмы значением $x = 1$. Существует бесконечное число функций – таких, что $f(1) > f(2) > f(3)$, и существует одинаково бесконечное число функций – таких, что $f(1) > f(3) > f(2)$. Следовательно, существует взаимно однозначное соответствие между числом функций, которые минимизируются при $x = 3$, и числом функций, которые минимизируются при $x = 2$.

Теперь рассмотрим множество всех оптимизационных алгоритмов A, исследующих $x = 2$ в качестве первого (после инициализации) шага в процессе поиска. Обозначим через $\overline{A}$ множество всех оптимизационных алгоритмов, исследующих $x = 3$ в качестве первого шага. Поскольку существует взаимно однозначное соответствие между числом функций, которые минимизируются при $x = 3$, и числом функций, которые минимизируются при $x = 2$, мы видим, что $\overline{A}$ найдет минимум до A ровно для

половины из всех возможных функций, тогда как A найдет минимум до $\overline{A}$ для другой половины всех возможных функций.

■

Приведенные выше соображения не содержат доказательства теоремы об отсутствии бесплатных обедов; они больше похожи на объяснения или иллюстрации и дают нам интуитивное представление о том, почему теорема об отсутствии бесплатных обедов верна. Строгие математические доказательства данной теоремы представлены в публикациях [Radcliffe и Surry, 1995] и [Wolpert и Macready, 1997]. Теорему об отсутствии бесплатных обедов можно сформулировать разными способами, многие из которых эквивалентны. Среди прочих по существу эквивалентных утверждений теорема об отсутствии бесплатных обедов утверждает следующее [Schumacher и соавт., 2001].

- Все оптимизационные алгоритмы работают одинаково хорошо при усреднении по всем возможным задачам (как указано в начале этого приложения).
- В случае любых двух оптимизационных алгоритмов A и B, если результативность A на задаче f задается выражением $V(A, f)$, существует задача g – такая, что $V(A, f) = V(B, g)$. Это утверждение говорит нам о том, что независимо от того, насколько хорошо алгоритм A выполняет какую-либо задачу, существует еще один алгоритм B, который одинаково хорошо выполняет какую-либо другую задачу. И наоборот, независимо от того, насколько плохо алгоритм B выполняет какую-либо задачу, существует еще одна задача, для которой алгоритм A работает так же плохо.
- Алгоритм, достигающий результативности, которая лучше, чем случайный поиск, подобен вечному двигателю [Schaffer, 1994]. Это утверждение называется законом сохранения эффективности обобщения.
- Бесполезно пытаться разрабатывать оптимизационный алгоритм, который будет лучше, чем случайный поиск, если только вы не сможете встроить в алгоритм специфичную для конкретной задачи информацию [English, 1999]. Это утверждение называется законом сохранения информации.
- В отсутствие специфичной для конкретной задачи информации мы должны исходить из того, что все возможные решения одинаково вероятны [Dembski и Marks, 2009b]. Это утверждение называется принципом недостаточного основания Бернулли.

Теорема об отсутствии бесплатных обедов должна придать нам некоторое равновесие в наших заявлениях. Например, предположим, что мы разрабатываем новый алгоритм или улучшенную версию какого-либо алгоритма, и мы хотим показать, что он лучше других алгоритмов, с целью опубликования наших результатов. Предположим, что у нас есть множество эталонных функций F. Мы используем $\overline{F}$ для обозначения дополнения F, то есть $\overline{F}$ – это все другие функции, кроме F. Если наш любимый алгоритм, алгоритм A, работает лучше, чем алгоритм B, на множестве эталонных функций F, то теорема об отсутствии бесплатных обедов убеждает нас в том, что алгоритм B работает лучше на $\overline{F}$.

До сих пор мы не говорили о том, как оценивать результативность. Существует ряд способов измерения результативности алгоритма. Например:

- результативность можно измерить лучшим решением, полученным после определенного числа поколений и определенного числа симуляций. Эту меру можно назвать *лучший из лучших*;
- результативность можно измерить средним числом лучших решений, полученных с помощью определенного числа симуляций, каждое из которых выполняется в течение определенного числа поколений. Эту меру можно назвать *среднее число лучших*;
- результативность можно измерить путем вычисления средней приспособленности всех особей в популяции после определенного числа поколений и нахождения лучшего из этих средних значений после определенного числа симуляций. Эту меру можно назвать *лучший из средних значений*;
- результативность можно измерить путем вычисления средней приспособленности всех особей в популяции после определенного числа поколений и нахождения среднего значения этих средних значений приспособленности после определенного числа симуляций. Эту меру можно назвать *средним числом средних*;
- результативность можно измерить стандартным отклонением лучших решений, найденных после определенного числа поколений и определенного числа симуляций;
- любая из вышеперечисленных мер результативности может быть скомбинирована любым способом для формирования гибридных мер результативности;
- любая из вышеуказанных мер результативности может быть усреднена по нескольким задачам.

По-видимому, существует столько же способов квантификации результативности, сколько и оптимизационных задач. На самом деле их бесконечное множество.

Это подводит нас к еще одной важной особенности теоремы об отсутствии бесплатных обедов: данная теорема применима независимо от того, как мы измеряем результативность. Иногда алгоритм рекламируется как более робастный, чем другие алгоритмы. Термин «робастность» вполне может означать, что алгоритм имеет хорошую результативность по широкому спектру функций или что алгоритм относительно нечувствителен к шуму оценивания функции приспособленности, вариациям параметров функции приспособленности или вариациям параметров алгоритма. Теорема об отсутствии бесплатных обедов гарантирует нам, что все алгоритмы одинаково робастны. И наоборот, она также говорит нам, что все алгоритмы одинаково специализированы [Schumacher и соавт., 2001]. Если алгоритм A робастнее алгоритма B на функции F, то алгоритм B робастнее алгоритма A на функции $\overline{F}$.

Мы видим, что в некотором смысле теорема об отсутствии бесплатных обедов противоречит интуиции. Но если мы посмотрим на нее под другими углами, то она весьма интуитивна. С учетом этого второго, интуитивного взгляда на теорему об отсутствии бесплатных обедов, неудивительно, что она появляется в литературе задолго до ее формальных доказательств в середине 1990-х годов. Например, Грегори Роулинс писал [Rawlins, 1991, стр. 7]:

> *...иногда высказывается предположение, что генетические алгоритмы универсальны в том, что их можно использовать для оптимизации любых функций. Эти утверждения верны только в очень ограниченном смысле; от любого алгоритма, удовлетворяющего одному из этих утверждений, можно ожидать результативности не лучшей, чем у случайного поиска по пространству всех функций.*

Реакции на теорему об отсутствии бесплатных обедов со стороны исследователей оптимизации варьировались. Некоторые говорят, что данная теорема не представляет практического интереса из-за ее условия «усредненно по всем возможным задачам». В реальном мире (в отличие от мира математической теории) нас интересуют не все возможные задачи, а задачи, возникающие в практических приложениях. Реальные задачи обычно не случайны или обманчивы, а имеют некоторую структуру. Это означает, что алгоритм спуска с вершины холма будет работать лучше, чем случайный поиск либо алгоритм восхождения

к вершине холма в реальных минимизационных задачах; и, по сути, это именно то, что мы обычно наблюдаем. В результате получаем инь для ян теоремы об отсутствии бесплатных обедов:

> *Не все оптимизационные алгоритмы работают одинаково хорошо при усреднении по всем интересующим задачам.*

Но это не значит, что теорема об отсутствии бесплатных обедов не имеет отношения к практикующим инженерам. Данная теорема обеспечивает прочную основу для того, что мы думаем и что знаем в интуитивном плане. То есть, для того чтобы найти хорошее решение оптимизационной задачи, нам нужно встроить в наш поисковый алгоритм специфические знания. Так мы «платим за обед» и получаем более высокую результативность, чем, скажем, случайный поиск. Если мы знаем, что хорошие решения данной задачи, как правило, группируются вместе (то есть поисковое пространство имеет некоторую регулярность), то мы знаем, что рекомбинация будет иметь тенденцию работать хорошо. Если у нас есть задача, которая имеет менее регулярное поисковое пространство, то мы знаем, что мутация будет представлять бóльшую важность.

Например, предположим, что мы пытаемся найти пропорционально-интегрально-дифференциальные (PID) управляющие параметры для оптимизации результативности управляющей системы. Мы можем разработать поиск, который вычисляет градиентную информацию (то есть чувствительность результативности к параметрам PID) и использует эту информацию в своем поиске. Это означает, что мы встраиваем специфичную для конкретной задачи информацию в наш поиск. Этот алгоритм регулировки PID избегает теоремы об отсутствии бесплатных обедов, потому что градиентная информация недоступна почти для всех функций. Мы платим за обед нашей градиентной информацией, поэтому вышли за рамки данной теоремы, и наш эволюционный алгоритм, вероятно, будет лучше, чем случайный поиск. Еще один способ констатировать это – сказать, что теорема об отсутствии бесплатных обедов рассматривает процесс оптимизации как слепой поиск без любой специфичной для конкретной задачи информации, встроенной в поиск. Но эффективные реальные поисковые алгоритмы не слепы, то есть они все-таки встраивают специфичную для конкретной задачи информацию [Culberson, 1998].

Еще один пример – инверсия в задаче коммивояжера (см. раздел 18.4.1). Многочисленные приложения инверсии гарантируют марш-

рут без пересеченных ребер. Следовательно, инверсия бóльшую часть времени проводит на маршрутах, которые лучше средних.

Теорема об отсутствии бесплатных обедов также косвенно говорит нам, почему представление задачи имеет важность (см. раздел 8.3). Если поисковое пространство задачи имеет регулярную структуру, то мы должны представить эту задачу таким образом, чтобы сохранить регулярность поискового пространства; тогда мы можем получить хорошие результаты со структурированным поиском. Если мы представим задачу таким образом, что представление не имеет структуры, то мы с тем же успехом могли бы использовать случайный поиск.

Уитли и Уотсон дают следующие практические следствия теоремы об отсутствии бесплатных обедов [Whitley и Watson, 2005].

- Конструирование оптимизационного алгоритма подразумевает компромисс между общностью и эффективностью для конкретной задачи. Алгоритм, который хорошо работает на широком диапазоне эталонов, может плохо работать для отдельной реальной задачи. И наоборот, алгоритм, который плохо работает на стандартных эталонах, может давать хорошие результаты для реальной задачи. Простые алгоритмы часто дают хорошие результаты; сложные алгоритмы могут быть спроектированы давать более высокие результаты для заданной задачи. Мы должны решить, сколько времени и усилий хотим потратить на то, чтобы отрегулировать наш алгоритм для заданной задачи, и насколько важно достичь лишь немногим более высокого результата оптимизации.
- Если мы встроим в наш оптимизационный алгоритм специфичную для конкретной задачи информацию, то сможем получить более высокие результаты, чем при использовании более общего алгоритма (при условии правильного использования специфичной для конкретной задачи информации). Это пример общего принципа, который инженеры и ученые подметили уже давно. Представление и общность вступают во взаимный компромисс. Инструменты и алгоритмы, которые предназначены для работы на широком спектре задач, в конечном итоге не работают хорошо на любых задачах.
- Представление оптимизационной задачи может оказать существенное влияние на результативность оптимизационного алгоритма. Существует бесконечное число способов представить любую оптимизационную задачу. Выбор подходящего представления может потребовать много работы, прежде чем алгоритм

будет реализован, но он может принести высокие дивиденды в долгосрочной перспективе (см. раздел 8.3).

- Не исходить из того, что оптимизационный алгоритм, который хорошо работает на эталонах, будет хорошо работать на реальных задачах; аналогичным образом не исходить из того, что алгоритм, который плохо работает на эталонах, будет плохо работать на реальных задачах. Данное следствие теоремы об отсутствии бесплатных обедов ставит под сомнение практическую важность большинства публикаций по эволюционным алгоритмам из-за их акцента на эталонных задачах и отсутствия в них акцента на реальных задачах.

Текущие исследования в области теорем об отсутствии бесплатного обеда включают в себя поиск множеств функций, на которые данная теорема не распространяется. Напомним, что теорема об отсутствии бесплатных обедов распространяется на все возможные оптимизационные задачи; она не распространяется на все множества оптимизационных задач. Например, при решении задач с одним минимумом алгоритм спуска с вершины холма явно будет лучше, чем случайный поиск. Менее интуитивным является результат, что данная теорема не распространяется на множество функций, которые могут быть описаны полиномами одной переменной с ограниченной сложностью [Christensen и Oppacher, 2001]. Кроме того, теорема об отсутствии бесплатных обедов не распространяется на некоторые типы коэволюционных задач [Wolpert и Macready, 2005].

Нахождение других функциональных множеств, на которые теорема об отсутствии бесплатных обедов не распространяется, и нахождение алгоритмов, которые обеспечивают бесплатный обед на этих множествах, могут иметь важные последствия для разработки практических оптимизационных алгоритмов. Эти вопросы связаны с такими проблемами, как сжимаемость функций, длина описания задач и различие между бесконечными множествами функций и конечными множествами функций (то есть замыканиями перестановок) [Schumacher и соавт., 2001], [Lattimore и Hutter, 2011].

В.2. Тестирование результативности

В данном разделе обсуждаются некоторые вопросы, связанные со статистикой и представлением результатов эволюционно-алгоритмического симулирования в дипломных работах, диссертациях и техни-

ческих исследованиях. Каждый, кто публикует или изучает работы в области эволюционно-алгоритмических исследований, должен иметь четкое представление об идеях этого раздела. В этом разделе показано, как сокращать тенденциозное смещение при подаче наших результатов. В нем также показано, как распознавать тенденциозность в результатах других авторов. Возможно, самое главное – в нем подчеркивается, что нам нужно иметь здоровый скептицизм в отношении других, когда мы читаем их научные работы, и здоровый скептицизм в отношении себя, когда мы документируем наши собственные результаты исследований.

В разделе В.2.1 дается обзор некоторых проблем, которые мы обычно встречаем в эволюционно-алгоритмических исследовательских работах. В разделе В.2.2 показано, как авторы нередко представляют результаты симулирования, чтобы подчернуть любой момент, который им нужен, то есть как они сообщают о результатах дезориентирующим образом (надеюсь, непреднамеренно). В нем также показано, как вместо этого можно подавать результаты симулирования в ясном и беспристрастном виде. В разделе В.2.3 отмечается несколько важных моментов о генераторах случайных чисел, которые мы используем в наших симуляциях. В разделе В.2.4 дается обзор статистической проверки на основе *t*-статистики, которая может сказать нам, являются ли статистически значимыми различия между двумя наборами результатов симулирования. В разделе В.2.5 дается обзор статистической проверки на основе *F*-статистики, которая может сказать нам, являются ли статистически значимыми различия между более чем двумя наборами результатов симулирования.

В.2.1. *Преувеличения на основе результатов симуляций*

Цитата Марка Твена в начале этой главы, взятая из книги «How to Lie with Statistics» (Как лгать с помощью статистики. М.: Альпина Паблишер, 2015.) [Huff и Geis, 1993], и данный раздел говорят нам, по существу, то же самое об эволюционно-алгоритмических исследованиях. Когда мы видим результаты эволюционно-алгоритмического исследования на бумаге или в книге, всегда присутствует некоторая тенденциозность. Тенденциозность может быть преднамеренной или непреднамеренной; она может быть явной или неявной; она может быть очевидной или утонченной; она может быть незначительной или может приводить к совершенно неправильным выводам; но так или иначе всегда присутствует некоторая тенденциозность. Когда мы читаем статью и

делаем выводы из результатов эволюционно-алгоритмического исследования, должны помнить, что сделали бы другие выводы, если бы автор решил представить другой набор результатов или даже если бы он решил представить те же результаты по-иному.

С учетом теоремы об отсутствии бесплатных обедов, в которой говорится, что все оптимизационные алгоритмы работают одинаково хорошо при усреднении по всем возможным оптимизационным задачам, кажется бесполезным проверять наши алгоритмы на эталонных задачах. Как писал Даррелл Уитли (Darrell Whit ley), «с теоретической точки зрения сравнительное оценивание поисковых алгоритмов является опасным, если не сомнительным, предприятием» [Whitley и Watson, 2005, стр. 333]. Если мы напишем статью об оптимизационном алгоритме A и покажем, что он работает лучше, чем алгоритм B на множестве эталонных функций F, то теорема об отсутствии бесплатных обедов гарантирует нам, что B работает лучше, чем A на множестве функций $\overline{F}$.

Однако это не должно нас слишком обескураживать. Если цель нашей работы – показать превосходство A над B на множестве F, то данная статья будет иметь успех. Помните, что теорема об отсутствии бесплатных обедов не говорит, что все алгоритмы работают одинаково хорошо на всех множествах задач F; она говорит, что все алгоритмы работают одинаково хорошо при усреднении по всем возможным задачам. Это означает, что алгоритм A действительно вполне мог бы работать лучше, чем B для конкретных задач или для конкретных типов задач.

Все это говорит о том, что эталонное тестирование представляет собой стоящее усилие, но мы должны помнить теорему об отсутствии бесплатных обедов, чтобы модулировать выводы, которые делаем. Типичная статья по эволюционным алгоритмам демонстрирует, что A работает лучше, чем B на множестве эталонов F, а затем делает вывод что-то вроде:

> *Поэтому мы видим, что А лучше, чем В, для оптимизации функций.*

Подобного рода заявления явно и принципиально неверны с точки зрения теоремы об отсутствии бесплатных обедов. Тем не менее типичные статьи по эволюционным алгоритмам включают такие утверждения в аннотацию, введение и заключение. Более скромным было бы заявление:

> *Следовательно, мы видим, что А лучше, чем В, для оптимизации функций, имеющих характеристики задач, которые мы исследовали в данной статье.*

Вместе с тем типичные статьи по эволюционным алгоритмам, которые демонстрируют эталонные результативности, не прилагают никаких усилий для изучения характеристик эталонов. В чем особенность эталонного множества F, который побуждает A работать лучше, чем B? Работает ли A лучше, чем B, потому что функции в F дифференцируемы, или потому, что они мультимодальны, или потому, что у них есть непрерывные вторые производные, или потому, что они ограничены, или по какой-то другой из миллиона возможных причин? На такие вопросы трудно ответить, и поэтому они обычно игнорируются. Поэтому еще лучшим (то есть более скромным) заявлением в статье по эволюционным алгоритмам было бы:

> *Следовательно, мы видим, что A лучше, чем B, для оптимизации функций, которые мы исследовали в этой статье.*

Это утверждение лучше, потому что оно не предпринимает попыток обобщения за пределы представленных в статье эмпирических результатов. Другими словами, оно не пытается заявлять слишком многого. Однако даже этого заявления, вероятно, уже будет слишком из-за многих способов регулировки и реализации алгоритмов.

Еще одна связанная с отсутствием бесплатных обедов опасность подачи эталонных результатов заключается в том, что эталонные функции могут находиться не в том же классе, что и интересующие реальные задачи. Мы исходим из того, что большинство исследователей эволюционных алгоритмов заинтересовано в возможном применении своих исследований в реальных задачах. Если статья по эволюционным алгоритмам демонстрирует хорошую результативность алгоритма A на множестве эталонов F, какое это имеет отношение к реальному миру? На самом деле, с учетом теоремы об отсутствии бесплатных обедов, это может указывать на то, что A плохо работает на реальных задачах. В конце концов, если A работает лучше на множестве F, которое включает в себя только эталонные, а не реальные задачи, то B будет работать лучше на множестве $\overline{F}$, которое включает в себя (среди прочего) все реальные задачи. Постоянная гонка за лучшую результативность на эталонных задачах приводит нас к некому множеству алгоритмов, которые были тонко отрегулированы для хорошей эталонной результативности, но которые могут быть бесполезны в инженерных приложениях. Как пишет Джон Хукер: «Хвост виляет собакой, поскольку задачи сами начинают проектировать алгоритмы» [Hooker, 1995].

С учетом данного обсуждения представляется, что в статьях по эволюционным алгоритмам следует уделять больше внимания приложе-

ниям и меньше внимания эталонам. Иногда бывает трудно добиться того, чтобы статья была принята для публикации, если она не включает результаты из модного в настоящее время множества эталонных функций. Однако такая чрезмерная опора на стандартные эталоны дает нам ложную уверенность. Сегодня редко можно увидеть реальное приложение в журнале по эволюционным алгоритмам, если оно не находится в специальном выпуске журнала в разделе «Приложения». Но с учетом теоремы об отсутствии бесплатных обедов приложения в статьях по эволюционным алгоритмам должны быть правилом, а не исключением. Только тестируя результативность эволюционного алгоритма на реальных задачах, мы можем быть уверены, что эволюционный алгоритм будет полезен.

В.2.2. *Как отчитываться (и как не отчитываться) о результатах симулирования*

В этом разделе показано, как авторы нередко отчитываются о результатах симулирования дезориентирующим образом (надеюсь, непреднамеренно) и как они могут подавать результаты для передачи любого сообщения, которое автор пожелает. В нем также показано, как вместо этого с помощью очень небольшой статистической подготовки можно подавать результаты симулирования в ясном и беспристрастном виде. Мы откладываем более тщательное обсуждение статистики до раздела В.2.4.

Предположим, что мы хотим сравнить эффективность эволюционных алгоритмов A и B на неком эталоне. Предположим, что мы выполняем алгоритмы A и B на неком эталоне. Каждый алгоритм работает в течение T поколений. В каждом поколении мы измеряем стоимость $f_{\min}$ лучшей особи во всей популяции. В результате получим множество данных, которое выглядит следующим образом:

$$\begin{aligned} &\text{Алгоритм } A\text{:} \quad f_{A,\min} = \{f_{A0}, f_{A1}, f_{A2}, \ldots, f_{AT}\} \\ &\text{Алгоритм } B\text{:} \quad f_{B,\min} = \{f_{B0}, f_{B1}, f_{B2}, \ldots, f_{BT}\}, \end{aligned} \tag{В.1}$$

где f_{A0} и f_{B0} – это стоимостные значения лучшей особи после инициализации и f_{Ai} и f_{Bi} – стоимостные значения лучшей особи после i-го поколения. Результативность алгоритма A можно квантифицировать по стоимости лучшей особи, которую он нашел в течение T поколений:

$$\text{Метрический показатель алгоритма } A\text{:} \quad \min_{i \in [0,T]} f_{Ai}. \tag{В.2}$$

Если допустить, что мы используем элитарность, то f_{Ai} – монотонно невозрастающая функция, поэтому

$$\text{Метрический показатель алгоритма } A\text{:} \quad \min_{i \in [0,T]} f_{Ai} = f_{AT}. \tag{В.3}$$

Если мы выполняем алгоритмы A и B на заданном эталоне и хотим увидеть, какой алгоритм работает лучше, то можем попросту сравнить f_{AT} и f_{BT}. Мы вполне можем заключить, что

$$\left.\begin{array}{l} A \text{ лучше } B, \text{ если } f_{AT} < f_{BT} \\ B \text{ лучше } A, \text{ если } f_{BT} < f_{AT} \end{array}\right\} \text{ Неверное обоснование.} \tag{В.4}$$

Как отмечалось выше, это обоснование является неверным. Мы не должны использовать порочную логику, как показано в уравнении (В. 4). Первостепенное значение имеет то, что эволюционные алгоритмы являются стохастическими, то есть зависят от исхода генератора случайных чисел. Следовательно, в общем случае данный эволюционный алгоритм будет давать разные результаты при каждом прогоне. Если мы проведем однократный эксперимент, то вполне можем сделать вывод, что алгоритм A лучше, но если мы проведем его снова, то вполне можем сделать вывод, что алгоритм B лучше. Поэтому при сравнении эволюционных алгоритмов критически важно проводить несколько симуляций и при сравнении алгоритмов использовать результаты всех симуляций. Использование нескольких симуляций, каждая из которых посеяна другим посевом случайного числа, называется симуляцией Монте-Карло, экспериментом Монте-Карло или методом Монте-Карло [Robert и Casella, 2010].

Итак, теперь предположим, что мы поняли концепцию симуляции Монте-Карло. Поэтому мы выполняем алгоритмы A и B на некой задаче и выполняем алгоритмы M раз каждый, где M – это число симуляций Монте-Карло. Каждый раз, когда мы проводим симуляцию, мы получаем другой результат для f_{AT} и f_{BT}. Используем запись f_{AT_k} и f_{BT_k} для обозначения стоимости алгоритмов A и B в конце T-го поколения k-й симуляции Монте-Карло. Мы можем записать наши данные моделирования следующим образом:

$$\begin{array}{ll} \text{Результаты алгоритм } A\text{:} & \{f_{AT1}, f_{AT2}, \ldots, f_{ATM}\} \\ \text{Результаты алгоритм } B\text{:} & \{f_{BT1}, f_{BT2}, \ldots, f_{BTM}\} \end{array} \tag{В.5}$$

Теперь с этими результатами мы можем проделать несколько разных вещей. Во-первых, мы можем сравнить среднюю результативность двух алгоритмов:

$$\begin{aligned} \overline{f}_A &= \frac{1}{M}\sum_{k=1}^{M} f_{ATk} \\ \overline{f}_B &= \frac{1}{M}\sum_{k=1}^{M} f_{BTk} \end{aligned} \tag{В.6}$$

Во-вторых, можем сравнить разницу в результативности двух алгоритмов[1]:

$$\begin{aligned} \sigma_A^2 &= \frac{1}{M-1}\sum_{k=1}^{M}(f_{ATk} - \overline{f}_A)^2 \\ \sigma_B^2 &= \frac{1}{M-1}\sum_{k=1}^{M}(f_{BTk} - \overline{f}_B)^2 \end{aligned} \tag{В.7}$$

В-третьих, можем сравнить результативность двух алгоритмов в лучшем случае:

$$\begin{aligned} \overline{f}_{A,\text{лучший}} &= \min_{k\in[1,M]} f_{ATk} \\ \overline{f}_{B,\text{лучший}} &= \min_{k\in[1,M]} f_{BTk} \end{aligned} \tag{В.8}$$

В-четвертых, можем сравнить результативность двух алгоритмов в худшем случае:

$$\begin{aligned} \overline{f}_{A,\text{худший}} &= \max_{k\in[1,M]} f_{ATk} \\ \overline{f}_{B,\text{худший}} &= \max_{k\in[1,M]} f_{BTk} \end{aligned} \tag{В.9}$$

Каждый из этих метрических показателей характеризует разные типы результативности. Среднестоимостные значения уравнения (В.6) говорят, насколько хорошо алгоритмы работают в среднем. Дисперсии уравнения (В.7) говорят нам о стабильности результативности алгоритмов. Стоимостные значения для лучшего случая уравнения (В.8) говорят, от какого алгоритма мы можем ожидать лучших результатов, если выполним его несколько раз. Стоимостные значения для худшего случая уравнения (В.9) говорят нам, от какого алгоритма мы можем

[1] Интуитивно мы ожидаем, что знаменатель в уравнении (В.7) должен быть M вместо $M-1$. Однако использование $M-1$ в знаменателе дает более качественную оценку дисперсии [Simon, 2006, задача 3.6].

ожидать лучших результатов, если выполним алгоритм только один раз, и при этом случайно посеем генератор случайных чисел таким образом, что получим необычно плохую результативность. В публикации [Eiben и Smit, 2011] приводится несколько дополнительных обсуждений метрических показателей результативности эволюционных алгоритмов.

Предположим, что мы выполняем алгоритмы *A* и *B* на эталонной задаче. Мы выполняем алгоритмы *M* раз каждый и вычисляем следующие метрические показатели:

$$\begin{aligned} \overline{f}_A &= 14, \quad \sigma_A^2 = 4, \quad \overline{f}_{A,\text{лучший}} = 7, \quad \overline{f}_{A,\text{худший}} = 23 \\ \overline{f}_B &= 16, \quad \sigma_B^2 = 3, \quad \overline{f}_{B,\text{лучший}} = 6, \quad \overline{f}_{B,\text{худший}} = 25 \end{aligned} \tag{B.10}$$

В статье, которую мы напишем об этом эксперименте, может появиться любое из следующих ниже утверждений.

1. Алгоритм *A* получает среднюю минимальную стоимость 14, тогда как алгоритм *B* получает среднюю минимальную стоимость 16. Это показывает, что алгоритм *A* работает лучше.
2. Алгоритм *A* имеет дисперсию 4, тогда как алгоритм *B* имеет дисперсию 3. Это показывает, что алгоритм *B* робастнее.
3. Алгоритм *A* получил лучшую стоимость 7, тогда как алгоритм *B* получил лучшую стоимость 6. Это показывает, что алгоритм *B* работает лучше.
4. Алгоритм A получил стоимость, которая ни разу не была хуже 23, тогда как алгоритм *B* получил стоимость, которая ни разу не была хуже 25. Это показывает, что алгоритм *A* работает лучше.

Ни одно из этих утверждений не является абсолютно ложным, но они показывают, как мы интерпретируем статистику в соответствии с предвзятыми идеями. То есть мы не объективны. Мы склонны представлять результаты таким образом, который благоприятствует любому алгоритму, согласующемуся с нашей тенденциозностью.

Рассмотрим первое указанное выше утверждение. Действительно верно, что в среднем алгоритм *A* работает лучше алгоритма *B*. Однако почему мы решили включить это утверждение в нашу статью вместо утверждения 2 или 3? Более того, насколько важна улучшенная результативность, которая отмечена в утверждении 1? Алгоритм *A* работает только на 2 единицы лучше, чем алгоритм *B*, который находится в пределах стандартных отклонений алгоритмов. На рис. B.3 показан график

средних значений и стандартных отклонений алгоритмов. Рисунок слева означает, что алгоритм *A* значительно лучше, чем алгоритм *B*, но рисунок справа показывает, что улучшенная результативность не так впечатляет, как мы могли бы подумать в противном случае. Любая заданная симуляция алгоритмов может легко привести к тому, что алгоритм *B* превзойдет алгоритм *A*, если они покажут результаты в одно стандартное отклонение иначе, чем их средние значения.

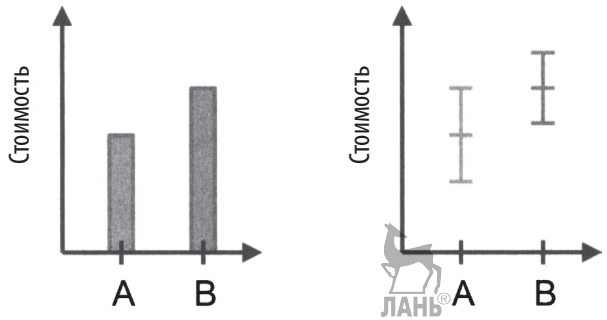


Рис. В.3. Средние значения (левый рисунок) и средние значения и стандартные отклонения (правый рисунок) результативности двух гипотетических алгоритмов. Рисунок слева делает алгоритм А более впечатляющим, чем рисунок справа

Коробчатые диаграммы представляют собой хороший способ построения графиков результатов работы эволюционных алгоритмов. Коробчатые графики по каждому алгоритму включают пять элементов данных: наименьший результат, нижний квартиль, медиана, верхний квартиль и наибольший результат *M* симуляций Монте-Карло [McGill и соавт., 1978]. Нижний квартиль – это значение, превышающее 25 % всех результатов, медиана – это значение, превышающее 50 % всех результатов, а верхний квартиль – это значение, превышающее 75 % всех результатов[2]. Пакет расширения MATLAB® Statistics Toolbox содержит функцию `boxplot` для создания коробчатых графиков. На рис. В.4 показан пример коробчатой диаграммы. Коробчатые диаграммы демонстрируют на одном графике много важной информации. Они показывают медианные результаты, показывают типичные результаты как срединные 50 % и показывают предельные результаты. Это делает коробчатые диаграммы кратким и информативным способом подачи результатов и сравнения алгоритмов.

[2] Обратите внимание, что коробчатые диаграммы содержат пять квартилей: 0%-, 25%-, 50%-, 75%- и 100%-ный квартили. 0%-ный квартиль – это минимальное значение, 50%-ный квартиль – это медианное значение, а 100%-ный квартиль – это максимальное значение.

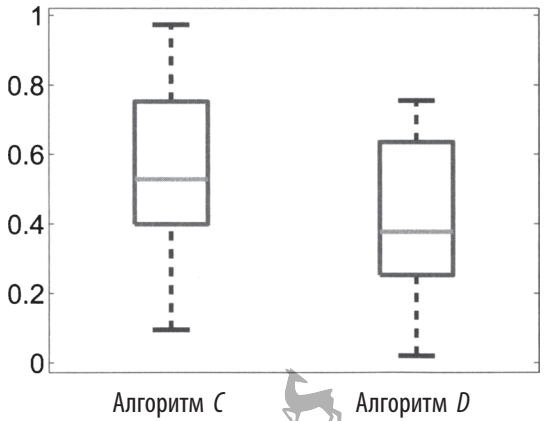


Рис. B.4. Типичная коробчатая диаграмма результативности двух гипотетических алгоритмов. Каждая коробка показывает срединные 50%-ные множества результатов для алгоритма. Линия внутри каждой коробки показывает медианный результат. Линии над и под каждой коробкой (соединенные с каждой коробкой пунктирными линиями) показывают максимальный и минимальный результаты

Вместе с тем нет никаких ошибкоустойчивых способов предотвратить тенденциозность в представлении результатов симулирования. Например, предположим, что мы проводим некое симулирование алгоритмов *A* и *B* и что мы собираем пять значений квартилей для каждого набора симуляций. Предположим, что мы делаем это для шести эталонов. Наши результаты вполне могут выглядеть примерно как в табл. B.1. Мы видим, что алгоритм *A* (современное состояние технологии) работает лучше на эталонах 1, 2 и 3, в то время как алгоритм *B* (наш недавно предложенный алгоритм) работает лучше на эталонах 4, 5 и 6. Что делать в этой ситуации? Мы потратили много месяцев на разработку, доработку, программирование, отладку и тестирование нашего нового алгоритма. Кроме того, нам нужны публикации, чтобы получить степень магистра или должность и финансирование. Многие исследователи в этой ситуации испытывают соблазн написать статью, которая включает эталоны 4, 5 и 6 и опускает результаты, полученные из эталонов 1, 2 и 3.

Такая практика неэтична. Из всех распространенных сегодня практик в области научных исследований и публикаций это далеко не худшее, но большинство из нас согласится, что это нечестно. Однако этика не всегда бывает черно-белой. Эта ситуация возникает в модулированных формах гораздо чаще, чем совершенно очевидная выборочная подача результатов, проиллюстрированная в табл. B.1. Например, если мы получаем предварительные результаты, которые не говорят хорошо о

наших исследованиях, то легко убедить себя, что по какой-то причине эти результаты являются подозрительными[3], а затем получить более обширные результаты, используя только те эталоны, которые хорошо говорят о наших исследованиях.

Таблица В.1. Образец результатов симуляции для двух алгоритмов на шести эталонах. Гипотетические числа – это средние значения набора симуляций Монте-Карло. Алгоритм *A* – это современное состояние технологии, и алгоритм *B* – недавно разработанный алгоритм, предлагаемый исследователем. При наличии этих результатов многие исследователи примут решение опубликовать данные из эталонов 4, 5 и 6 и исключить результаты из эталонов 1, 2 и 3

	F_1	F_2	F_3	F_4	F_5	F_6
Алгоритм *A*	13	21	16	45	24	33
Алгоритм *B*	16	30	22	36	17	28

Стандарты рецензирования поощряют эту тонкую форму нечестности. Если новый предложенный алгоритм или вариация существующего алгоритма не дает более качественных результатов, чем современное состояние технологии, он часто отклоняется и не публикуется. Это происходит даже несмотря на то, что современное состояние технологии, по определению, эволюционировало на протяжении многих десятилетий, в то время как новые алгоритмы, по определению, находятся в зачаточном состоянии и еще не имели возможности созреть. Предложения и статьи 1960-х годов, связанные с генетико-алгоритмическими исследованиями, регулярно отклонялись, поскольку генетические алгоритмы еще не были так хороши, как установившиеся подходы к оптимизации. Типичной была критика, высказанная в 1966 году: «Все, что вы предложили [для генетико-алгоритмических исследований], может быть выражено гораздо яснее и острее с точки зрения древовидного поиска» [Fogel, 1999, стр. xi]. Научно-исследовательское учреждение предположительно открыто для новых идей, но, как и любое другое учреждение, мы склонны сосредоточиваться на себе и препятствовать тем, кто мыслит нестандартно.

Вместо того чтобы поощрять инкрементные результаты и настаивать на постоянном улучшении эталонных результатов, рецензенты должны поощрять новизну и изобретательность, потому что мы не можем

[3] Возможно, мы не уверены в математической формулировке эталона. Возможно, эталон занимает слишком много процессорного времени, и мы хотим сосредоточиться на более быстрых эталонах. Возможно, эталон не так популярен, как другие. Возможно, эталон имеет какие-то характеристики, которые убеждают нас в его нетипичности. Если долго вглядываться, то мы можем найти много причин, для того чтобы исключить какой-либо эталон.

предсказать, какие новые алгоритмы или технологии будут оказывать влияние в будущем. Кроме того, с учетом теоремы об отсутствии бесплатных обедов рецензенты должны не просто поощрять, но и настаивать на том, чтобы авторы обсуждали недостатки и слабые места предлагаемых ими алгоритмов. Авторы не должны испытывать давление в том, чтобы скрывать негативные результаты, но должны быть обязаны публиковать отрицательные результаты в интересах открытости и прозрачности.

Коэффициент успешности

Одним из общепринятых метрических показателей результативности в эволюционных алгоритмах является коэффициент успешности (success rate) [Suganthan и соавт., 2005], [Liang и соавт., 2006], [Mallipeddi и Suganthan, 2010]. То есть при наличии некоторой эталонной функции стоимости $f(x)$ с известным минимумом f^* мы проводим M симуляций нашего эволюционного алгоритма, каждую для конкретного числа оцениваний функции F_{max}. Мы считаем данное моделирование успешным, если оно находит некую особь x – такую, что $f(x) < f^* + \epsilon$, где ϵ – положительный порог успеха. Коэффициент успешности S – это процент успешных симуляций.

Данный метрический показатель стал настолько популярным, что многие редакторы и рецензенты не будут публиковать статьи без него. Однако с этим метрическим показателем связано несколько проблем. Во-первых, выбор ϵ является произвольным и может оказать существенное влияние на очевидные достоинства двух конкурирующих алгоритмов. Если $\epsilon = \epsilon_1$, то алгоритм A вполне может превзойти алгоритм B, но если $\epsilon = \epsilon_2 \neq \epsilon_1$, то алгоритм B вполне может превзойти алгоритм A. Во-вторых, число оцениваний функции F_{max} является произвольным и может также повлиять на очевидные достоинства двух конкурирующих алгоритмов. Если $F_{max} = F_1$, то алгоритм A вполне может превзойти алгоритм B, но если $F_{max} = F_2 \neq F_1$, то алгоритм B вполне может превзойти алгоритм A. В-третьих, для того чтобы получить уверенную оценку коэффициента успешности S, число симуляций M должно быть нереально большим. Мы вполне можем использовать $M = 20$ и найдем $S = 30$ %, а затем вернуться на следующий день и провести еще 20 симуляций и найти $S = 40$ %. Для того чтобы получить достаточно небольшое стандартное отклонение в S, нам, возможно, потребуется использовать очень большое значение M, но такие большие значения M вполне могут потребовать для сбора данных много дней или недель [Clerc, 2012b]. Когда в публикациях сообщается о коэффициенте успешности, он никогда не

сопровождается стандартным отклонением коэффициента успешности, и это отрицает его полезность в качестве метрического показателя.

Размер популяции

Для конкретной задачи ЭА #1 может показывать лучший результат с размером популяции N_1, тогда как ЭА #2 может показывать лучший результат с размером популяции N_2. Следовательно, когда мы сравниваем два эволюционных алгоритма, нам, возможно, сначала нужно отрегулировать каждый эволюционный алгоритм отдельно, с тем чтобы найти их оптимальные размеры популяции. С другой стороны, цель нашего сравнения вполне может заключаться в том, чтобы увидеть, какой эволюционный алгоритм лучше всего работает с заданным размером популяции. В этом случае мы должны использовать одинаковый размер популяции для обоих эволюционных алгоритмов. Например, результативность эволюционного алгоритма при небольших размерах популяции вполне может быть важной темой исследования. Таким образом, вопрос о том, какие размеры популяции использовать при сравнении эволюционных алгоритмов, выражен нечетко, но зависит от цели сравнения.

В.2.3. *Случайные числа*

Для получения допустимых результатов эволюционно-алгоритмического симулирования необходимо четко разбираться в генераторах случайных чисел. Генератор случайных чисел может существенно повлиять на работу эволюционного алгоритма. Качество решения эволюционного алгоритма не обязательно коррелирует с качеством генератора случайных чисел в эволюционном алгоритме [Clerc, 2012b]. Для отдельных задач и отдельных эволюционных алгоритмов более качественная генерация случайных чисел приводит к более высокой результативности эволюционного алгоритма, в то время как для других задач и других эволюционных алгоритмов менее качественная генерация случайных чисел приводит к более высокой результативности эволюционного алгоритма.

Многие случайные числа не являются случайными. На самом деле определение и существование случайности являются предметом отдельных споров. Генераторы случайных чисел в компьютерах не генерируют случайных чисел по-настоящему; они генерируют псевдослучайные числа. Псевдослучайные числа выглядят как случайные, но на самом деле они не случайны, потому что генерируются с помощью детерминированных алгоритмов.

Остальная часть этого раздела посвящена среде программирования на языке MATLAB для иллюстрации, но ее можно легко обобщить на любую другую среду программирования. Рассмотрим функцию языка MATLAB `rand`, которая возвращает случайное число, равномерно распределенное на открытом интервале (0,1). Предположим, что мы начинаем сеанс в среде MATLAB с использованием языка MATLAB версии 7.13.0.564 (R2011b) на 32-разрядном компьютере Windows 8. Если мы применим функцию `rand` для генерации пяти случайных чисел, то получим числа

$$0.8147,\ 0.9058,\ 0.1270,\ 0.9134,\ 0.6324. \qquad \text{(B.11)}$$

Этот ряд выглядит довольно случайным. Но если мы выключим компьютер, вернемся на следующий день, снова запустим среду MATLAB и снова с помощью функции `rand` сгенерируем пять случайных чисел, то получим точно такие же числа, как показано выше! Внезапно цифры не выглядят случайными. Причина в том, что для генерации псевдослучайных чисел в языке MATLAB используется детерминированный алгоритм и что алгоритм имеет то же исходное состояние всякий раз, когда среда MATLAB начинает свою работу.

Этот факт имеет важное значение для тестирования эволюционных алгоритмов. Предположим, что мы включаем компьютер, выполняем эволюционный алгоритм и получаем результативность f_1. Теперь предположим, что мы выходим из системы, возвращаемся на следующий день, входим в систему, запускаем MATLAB и снова выполняем эволюционный алгоритм. Мы получим точно такой же результат, потому что случайные числа, которые мы генерируем в нашем эволюционном алгоритме для отбора, мутации и других целей, будут точно такими же, как и в предыдущий день. Хотя эволюционный алгоритм является стохастическим и, следовательно, теоретически должен давать разные результаты при каждом прогоне, на практике он будет давать одинаковый результат изо дня в день, потому что он опирается на генератор псевдослучайных чисел, который возвращает числа, не являющиеся по-настоящему случайными, и который инициализируется в одинаковое состояние, всякий раз когда среда MATLAB начинает свою работу.

Теперь предположим, что мы включаем компьютер, выполняем эволюционный алгоритм и получаем результат f_1. Мы оставляем среду MATLAB работать, возвращаемся на следующий день и снова выполняем эволюционный алгоритм. В этот раз мы в общем случае получим другой результат. Причина в том, что мы не реинициализировали генератор случайных чисел, перезапустив среду MATLAB. Поэтому при по-

вторном выполнении эволюционного алгоритма среда MATLAB будет генерировать разные ряды случайных чисел, чем при первом выполнении эволюционного алгоритма.

Если мы хотим генерировать один и тот же ряд случайных чисел, то можем применить функцию MATLAB `rng(seed)` для реинициализации генератора случайных чисел в заданное целое число `seed`[4]. В результате генератор случайных чисел инициализируется так, что он начинается в заданном состоянии и таким образом генерирует ту же самую последовательность случайных чисел из одного прогона в другой. Например, команды MATLAB

```
rng(489)
rand, rand, rand, rand, rand                    (B.12)
```

генерируют случайные числа 0.9780, 0.8578, 0.0599, 0.0894, 0.4774 независимо от того, когда выполняются команды, и не важно, как долго среда MATLAB работала до этого. С помощью данного подхода мы можем создавать идентичные результаты эволюционного алгоритма из одной симуляции в другую, даже если результаты эволюционного алгоритма являются по своей сути стохастическими. Иногда дефект или результат работы эволюционного алгоритма проявляется только при особых обстоятельствах. Для того чтобы повторить дефект или результат, нам нужно инициализировать генератор случайных чисел одинаковым посевом. Поэтому при каждом прогоне эволюционного алгоритма всегда полезно отслеживать исходный посев генератора случайных чисел. Отслеживание посева позволяет нам точно воспроизводить результаты, что значительно ускоряет отладку.

Если мы хотим генерировать разные ряды случайных чисел всякий раз, когда запускаем наш эволюционный алгоритм, то мы можем снова применить функцию MATLAB `rng`, но вместо этого предоставить ей случайный посев. Например, целое число, представляющее текущую дату и время, является довольно случайным, и поэтому использование даты и времени в качестве посева приводит к другой последовательности случайных чисел всякий раз, когда мы запускаем среду MATLAB. Команда MATLAB `clock` возвращает шестиэлементный вектор, содержащий текущий год, месяц, день, часы, минуты и секунды. Поэтому команды MATLAB

```
Seed = sum(100*clock);
rng(Seed);
rand, rand, rand, rand, rand                    (B.13)
```

[4] Функция `rng` была введена примерно в 2011 году. Более ранние версии MATLAB предоставляют для инициализации генератора случайных чисел другие функции.

генерируют другую последовательность случайных чисел всякий раз, когда мы их исполняем. С помощью этого подхода мы можем создавать произвольно варьирующиеся результаты эволюционного алгоритма из одной симуляции в другую, даже если выполняем нашу симуляцию первым делом утром после запуска среды MATLAB. Этот подход также позволяет нам отслеживать начальный посев случайных чисел, чтобы его можно было записывать и использовать позже в случае необходимости отладки исходного кода.

Теперь предположим, что мы хотим провести простое сравнение двух эволюционных алгоритмов. Первым случайным процессом в наших эволюционных алгоритмах является генерация исходной популяции. Поэтому мы выполняем первый эволюционный алгоритм и получаем некие результаты. Затем мы выполняем второй эволюционный алгоритм и получаем другие результаты. Однако если мы не будем реинициализировать генератор случайных чисел между симуляциями, то два эволюционных алгоритма начнут с совершенно разных начальных популяций. Это может дать несправедливое преимущество одному эволюционному алгоритму. Если мы хотим более справедливо сравнивать разные эволюционные алгоритмы, они должны начинать с одной и той же исходной популяции. Это можно сделать с помощью следующей ниже последовательности команд:

```
Seed = sum(clock);
rng(Seed);
EA!;
rng(Seed);
EA2;
```
(B.14)

Данная последовательность гарантирует, что генератор случайных чисел инициализируется одинаково для обоих эволюционных алгоритмов, поэтому оба эволюционных алгоритма будут начинаться с одной и той же случайной популяции, при условии что случайная популяция создается одинаково в ЭА #1 и ЭА #2.

B.2.4. *t*-тесты

Статистическая проверка гипотез на основе статистического показателя *t*, или *t*-тест, была изобретена в 1908 году Уильямом Сили Госсетом, ирландским химиком, который использовал его для контроля качества пива для пивоварни, где он работал [Box, 1987]. Он опубликовал метод под псевдонимом «Студент», поскольку его работодатель не хотел, что-

бы его конкуренты знали, что они используют статистические методы. *t*-тест имеет много применений, но в этом разделе мы ограничимся нашим обсуждением очень специфического приложения, связанного с интерпретацией эволюционно-алгоритмических экспериментов, и продемонстрируем его применение без каких-либо выводов. Стандартные книги по математической статистике включают дополнительные сведения и выводы относительно *t*-теста [Salkind, 2007].

Основной вопрос, на который отвечает проверка гипотез на основе статистического показателя *t*, или *t*-тест, заключается в том, как различить, что два множества экспериментальных результатов значимо отличаются? Когда мы здесь говорим «значимо отличаются», мы не имеем в виду, насколько велика разница; вместо этого мы обращаемся к вопросу о том, является ли разница фундаментальной либо вместо этого происходит из-за случайных флуктуаций. Например, предположим, что мы измеряем успеваемость, или, точнее, результативность, двух студентов в двух отдельных экзаменах в течение курса:

Студент *A*: 69 %, 84 %
Студент *B*: 66 %, 83 % (B.15)

В обоих экзаменах студент *A* набрал больше баллов, чем студент *B*. Но мы, вероятно, не сделаем вывод, что между этими двумя студентами есть разница; мы припишем превосходящую успеваемость студента *A* случайности и предположим, что эти два студента имеют, по существу, идентичную успеваемость и способности. Но теперь предположим, что наше суждение опирается на 10 экзаменов:

Студент *A*: 69, 84, 75, 93, 92, 88, 68, 74, 89, 81 %
Студент *B*: 66, 83, 72, 88, 95, 83, 71, 71, 84, 80 % (B.16)

Студент *A* имел более высокий балл восемь раз из 10. Теперь мы начнем подозревать, что студент *A* действительно лучше студента *B* и что их различия в успеваемости вызваны не просто случайными вариациями. Различия невелики, но, как представляется, статистически значимы. Но как оценить вероятность того, что эта гипотеза является истинной или ложной? В частности, какова вероятность того, что мы получим результаты уравнения (B.16), если студенты *A* и *B* были равными по успеваемости? То есть какова вероятность того, что различия в результатах уравнения (B.16) вызваны просто случайными вариациями? Это именно тот вопрос, на который отвечает *t*-тест. Гипотеза о том, что различия в результатах студентов *A* и *B* вызваны просто случайными вариациями, называется нулевой гипотезой. Обратите внимание, что если студенты

A и *B* равны по успеваемости и мы даем каждому студенту 10 экзаменов, то существует равная вероятность, что любой из студентов, *A* или *B*, будет показывать результаты лучше большую часть времени.

Теперь вернемся к задаче анализа результатов эволюционно-алгоритмического симулирования. Предположим, что мы выполняем алгоритмы *A* и *B* на некой оптимизационной задаче. Мы знаем важность симуляций Монте-Карло, поэтому выполняем каждый алгоритм *M* раз и вычисляем среднюю результативность и дисперсию каждого алгоритма, как показано в уравнениях (B.6) и (B.7):

$$\begin{aligned} &\text{Алгоритм } A\text{: Среднее значение} = \overline{f}_A, \quad \text{Дисперсия} = \sigma_A^2 \\ &\text{Алгоритм } B\text{: Среднее значение} = \overline{f}_B, \quad \text{Дисперсия} = \sigma_B^2 \end{aligned} \tag{B.17}$$

Какова вероятность того, что мы получим эти результаты, если результативность (то есть успеваемость) алгоритмов *A* и *B* принципиально идентична? На этот вопрос отвечает *t*-тест, или статистическая проверка на основе статистического показателя *t*. Статистический показатель *t*, так называемая *t*-статистика, определяется как

$$t = \frac{\left|\overline{f}_A - \overline{f}_B\right|}{\sqrt{(\sigma_A^2 + \sigma_B^2)/M}}. \tag{B.18}$$

Мы видим, что статистический показатель *t* измеряет разницу между исходами алгоритмов *A* и *B*. Если между $\overline{f}_A$ и $\overline{f}_B$ есть большая разница, то статистический показатель *t* будет большим. Однако если σ_A^2 или σ_B^2 большие, то это указывает на то, что существует большая вариация в результативности алгоритма *A* или *B*, которая разбавляет эффект большой разницы между $\overline{f}_A$ и $\overline{f}_B$, что делает *t* малым. Крупное значение *M* имеет тенденцию делать *t* большим, так как большое число экспериментов является более надежным показателем результативности, чем небольшое число экспериментов.

После вычисления проверочного статистического показателя *t* вычисляется величина, именуемая степенями свободы:

$$d = \frac{(M-1)(\sigma_A^2 + \sigma_B^2)^2}{\sigma_A^4 + \sigma_B^4}. \tag{B.18}$$

Степени свободы косвенно говорят нам, насколько большим должен быть статистический показатель *t*, чтобы дать нам определенный уровень уверенности в том, что существует статистически значимая разница в результативности алгоритмов *A* и *B*.

После того как мы вычисляем t и d, мы смотрим в таблицу t-теста, чтобы найти вероятность того, что разница в результативности алгоритмов A и B вызвана случайной вариацией, а не фундаментальной разницей между двумя алгоритмами. Эти вероятности называются p-значениями[5]. Таблица В.2 показывает несколько значений t-теста. Более полные таблицы можно найти в книгах по статистике либо в интернете. Для аппроксимации значений между координатами в таблице можно использовать любой приемлемый интерполяционный метод. Мы можем также использовать функции t-тестирования в статистическом программном обеспечении, включая MATLAB, Microsoft Excel® и многие другие программные пакеты.

Таблица В.2. Таблица двухстороннего t-теста, то есть, двусторонней проверки гипотез на основе статистического показателя t. Каждая строка соответствует степени свободы d, и каждый столбец соответствует вероятности в p-значении. Числа в таблице показывают значения статистического показателя t для заданных степеней свободы d и заданную вероятность в p-значении, что различия между двумя множествами экспериментов полностью обусловлены случайной вариацией

d	50%	40%	30%	20%	10%	5%	2%	1%	0.5%	0.1%
1	1.000	1.376	1.963	3.078	6.314	12.71	31.82	63.66	127.3	636.6
2	0.816	1.061	1.386	1.886	2.920	4.303	6.965	9.925	14.09	31.60
3	0.765	0.978	1.250	1.638	2.353	3.182	4.541	5.841	7.453	12.92
4	0.741	0.941	1.190	1.533	2.132	2.776	3.747	4.604	5.598	8.610
5	0.727	0.920	1.156	1.476	2.015	2.571	3.365	4.032	4.773	6.869
6	0.718	0.906	1.134	1.440	1.943	2.447	3.143	3.707	4.317	5.959
7	0.711	0.896	1.119	1.415	1.895	2.365	2.998	3.499	4.029	5.408
8	0.706	0.889	1.108	1.397	1.860	2.306	2.896	3.355	3.833	5.041
9	0.703	0.883	1.100	1.383	1.833	2.262	2.821	3.250	3.690	4.781
10	0.700	0.879	1.093	1.372	1.812	2.228	2.764	3.169	3.581	4.587
11	0.697	0.876	1.088	1.363	1.796	2.201	2.718	3.106	3.497	4.437
12	0.695	0.873	1.083	1.356	1.782	2.179	2.681	3.055	3.428	4.318
13	0.694	0.870	1.079	1.350	1.771	2.160	2.650	3.012	3.372	4.221
14	0.692	0.868	1.076	1.345	1.761	2.145	2.624	2.977	3.326	4.140
15	0.691	0.866	1.074	1.341	1.753	2.131	2.602	2.947	3.286	4.073
16	0.690	0.865	1.071	1.337	1.746	2.120	2.583	2.921	3.252	4.015

[5] *P*-значение – это наименьшая величина уровня значимости, при которой нулевая гипотеза отвергается для данного значения проверочного статистического показателя t. *–Прим. перев.*

d	50%	40%	30%	20%	10%	5%	2%	1%	0.5%	0.1%
17	0.689	0.863	1.069	1.333	1.740	2.110	2.567	2.898	3.222	3.965
18	0.688	0.862	1.067	1.330	1.734	2.101	2.552	2.878	3.197	3.922
19	0.688	0.861	1.066	1.328	1.729	2.093	2.539	2.861	3.174	3.883
20	0.687	0.860	1.064	1.325	1.725	2.086	2.528	2.845	3.153	3.850
21	0.686	0.859	1.063	1.323	1.721	2.080	2.518	2.831	3.135	3.819
22	0.686	0.858	1.061	1.321	1.717	2.074	2.508	2.819	3.119	3.792
23	0.685	0.858	1.060	1.319	1.714	2.069	2.500	2.807	3.104	3.767
24	0.685	0.857	1.059	1.318	1.711	2.064	2.492	2.797	3.091	3.745
25	0.684	0.856	1.058	1.316	1.708	2.060	2.485	2.787	3.078	3.725
26	0.684	0.856	1.058	1.315	1.706	2.056	2.479	2.779	3.067	3.707
27	0.684	0.855	1.057	1.314	1.703	2.052	2.473	2.771	3.057	3.690
28	0.683	0.855	1.056	1.313	1.701	2.048	2.467	2.763	3.047	3.674
29	0.683	0.854	1.055	1.311	1.699	2.045	2.462	2.756	3.038	3.659
30	0.683	0.854	1.055	1.310	1.697	2.042	2.457	2.750	3.030	3.646
40	0.681	0.851	1.050	1.303	1.684	2.021	2.423	2.704	2.971	3.551
50	0.679	0.849	1.047	1.299	1.676	2.009	2.403	2.678	2.937	3.496
60	0.679	0.848	1.045	1.296	1.671	2.000	2.390	2.660	2.915	3.460
80	0.678	0.846	1.043	1.292	1.664	1.990	2.374	2.639	2.887	3.416
100	0.677	0.845	1.042	1.290	1.660	1.984	2.364	2.626	2.871	3.390

■ Пример 2.5

Предположим, мы выполняем алгоритмы *A* и *B* на некоторой оптимизационной задаче. Мы выполняем каждый алгоритм шесть раз ($M = 6$) и получаем следующие ниже результаты, которые записываются в формате уравнения (B.5):

$$\text{Алгоритм } A\text{: } \{f_{ATk}\} = \{30.02,\ 29.99,\ 30.11,\ 29.97,\ 30.01,\ 29.99\}$$
$$\text{Алгоритм } B\text{: } \{f_{BTk}\} = \{29.89,\ 29.93,\ 29.72,\ 29.98,\ 30.02,\ 29.98\} \qquad \text{(B.20)}$$

Для того чтобы оценить вероятность получения этих различий из-за простой случайной вариации, а не из-за какой-либо фундаментальной разницы между результативностью двух алгоритмов, мы сначала вычисляем среднее значение и дисперсию результатов, как показано в уравнениях (B.6), (B.7) и (B.17):

$$\begin{aligned} \bar{f}_A &= 30.015, \quad \sigma_A^2 = 0.0497 \\ \bar{f}_B &= 29.920, \quad \sigma_B^2 = 0.1079 \end{aligned} \tag{B.21}$$

Затем мы применяем уравнения (B.18) и (B.19) для вычисления проверочного статистического показателя t и степеней свободы:

$$t = 1.959, \quad d = 7.0306. \tag{B.22}$$

Наконец, мы смотрим в табл. B.2 и видим, что для $d = 7.0306$ проверочный статистический показатель t равен 1.959 при p-значении около 9 %. Это означает, что если результативность алгоритмов A и B принципиально эквивалентна, существует 9%-ная вероятность того, что мы получим результаты, показанные в уравнении (B.20). Эквивалентным образом мы можем сказать, что если результативность алгоритмов A и B эквивалентна, то существует 91%-ная вероятность того, что мы увидим меньшие вариации, чем те, которые показаны в уравнении (B.5). Являются ли эти вероятности достаточно для нас значимыми, чтобы заключить, что алгоритмы принципиально отличаются, – это качественное суждение.

■

Существует несколько предположений, лежащих в основе обсуждения вопроса проверки гипотез на основе статистического показателя t этого раздела.

1. Во-первых, мы исходим из допущения, что алгоритм A может быть лучше или хуже алгоритма B, поэтому в этом разделе используем двухсторонний t-тест. Поэтому в заголовке табл. B.2 указано, что она предназначена для двухсторонних t-тестов. Заголовки столбцов (то есть p-значения) табл. B.2 изменились бы, если бы мы захотели провести односторонние t-тесты, но односторонние тесты, как правило, не имеют отношения к анализу эволюционно-алгоритмических экспериментов.
2. t-тест исходит из допущения, что результаты каждого эксперимента подчиняются гауссову распределению. То есть если мы выполним много симуляций для алгоритма A и выведем результаты на гистограмме, то увидим гауссову кривую; то же допущение относится и к алгоритму B. Поскольку предельные значения в гауссовом распределении $\pm\infty$, не существует никаких компьютерных симуляций или физических экспериментов, которые по-настоящему гауссовы. Но многие процессы достаточно хорошо аппрок-

симируются гауссовыми распределениями. Центральная предельная теорема гарантирует нам, что многочисленные эксперименты и симуляции имеют почти гауссовы результаты [Grinstead и Snell, 1997], поэтому гауссовость является разумным допущением. Однако верификация того, что наш процесс действительно гауссов, является отдельной задачей. В целом можно с уверенностью допустить гауссовость в отсутствие доказательств обратного.

3. *t*-тест исходит из допущения, что у нас есть только два множества данных. Если у нас более двух множеств данных – например, результаты алгоритмов *A*, *B* и *C*, – тогда для проверки статистически значимых различий мы не сможем использовать попарные *t*-тесты на парах *A*/*B*, *A*/C и *B*/*C*. Мы обсудим этот вопрос подробнее ниже в разделе B.2.5.
4. В данном разделе принимается допущение, что размеры двух выборок одинаковы; то есть мы проводим *M* экспериментов для обоих алгоритмов *A* и *B*. Если для двух алгоритмов у нас разное число экспериментов, то нам нужно слегка модифицировать уравнения этого раздела.

Многие неверно истолковывают результаты *t*-тестов. Как мы отмечали ранее в этом разделе, *t*-тест дает вероятность того, что наши результаты будут получены, если результативность алгоритмов *A* и *B* будет принципиально эквивалентной. Мы можем записать это как

$$p = \Pr(R = r \mid A = B). \tag{B.23}$$

На словах *p*-значение равно вероятности того, что результаты *R* равны полученным значениям *r*, при условии что результативность алгоритмов *A* и *B* эквивалентна[6]. Здесь мы исправляем некоторые распространенные недоразумения относительно *p*-значения.

1. $p \neq \Pr(A = B)$. То есть *p*-значение равно вероятности того, что результативность алгоритмов эквивалентна.
2. $1 - p \neq \Pr(A \neq B)$. Это утверждение очень похоже на приведенное выше утверждение. То есть мы не можем использовать *p*-значение для получения вероятности того, что результативность алгоритмов отличается.
3. $p \neq \Pr(A = B \mid R = r)$. То есть *p*-значение не равно вероятности того, что результативность алгоритмов эквивалентна, при наличии по-

[6] Обратите внимание, что запись $A = B$ *не* означает, что два алгоритма эквивалентны, а означает, что их результативность эквивалентна.

лученных нами результатов. Из теоремы Байеса [Grinstead и Snell, 1997] мы знаем, что

$$\Pr(A = B \mid R = r) = \frac{\Pr(R = r \mid A = B)\Pr(A = B)}{\Pr(R = r)}$$

$$= \frac{p \Pr(A = B)}{\Pr(R = r)}. \quad \text{(B.24)}$$

Поэтому $\Pr(A = B \mid R = r)$, то есть вероятность того, что результативность алгоритмов A и B эквивалентна при наличии результатов, которые мы наблюдали, прямо пропорциональна p-значению. Но если мы хотим вычислить численное значение для $\Pr(A = B \mid R = r)$, то должны знать не только p, но и $\Pr(A = B)$ (то есть априорную вероятность того, что результативность двух алгоритмов эквивалентна) и $\Pr(R = r)$ (то есть априорную вероятность того, что нами были получены результаты, которые мы получили).

4. $p \neq \Pr(R = r)$. То есть p-значение не равно априорной вероятности получения результатов, которые мы получили. Как показывает уравнение (B.23), p равно апостериорной вероятности получения результатов, которые мы наблюдали, *при условии* что результативность алгоритмов эквивалентна.
5. Предположим, что p – это малое число, поэтому из уравнения (B.24) мы делаем вывод, что $A \neq B$. Тогда $(1 - p)$ не равно вероятности того, что второй эксперимент даст тот же вывод.
6. P-значение не говорит нам количественно, насколько различны эти два алгоритма. Однако p-значение имеет положительную корреляцию с величиной разницы, поэтому большее p-значение указывает на бóльшую разницу.

Все это может выглядеть как придирки к мелочам, зациклившись на деталях смысла p-значения, но вместе с тем могут существовать большие различия между истинным смыслом p-значения и распространенными, но ложными интерпретациями, отмеченными выше [Johnson, 1999], [Schervish, 1996], [Sterne и Smith, 2001].

B.2.5. *F-тесты*

Для целого ряда задач мы можем использовать проверку гипотез на основе статистического показателя F, или F-тест, как и t-тест. В этом разделе рассматривается одно простое применение F-теста, которое

особенно полезно для анализа результатов работы эволюционных алгоритмов. Как и в случае с *t*-тестом, мы ограничиваем наше обсуждение *F*-теста очень конкретным приложением, связанным с интерпретацией эволюционно-алгоритмических экспериментов, и демонстрируем его применение без каких-либо выводов.

Предположим, что мы тестируем алгоритмы 1, 2 и 3 на оптимизационной задаче. Мы хотим использовать результаты, чтобы судить о том, есть ли статистически значимая разница между тремя алгоритмами. Как уже упоминалось в предыдущем разделе, мы не можем выполнить попарные *t*-тесты на алгоритмах 1 и 2, 1 и 3 и затем 2 и 3. В качестве простого интуитивного объяснения того, почему попарное *t*-тестирование не работает, предположим, что мы бросаем справедливый шестигранный кубик. Мы знаем, что у нас есть 1/6 ≈ 17%-ный шанс получить 1. Теперь предположим, что мы бросаем кубик три раза. У нас есть вероятность $1 - (5/6)^3 \approx 42$ % получить 1, по крайней мере, в одном из этих бросков. Вероятность того, что событие произойдет после одного испытания, не совпадает с вероятностью того, что оно произойдет после нескольких испытаний. *t*-тест дает нам вероятность наблюдения определенных различий между двумя алгоритмами, при условии что результативность двух алгоритмов идентична. Однако если мы проведем этот тест более одного раза, то вероятность получения этих различий возрастает.

Теперь, когда мы знаем, что не можем использовать *t*-тест, давайте обсудим, как использовать *F*-тест. Предположим, что мы выполняем *G* отдельных алгоритмов для некой оптимизационной задачи. Эти алгоритмы могут быть одним и тем же алгоритмом, но с разными параметрами (например, *G* разных мутаций). Мы выполняем каждый алгоритм *M* раз и вычисляем среднюю результативность и дисперсию каждого алгоритма, как показано в уравнениях (В.6) и (В.7):

$$\text{Среднее значение} = \bar{f}_g, \quad \text{Дисперсия} = \sigma_g^2 \quad \text{для } g \in [1, G]. \qquad \text{(В.25)}$$

Какова вероятность того, что мы получим эти результаты, если результативность алгоритмов *G* принципиально идентична? То есть какова вероятность того, что эти результаты не связаны с какой-либо фундаментальной разницей между алгоритмами *G*, а происходят просто из-за случайных экспериментальных вариаций? На этот вопрос отвечает *F*-тест, или проверка гипотез на основе статистического показателя *F*. Статистический показатель *F*, так называемая *F*-стастистика, рассчитывается как

$$\bar{f} = \frac{1}{G}\sum_{g=1}^{G}\bar{f}_g$$

$$S_w = \frac{1}{G}\sum_{g=1}^{G}\sigma_g^2$$

$$S_b = \frac{M}{G-1}\sum_{g=1}^{G}(\bar{f}_g - \bar{f})^2$$

$$F = S_b / S_w \qquad \text{(B.26)}$$

где $\bar{f}$ – это средний метрический показатель результативности всех алгоритмов, S_w – внутригрупповая дисперсия, которая является мерой средней дисперсии алгоритмов, S_b – межгрупповая дисперсия, которая является мерой дисперсии результативности всех алгоритмов, F – статистический показатель, то есть F-статистика. Мы видим, что большая разница в результативности алгоритмов соответствует большому значению статистического показателя F.

После вычисления статистического показателя F вычисляются величины, которые называются числителем степени свободы D_n и знаменателем степени свободы D_d:

$$D_n = G - 1$$
$$D_d = G(M - 1) \qquad \text{(B.27)}$$

Степени свободы косвенно говорят нам, насколько велик статистический показатель F, для того чтобы дать нам определенный уровень уверенности в том, что существует статистически значимая разница в результативности алгоритмов.

После того как мы вычислим F, D_n и D_d, мы смотрим в таблицу F-тестов, чтобы найти вероятность того, что разница в результативности G алгоритмов вызвана случайной вариацией, а не фундаментальной разницей между двумя или более алгоритмами. Поскольку у нас два параметра степени свободы (D_n и D_d), нам нужна отдельная таблица для каждого уровня вероятности. В табл. В.3 и В.4 показано несколько пороговых значений F-теста для вероятностных значений 5 % и 1 %. Более полные таблицы можно найти в книгах по статистике или в интернете. Для аппроксимации значений между координатами в таблицах можно использовать любой приемлемый интерполяционный метод. Мы можем также использовать функции t-тестирования в статистически программном обеспечении, включая MATLAB, Microsoft Excel® и многие другие программные пакеты.

Таблица B.3. Таблица *F*-теста для 5%-ной вероятности. Каждая строка соответствует знаменателю степени свободы D_d, а каждый столбец соответствует числителю степени свободы D_n. Числа в таблице показывают значения *F*, которые позволяют сделать вывод, что существует 5%-ная или меньшая вероятность того, что различия между несколькими множествами экспериментов полностью обусловлены случайной вариацией

	$D_n = 1$	2	3	4	5	6	7	8
$D_d = 1$	161.47	199.49	215.74	224.50	230.07	234.00	236.77	238.95
2	18.51	18.99	19.16	19.24	19.29	19.32	19.35	19.37
3	10.12	9.55	9.27	9.11	9.01	8.94	8.88	8.84
4	7.70	6.94	6.59	6.38	6.25	6.16	6.09	6.04
5	6.60	5.78	5.40	5.19	5.05	4.95	4.87	4.81
6	5.98	5.14	4.75	4.53	4.38	4.28	4.20	4.14
7	5.59	4.73	4.34	4.12	3.97	3.86	3.78	3.72
8	5.31	4.45	4.06	3.83	3.68	3.58	3.50	3.43
9	5.11	4.25	3.86	3.63	3.48	3.37	3.29	3.22
10	4.96	4.10	3.70	3.47	3.32	3.21	3.13	3.07

Таблица B.4. Таблица *F*-теста для 1%-ной вероятности. Каждая строка соответствует знаменателю степени свободы D_d, а каждый столбец соответствует числителю степени свободы D_n. Числа в таблице показывают значения *F*, которые позволяют сделать вывод, что существует 1%-ная или меньшая вероятность того, что различия между несколькими множествами экспериментов полностью обусловлены случайной вариацией

	$D_n = 1$	2	3	4	5	6	7
$D_d = 1$	4063.25	4992.22	5404.03	5636.51	5760.41	5889.88	5889.88
2	98.50	98.99	99.15	99.26	99.30	99.34	99.34
3	34.11	30.81	29.45	28.70	28.23	27.91	27.67
4	21.19	17.99	16.69	15.97	15.52	15.20	14.97
5	16.25	13.27	12.05	11.39	10.96	10.67	10.45
6	13.74	10.92	9.77	9.14	8.74	8.46	8.25
7	12.24	9.54	8.45	7.84	7.46	7.19	6.99
8	11.25	8.64	7.59	7.00	6.63	6.37	6.17
9	10.56	8.02	6.99	6.42	6.05	5.80	5.61
10	10.04	7.55	6.55	5.99	5.63	5.38	5.20

Обратите внимание, что статистический показатель F в табл. В.3 и В.4 уменьшается вместе с D_n и D_d. То есть, по мере того как число групп G и число экспериментов Монте-Карло M увеличиваются, нам требуются меньшие различия между метрическими показателями результативности алгоритмов, чтобы заключить, что эти различия не связаны с простыми случайными вариациями. Например, рассмотрим табл. В.3 с $D_n = D_d = 3$. Если $F = 9.27$, то существует 5%-ная вероятность того, что наблюдаемые различия обусловлены случайными причинами. Если $F > 9.27$, то существует менее 5 % вероятности того, что наблюдаемые различия вызваны случайной вариацией, поэтому существует более 95 % вероятности того, что наблюдаемые различия не вызваны случайной вариацией. Сравните это с $D_n = D_d = 5$. В этом случае если $F = 5.05$, то существует 5%-ная вероятность того, что наблюдаемые различия обусловлены случайными причинами. Если $F > 5.05$, то существует менее 5 % вероятности того, что наблюдаемые различия вызваны случайной вариацией, поэтому существует более 95 % вероятности того, что наблюдаемые различия не вызваны случайной вариацией.

■ Пример 2.6

Предположим, что мы выполняем алгоритмы A, B и C для некой оптимизационной задачи. Мы выполняем каждый алгоритм четыре раза ($M = 4$) и получаем следующие ниже результаты, которые записываются в формате уравнения (В.5):

$$\begin{aligned} &\text{Алгоритм } A\text{: } \{f_{ATk}\} = \{4, 5, 3, 2\} \\ &\text{Алгоритм } B\text{: } \{f_{BTk}\} = \{6, 4, 4, 5\} \\ &\text{Алгоритм } C\text{: } \{f_{CTk}\} = \{5, 7, 6, 6\} \end{aligned} \tag{В.28}$$

Для того чтобы оценить вероятность получения этих различий из-за простой случайной вариации, а не из-за какой-либо фундаментальной разницы между результативностью трех алгоритмов, мы сначала вычисляем среднее значение и дисперсию результатов, как показано в уравнениях (В.6), (В.7) и (В.25):

$$\begin{aligned} \bar{f}_A &= 3.50, \quad \sigma_A^2 = 1.67 \\ \bar{f}_B &= 4.75, \quad \sigma_B^2 = 0.92 \\ \bar{f}_C &= 6.00, \quad \sigma_C^2 = 0.67 \end{aligned} \tag{В.29}$$

Затем мы применяем уравнение (В.26) для вычисления статистического показателя F:

$$\begin{aligned} \bar{f} &= 4.75 \\ S_w &= 1.08 \\ S_b &= 6.25 \\ F &= 5.77 \end{aligned} \tag{B.30}$$

Далее применяем уравнение (B.27) для вычисления степеней свободы:

$$\begin{aligned} D_n &= 2 \\ D_d &= 9 \end{aligned} \tag{B.31}$$

Теперь мы смотрим в табл. B.3, которая является таблицей 5%-ных *F*-тестов, и видим, что для этих значений степени свободы статистический показатель *F* равен 4.25. Наш статистический показатель *F* равен 5.77, поэтому если результативность алгоритмов *A*, *B* и *C* принципиально эквивалентна, то вероятность того, что мы получим результаты, показанные в уравнении (B.28), равна менее 5 %. Эквивалентным образом мы можем сказать, что если результативность алгоритмов *A*, *B* и *C* эквивалентна, то вероятность равна больше чем 95 %, что мы увидим меньшие вариации, чем те, которые показаны в уравнении (B.28). Мы также можем посмотреть в табл. B.4, которая является таблицей 1%-ных *F*-тестов, чтобы увидеть, что для $D_n = 2$ и $D_d = 9$ статистический показатель *F* равен 8.02. Наш статистический показатель *F* равен 5.77, поэтому если результативность алгоритмов *A*, *B* и *C* принципиально эквивалентна, то вероятность равна больше чем 1 %, что мы увидим различия, показанные в уравнении (B.28). Эквивалентным образом мы можем сказать, что если результативность алгоритмов *A*, *B* и *C* эквивалентна, то вероятность равна менее 99 %, что мы увидим меньшие вариации, чем те, которые показаны в уравнении (B.28). Объединив эти результаты с помощью простой линейной интерполяции, мы заключаем

$$\begin{aligned} F = 4.25 \quad &\text{дает} \quad p = 5\,\% \\ F = 8.02 \quad &\text{дает} \quad p = 1\,\% \\ \text{следовательно,} \quad F = 5.77 \quad &\text{дает} \quad p \approx 3.4\,\% \end{aligned} \tag{B.32}$$

То есть если результативность алгоритмов *A*, *B* и *C* принципиально одинакова, то вероятность того, что мы увидим различия, показанные в уравнении, составляет около 3.4 % (B.28). На этом этапе мы можем выполнить попарные *t*-тесты или более простые тесты, чтобы увидеть, где лежат различия в алгоритмах. Уравнение (B.29) показывает, что алгоритм *C* значительно отличается от алгоритмов *A* и *B*, поэтому мы мо-

жем заключить, что вероятность 3.4 % *F*-теста в основном обусловлена алгоритмом *C*.

■

Как и проверка гипотез на основе статистического показателя *t*, проверка гипотез на основе статистического показателя *F* исходит из допущения, что результаты каждого эксперимента подчиняются гауссовому распределению. В отличие от *t*-теста, на результаты *F*-теста могут сильно влиять ненормальные распределения, поэтому результаты *F*-теста могут быть неверными, если нарушено гауссово допущение. В этих случаях мы можем использовать непараметрические тесты, которые не исходят из допущения, что данные подчиняются какой-либо конкретной функции распределения вероятностей. В распоряжении исследователя эволюционных алгоритмов имеется целый ряд непараметрических статистических проверок гипотез [Good и Hardin, 2009], [Kanji, 2006], включая широко используемый тест Уилкоксона [Corder и Foreman, 2009].

B.3. Заключение

Проницательный читатель заметит, что эта книга нарушает многие принципы, описанные в данном приложении. Например, при сравнении разных алгоритмов и вариаций эволюционных алгоритмов этой книги мы не проводили статистического тестирования. Это несоответствие является целенаправленным, а не из-за лени, притворства или цейтнота. Мы могли бы легко включить статистические проверки гипотез в эту книгу. Но цель данной книги – прежде всего инструктирующая, а не научно-исследовательская. Поэтому главы не выложены и не представлены таким образом, который подходит для научного журнала или научной монографии. Если бы мы в этой книге придерживались стандартов рецензирования, то она была бы переполнена техническими и числовыми деталями, которые затмили бы простоту алгоритмов и результатов.

Общая цель настоящей книги – предоставить простую, широкую, практичную базовую образовательную подготовку в области эволюционных алгоритмов. Конкретная цель данного приложения – побудить исследователей и рецензентов тщательнее обдумывать стандарты эволюционно-алгоритмического исследования. Некоторые дополнительные ресурсы, которые предоставляют отличные рекомендации для проведения эволюционно-алгоритмических экспериментов и подготовки отчетности о результатах, включают публикации [Barr и соавт.,] и [Crepinsek и соавт., 2013].

Приложение C

Эталонные оптимизационные функции

…Алгоритмы могут быть кастомизированы для отдельного множества тестовых задач; если тестовые задачи не представляют типы задач, для которых эволюционные алгоритмы лучше всего подходят на практике, то это вызывает беспокойство.

– Даррелл Уитли [Whitley и соавт., 1996]

В этом приложении представлено несколько стандартных эталонных оптимизационных задач, которые могут применяться для сравнения оптимизационных алгоритмов. Мы используем $x = [x_1, \ldots, x_n]$ для обозначения n-мерной области функции и $f(x)$ для обозначения значения скалярной функции.

В приложении C.1 представлены эталоны неограниченной оптимизации, в приложении C.2 – эталоны ограниченной оптимизации, в приложении C.3 – эталоны многокритериальной оптимизации, для которых $f(x)$ – это вектор. В приложении C.4 даны эталоны динамической оптимизации, в приложении C.5 – эталоны шумной оптимизации, приложение B.6 включает эталоны задачи коммивояжера.

Некоторые оптимизационные алгоритмы естественным образом смещены в сторону определенных типов поисковых пространств. Вследствие этого важно модифицировать эталоны этого приложения путем встраивания в задачу смещений и матрицы поворота. Мы обсуждаем этот важный момент в приложении C.7.

Эталоны полезны и важны для получения сравнительных результатов между разными эволюционными алгоритмами. Но в конечном ито-

ге интереснее и полезнее тестировать оптимизационные алгоритмы на задачах, имеющих приложения в реальном мире.

С.1. Неограниченные эталоны

Задача состоит в том, чтобы минимизировать $f(x)$ по всем x. Мы используем x^* для обозначения оптимизирующего значения x, и $f(x^*)$ – это минимальное значение $f(x)$:

$$x^* = \arg\min_x f(x). \tag{С.1}$$

Многие эталоны, которые мы демонстрируем в этом разделе, взяты из публикаций [Bäck, 1996], [Cai и Wang, 2006] и [Yao и соавт., 1999]. Подробную информацию о неограниченных эталонах и оценочных метрических показателях для эволюционно-алгоритмических конкурсов на Конгрессе IEEE по эволюционным вычислениям 2005 года можно найти в публикации [Suganthan и соавт., 2005]; публикация [Ali и соавт., 2005] также содержит ряд неограниченных эталонов. В книге [Floudas и соавт., 2010] весь материал посвящен определению неограниченных оптимизационных эталонов. Мы сузили представленные здесь эталоны, ограничившись только теми функциями, которые могут быть определены для любого числа размерностей n. Был предложен ряд других эталонов, в том числе с фиксированным числом размерностей. Но мы думаем, что интереснее тестировать оптимизационные алгоритмы на эталонах, чья размерность может варьироваться, что дает возможность разведывать результативность как функцию числа размерностей. Области, которые мы указываем в следующих далее подразделах, являются общепринятыми, однако исследователи также используют многие другие области.

С.1.1. *Сферическая функция*

Сферическая функция задана в виде:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n} x_i^2 \\ x^* &= 0 \\ f(x^*) &= 0 \end{aligned} \tag{С.2}$$

где $x_i \in [-5.12, +5.12]$. В диссертации Кена Де Йонга данная функция называется функцией 1 [De Jong, 1975], и она является задачей 1.1 и задачей 2.17 в публикации [Schwefel, 1995]. На рис. С.1 показан график $f(x)$ в двух размерностях. Данная функция представляет собой очень простую

оптимизационную задачу, и почти любой сносный алгоритм должен уметь точно отыскивать свой минимум. Данная задача обеспечивает хороший предварительный тест для оптимизационных алгоритмов. Она также обеспечивает хороший эталон для сравнения между алгоритмами, потому что многие оптимизационные задачи приближенно квадратичны вблизи их минимума.

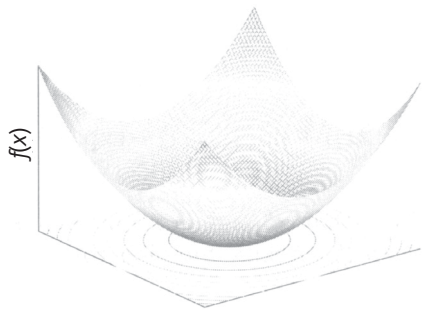


Рис. С.1. Двумерная сферическая функция

С.1.2. *Функция Экли*

Функция Экли задана в виде:

$$\begin{aligned} f(x) &= 20 + e - 20\exp\left(-0.2\sum_{i=1}^{n} x_i^2/n\right) - \exp\left(\sum_{i=1}^{n}(\cos 2\pi x_i)/n\right) \\ x^* &= 0 \\ f(x^*) &= 0 \end{aligned} \qquad \text{(С.3)}$$

где $x_i \in [-30, +30]$. Данный эталон был предложен в публикации [Ackley, 1987b]. На рис. С.2 показан график $f(x)$) в двух размерностях. Многочисленные локальные минимумы данной функции делают ее задачей для оптимизационных алгоритмов.

Рис. С.2. Двумерная функция Экли

С.1.3. *Тестовая функция Экли*

Тестовая функция Экли задана в виде:

$$f(x) = \sum_{i=1}^{n-1} 3(\cos(2x_i) + \sin(2x_{i+1})) + \exp(-0.2)\sqrt{x_i^2 + x_{i+1}^2}, \qquad \text{(C.4)}$$

где $x_i \in [-30, +30]$. Обратите внимание, что x^* и $f(x^*)$ неизвестны для этой задачи. Данный эталон аналогичен функции Экли с его многочисленными холмами и долинами, как показано на рис. С.3.

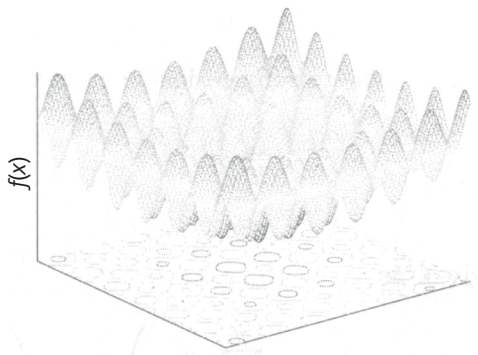


Рис. С.3. Двумерная тестовая функция Экли

С.1.4. *Функция Розенброка*

Функция Розенброка (Rosenbrock) задана в виде:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2] \\ x^* &= [1, \ldots, 1] \\ f(x^*) &= 0 \end{aligned} \qquad \text{(C.5)}$$

где $x_i \in [-2.048, +2.048]$. Данный эталон был предложен в публикации [Rosenbrock, 1960]. В диссертации Де Йонга [De Jong, 1975] он называется функцией 2, и он является задачей 2.4, 2.24 и 2.25 в публикации [Schwefel, 1995]. На рис. С.4 показан график $f(x)$ в двух размерностях. Данная функция имеет длинную, узкую, бананообразную долину, что делает ее задачей для оптимизационных алгоритмов.

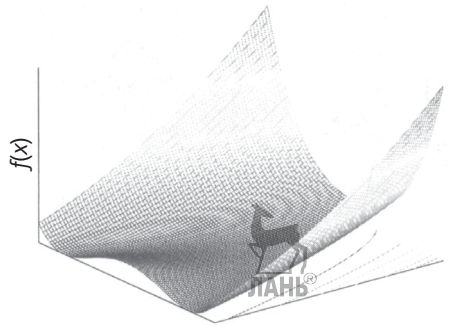


Рис. С.4. Двумерная функция Розенброка

С.1.5. *Функция Флетчера*

Функция Флетчера, так называемая функция Флетчера-Пауэлла, имеет вид:

$$
\begin{aligned}
f(x) &= \sum_{i=1}^{n} (A_i - B_i) \\
A_i &= \sum_{i=1}^{n} (a_{ij} \sin \alpha_j + b_{ij} \cos \alpha_j) \\
B_i &= \sum_{i=1}^{n} (a_{ij} \sin x_j + b_{ij} \cos x_j) \\
\alpha_i &\in [-\pi, \pi], \quad i \in \{1, \ldots, n\} \\
a_{ij} b_{ij} &\in [-100, 100], \quad i, j \in \{1, \ldots, n\} \\
x^* &= \alpha \\
f(x^*) &= 0
\end{aligned}
\tag{С.6}
$$

где $x_i \in [-\pi, +\pi]$. Данный эталон был предложен в работе [Fletcher и Powell, 1963] и называется задачей 2.13 в публикации [Schwefel, 1995]. На рис. С.5 показан график $f(x)$ в двух размерностях для конкретных значений a_{ij}, b_{ij} и α_i. Данная функция интересна тем, что изменяется с каждой реализацией a_{ij}, b_{ij} и α_i. Эти параметры часто задаются с помощью равномерного генератора случайных чисел.

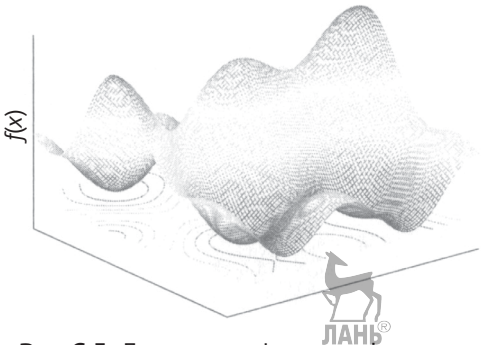


Рис. С.5. Двумерная функция Флетчера

С.1.6. *Функция Гриванка*

Функция Гриванка (Griewank), которая иногда пишется как Griewangk, задана в виде:

$$\begin{aligned} f(x) &= 1 + \sum_{i=1}^{n} x_i^2/4000 - \prod_{i=1}^{n} \cos(x_i/\sqrt{i}) \\ x^* &= 0 \\ f(x^*) &= 0 \end{aligned} \qquad \text{(C.7)}$$

где $x_i \in [-600, +600]$. Данный эталон обсуждается в публикации [Bäck и соавт., 1997a, раздел B2.7]. На рис. С.6 показан график $f(x)$ в двух размерностях. Функция имеет много локальных оптимумов, и сомножитель в $f(x)$ вызывает большую взаимозависимость между компонентами x.

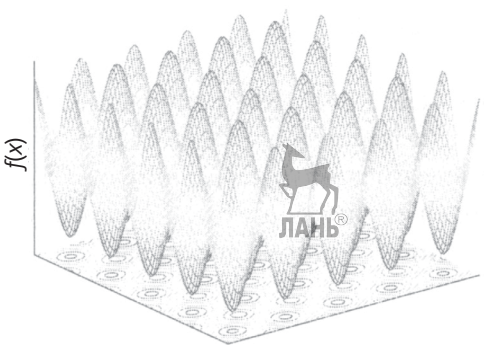


Рис. С.6. Двумерная функция Гриванка

С.1.7. *Штрафная функция #1*

Штрафная функция #1 задана в виде:

$$f(x) = \frac{\pi}{n}\left\{10\sin^2(\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2[1 + 10\sin^2(\pi x_{i+1})] + (x_i - 1)^2\right\} + \sum_{i=1}^{n} u_i$$

$$u_i = \begin{cases} k(x_i - a)^m & x_i > a \\ 0 & -a \le x_i \le a \\ k(-x_i - a)^m & x_i < -a \end{cases}$$

$$y_i = 1 + (x_i + 1)/4$$

$$x^* = [1, \ldots, 1]$$

$$f(x^*) = 0 \qquad \text{(С.8)}$$

где $x_i \in [-50, +50]$. Данный эталон приводится в публикации [Yao и соавт., 1999]. Значения для k, a и m не заданы, но мы обычно используем $k = 100$, $a = 10$ и $m = 4$. На рис. С. 7 показан график $f(x)$ в двух размерностях. Данная функция имеет только один минимум, но эта функция очень мелкая в минимуме, поэтому найти минимум с высокой точностью непросто.

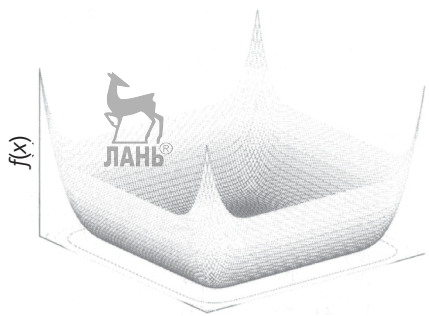


Рис. С.7. Двумерная штрафная функция #1

С.1.8. *Штрафная функция #2*

Штрафная функция #2 задана в виде:

$$f(x) = \sum_{i=1}^{n} u_i + 0.1\left\{10\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2[1 + \sin^2(3\pi x_{i+1})] + (x_n - 1)^2[1 + \sin^2(2\pi x_n)]\right\}$$

$$x^* = [1, \ldots, 1]$$
$$f(x^*) = 0 \qquad \text{(C.9)}$$

где $x_i \in [-50, +50]$ и u_i задано в уравнении (C.8). Данный эталон приводится в публикации [Yao и соавт., 1999]. Как и штрафная функция #1, значения для k, a и m не заданы, но мы обычно используем $k = 100$, $a = 5$ и $m = 4$. На рис. C.8 показан график $f(x)$ в двух размерностях. Как и штрафная функция #1, штрафная функция #2 имеет только один минимум, но эта функция очень мелкая в минимуме, поэтому трудно найти минимум с высокой точностью.

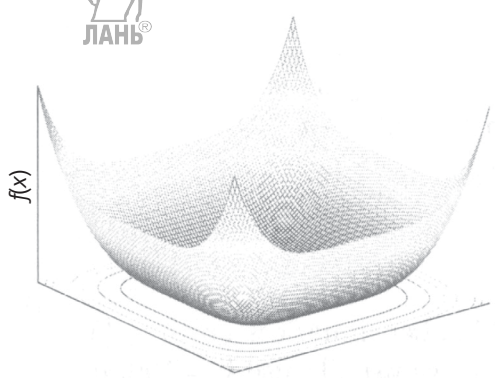


Рис. C.8. Двумерная штрафная функция #2

C. *1.9. Квартичная функция*

Квартичная функция (четвертой степени) задана в виде:

$$f(x) = \sum_{i=1}^{n} i\, x_i^4$$
$$x^* = 0$$
$$f(x^*) = 0 \qquad \text{(C.10)}$$

где $x_i \in [-1.28, +1.28]$. Нередко в $f(x)$ добавляется шум, но это не изменяет аргумент минимума. В диссертации Де Йонга [De Jong, 1975] данный эталон называется функцией 4, и он также приводится в публикации [Yao и соавт., 1999]. Квартичная функция может быть записана с x_i, возведенным во вторую, а не в четвертую степень, и в этом случае она называется гиперэллипсоидной функцией, или взвешенной сферичес-

кой функцией [Ros и Hansen, 2008]. Однако иногда гиперэллипсоидная функция записывается следующим образом [Yao и Liu, 1997]:

$$f(x) = \sum_{i=1}^{n} 2^i x_i^2. \tag{C.11}$$

На рис. С.9 показан график $f(x)$ в двух размерностях. Как и штрафные функции, квартичная функция имеет только один минимум, но эта функция очень мелкая в минимуме, поэтому трудно найти минимум с высокой точностью.

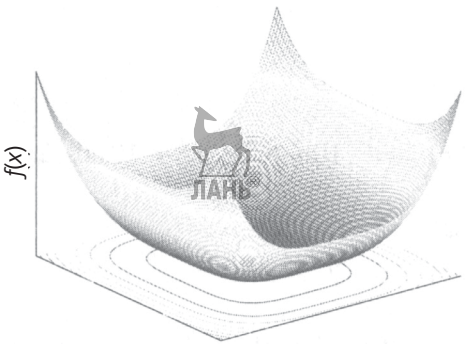


Рис. С.9. Двумерная квартичная функция

С.1.10. *Десятистепенная функция*

Десятистепенная функция задана в виде:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n} x_i^{10} \\ x^* &= 0 \\ f(x^*) &= 0 \end{aligned} \tag{C.12}$$

где $x_i \in [-5.12, +5.12]$. Данный эталон был предложен в публикации [Schwefel, 1995] в качестве задачи 2.23, а также приводится в публикации [Yao и соавт., 1999]. На рис. С.10 показан график $f(x)$ в двух размерностях. Как и квартичная и штрафные функции, десятистепенная функция имеет только один минимум, но эта функция очень мелкая в минимуме, поэтому трудно найти минимум с высокой точностью.

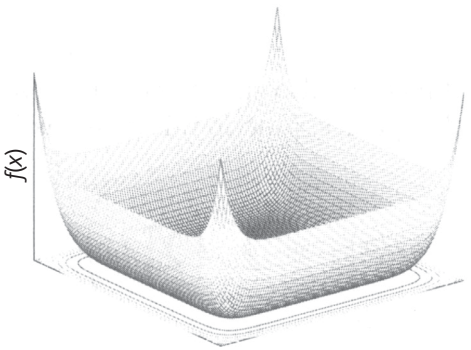


Рис. С.10. Двумерная десятистепенная функция

С.1.11. *Функция Растригина*

Функция Растригина задана в виде:

$$\begin{aligned} f(x) &= 10n + \sum_{i=1}^{n} x_i^2 - 10\cos(2\pi x_i) \\ x^* &= 0 \\ f(x^*) &= 0 \end{aligned} \qquad \text{(С.13)}$$

где $x_i \in [-5.12, +5.12]$. Данный эталон был предложен в публикации [Rastrigin, 1974], а также приводится в публикации [Yao и соавт., 1999]. На рис. С.11 показан график $f(x)$ в двух размерностях. Функция Растригина похожа на функцию Гриванка. Число локальных минимумов в функции Растригина экспоненциально увеличивается вместе с n [Beyer и Schwefel, 2002].

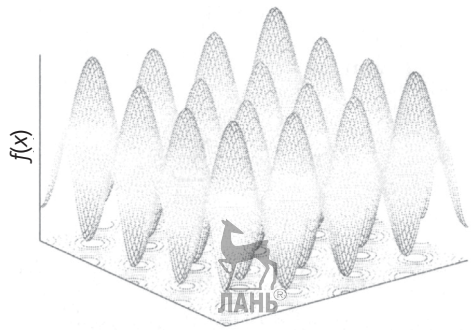


Рис. С.11. Двумерная функция Растригина

С.1.12. *Функция двойной суммы Швефеля*

Функция двойной суммы Швефеля, так называемая функция хребта Швефеля [Price и соавт., 2005], Швефель 1.2 и квадратичная функция, задана в виде:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n}\left(\sum_{j=1}^{i} x_j\right)^2 \\ x^* &= 0 \\ f(x^*) &= 0 \end{aligned} \tag{С.14}$$

где $x_i \in [-65.536, +65.536]$. Данный эталон-тест также называется повернутой гиперэллипсоидной функцией [Ros и Hansen, 2008]. Он был предложен в публикации [Schwefel, 1995] в качестве задачи 1.2 и задачи 2.9, а также приводится в публикации [Yao и соавт., 1999]. В данной квадратичной функции число условий пропорционально n^2. На рис. С.12 показан график $f(x)$ в двух размерностях.

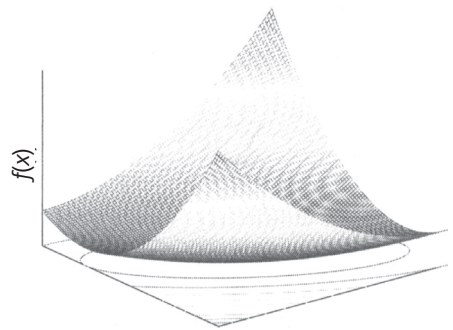


Рис. С.12. Двумерная функция двойной суммы Швефеля

С.1.13. *Функция max Швефеля*

Функция max Швефеля, так называемая функция Швефеля 2.21, имеет вид:

$$\begin{aligned} f(x) &= \max_i(|x_i| \ : \ i \in \{1, \ldots, n\}) \\ x^* &= 0 \\ f(x^*) &= 0 \end{aligned} \tag{С.15}$$

где $x_i \in [-100, +100]$. Данный эталон был предложен в публикации [Schwefel, 1995], а также приводится в публикации [Yao и соавт., 1999].

Эта функция является недифференцируемой. На рис. С.13 показан график $f(x)$ в двух размерностях.

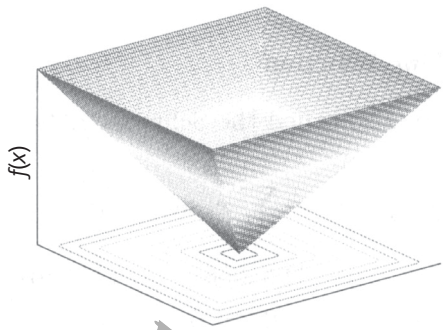


Рис. С.13. Двумерная функция max Швефеля

С.1.14. *Функция Швефеля абсолютной величины*

Функция Швефеля абсолютной величины, так называемая функция Швефеля 2.22, задана в виде:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i| \\ x^* &= 0 \\ f(x^*) &= 0 \end{aligned} \qquad \text{(С.16)}$$

где $x_i \in [-10, +10]$. Данный эталон был предложен в качестве задачи 2.22 в публикации [Schwefel, 1995], а также приводится в публикации [Yao и соавт., 1999]. Эта функция является недифференцируемой. На рис. С.14 показан график $f(x)$ в двух размерностях.

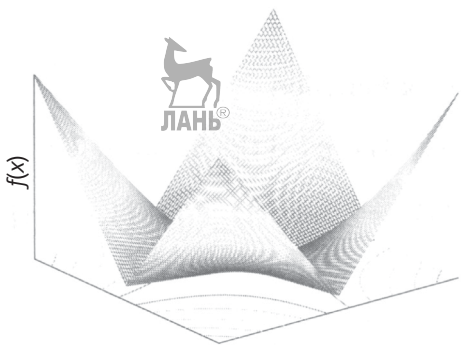


Рис. С.14. Двумерная функция Швефеля абсолютной величины

С.1.15. *Синусоидальная функция Швефеля*

Синусоидальная функция Швефеля, так называемая функция Швефеля 2.26, задана в виде:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n} x_i \sin\sqrt{|x_i|} \\ x^* &= [420.9687, \ldots, 420.9867] \\ f(x^*) &= -12965.5 \end{aligned} \tag{С.17}$$

где $x_i \in [-500, +500]$. Данный эталон был предложен в публикации [Schwefel, 1995] в качестве задач 2.3 и 2.26, а также приводится в публикации [Yao и соавт., 1999]. Эта функция имеет много местных минимумов. На рис. С.15 показан график $f(x)$ в двух размерностях.

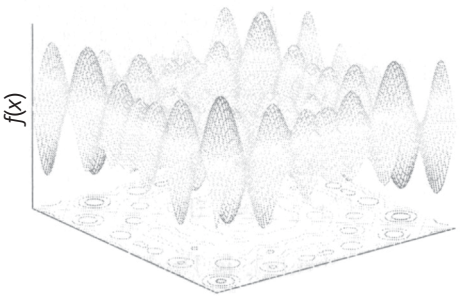


Рис. С.15. Двумерная синусоидальная функция Швефеля

С.1.16. *Ступенчатая функция*

Ступенчатая функция имеет вид:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n} (\text{floor}(x_i + 0.5))^2 \\ x^* &= 0 \\ f(x^*) &= 0 \end{aligned} \tag{С.18}$$

где $x_i \in [-100, +100]$ и где функция `floor` возвращает наименьшее целое число, меньшее или равное ее аргументу. Данный эталон в диссертации Де Йонга [De Jong, 1975] называется функцией 3, и он также приводится в публикации [Yao и соавт., 1999]. Эта функция не дифференцируема

и имеет много плато. На рис. С.16 показан график $f(x)$ в двух размерностях.

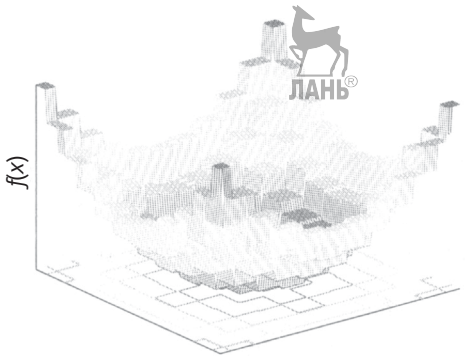


Рис. С.16. Двумерная ступенчатая функция

С.1.17. *Функция абсолютной величины*

Функция абсолютной величины задана в виде:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n} |x_i| \\ x^* &= 0 \\ f(x^*) &= 0 \end{aligned} \qquad \text{(С.19)}$$

где $x_i \in [-10, +10]$. Данный эталон является задачей 2.20 в публикации [Schwefel, 1995]. Эта функция не дифференцируема. На рис. С.17 показан график $f(x)$ в двух размерностях.

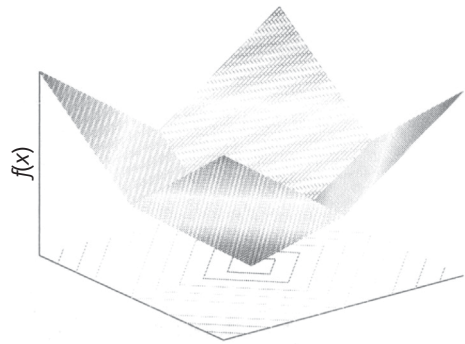


Рис. С.17. Двумерная функция абсолютной величины

С.1.18. *Функция лисьей норы Шекеля*

Функция лисьей норы Шекеля задана в виде:

$$\begin{aligned} f(x) &= \left[\frac{1}{500}+\sum_{j=1}^{2}5\frac{1}{j+\sum_{i=1}^{n}(x_i-a_{ij})^6}\right]^{-1} \\ x^* &= [-32,\ldots,-32] \\ f(x^*) &\approx 1 \end{aligned} \tag{С.20}$$

где $x_i \in [-65.536, +65.536]$ и где a_{ij} – это элемент в i-й строке и j-м столбце a. Для двух размерностей ($n = 2$) a задана в виде:

$$\begin{aligned} a &= \begin{bmatrix} b_0 & \cdots & b_0 \\ b_1 & \cdots & b_5 \end{bmatrix} \\ b_0 &= [\,-32 \quad -16 \quad 0 \quad 16 \quad 32\,] \\ b_i &= (16(i-1)-32)\,[\,1 \quad 1 \quad 1 \quad 1 \quad 1\,] \end{aligned} \tag{С.21}$$

Данный эталон называется функцией 5 в диссертации Де Йонга [De Jong, 1975], и он также приводится в [Yao и соавт., 1999]. Эта функция имеет несколько локальных минимумов, не все из которых имеют одинаковое значение, и она имеет резкое падение до минимума, как показано на рис. С.18. Дополнительные строки могут быть увеличены до a, если $n > 2$ citeBersini.

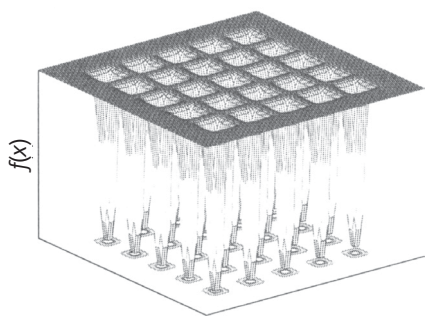


Рис. С.18. Двумерная функция лисьей норы Шекеля

С.1.19. *Функция Михалевича*

Функция Михалевича задана в виде:

$$f(x) = -\sum_{i=1}^{n}\sin x_i \sin^{2m}(i x_i^2/\pi), \tag{С.22}$$

где $x_i \in [0, \pi]$ и m – это параметр, управляющий сложностью поиска. Обратите внимание, что x^* и $f(x^*)$ для этой задачи неизвестны. Данный эталон приводится в публикации [Michalewicz, 1996]. Эта функция имеет длинные узкие долины с резким падением до минимума, как показано на рис. С.19.

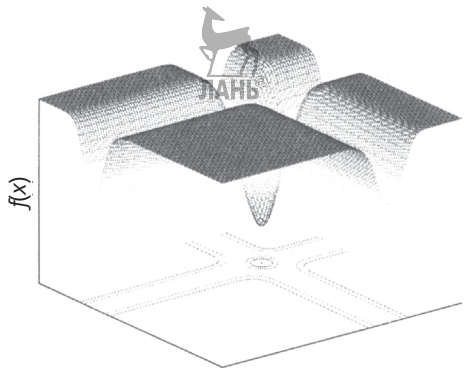


Рис. С.19. Двумерная функция Михалевича с $m = 10$

С.1.20. *Функция синусоидальной огибающей*

Функция синусоидальной огибающей задана в виде:

$$f(x) = -\sum_{i=1}^{n-1} \frac{\sin^2 \sqrt{x_i + x_{i+1} - 0.5}}{(0.001(x_i^2 + x_{i+1}^2) + 1)^2}, \qquad \text{(С.23)}$$

где $x_i \in [-100, +100]$. Обратите внимание, что x^* и $f(x^*)$ для этой задачи неизвестны. Данный эталон, также называемый функцией Шеффера [Cheng и соавт., 2008], имеет много долин и локальных минимумов, как показано на рис. С.20.

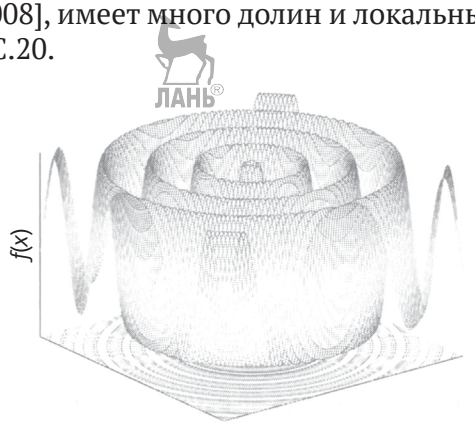


Рис. С.20. Двумерная синусоидальная огибающая

С.1.21. *Функция в форме упаковки для яиц*

Функция в форме упаковки для яиц (eggholder) задана в виде:

$$f(x) = -\sum_{i=1}^{n-1}(x_{i+1}+47)\sin\sqrt{|x_{i+1}+x_i/2+47|}+x_i\sin\sqrt{|x_i-x_{i+1}-47|}, \quad \text{(С.24)}$$

где $x_i \in [-512, +512]$. Обратите внимание, что x^* и $f(x^*)$ для этой задачи неизвестны. Данный эталон приводится в публикации [Wu и Chow, 2007]. Его двумерный график показан на рис. С.21.

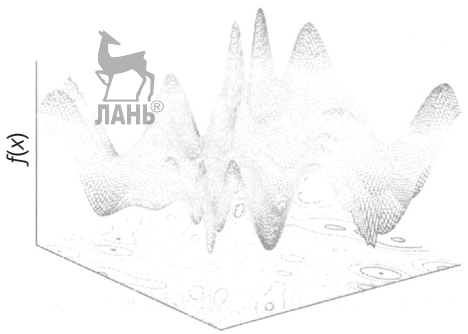


Рис. С.21. Двумерная функция в форме упаковки для яиц

С.1.22. *Функция Вейерштрасса*

Функция Вейерштрасса задана в виде:

$$f(x) = \sum_{i=1}^{n}\left\{\sum_{k=0}^{k_{\max}}\left[a^k\cos\left(2\pi b^k(x_i+0.5)\right)\right]\right\} - n\sum_{k=0}^{k_{\max}}\left[a^k\cos(\pi b^k)\right]$$

$$x^* = 0$$

$$f(x^*) = 0 \quad \text{(С.25)}$$

где $x_i \in [-5, +5]$, $a = 0.5$, $b = 3$ и $k_{\max} = 20$. Данный эталон приводится в публикации [Liang и соавт., 2005]. У него есть интересное свойство, что, по мере того как $n \to \infty$, она везде непрерывна, но нигде не дифференцируема, и она везде немонотонна. График двумерной функции Вейерштрасса показан на рис. С.22.

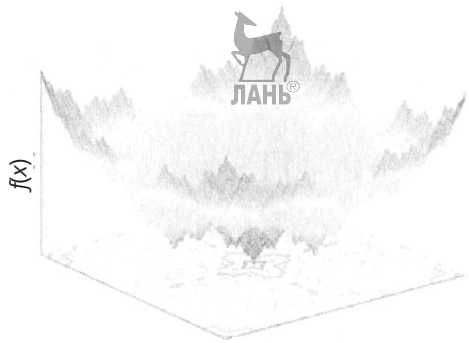


Рис. С.22. Двумерная функция Вейерштрасса

С.2. Ограниченные эталоны

Ограниченная оптимизационная задача предусматривает минимизацию $f(x)$ по всем x так, что $x \in \mathcal{F} \in \mathcal{R}^n$, где $\mathcal{F}$ – это допустимое множество и n – размерность задачи. Мы используем x^* для обозначения оптимизирующего значения x, и $f(x^*)$ – это ограниченный минимум $f(x)$:

$$x^* = \arg\min_x f(x)$$

$$\text{такой, что} \quad g_i(x) \le 0 \ \text{ для } i \in [1, m] \text{ и } h_j(x) = 0 \ \text{ для } j \in [1, p]. \quad \text{(С.26)}$$

Данная задача включает $(m + p)$ ограничений, m из которых являются ограничениями в виде неравенства, а p – ограничениями в виде равенства. Многие задачи такой формы имеют длинные и запутанные формы для $f(x)$, $g_i(x)$ и $h_j(x)$, и поэтому она требует много места, просто чтобы записать задачу. Поэтому в данном разделе мы показываем только простые ограниченные эталоны, а также ссылки на публикации, где можно найти более длинные и более сложные эталоны.

Ограниченные эталонные функции приводятся в публикациях [Araujo и соавт., 2009], [Coello Coello, 2000A], [Coello Coello, 2002], [Deb, 2000], [Mezura-Montes и Coello Coello, 2005] и [Runarsson и Yao, 2000]. Подробную информацию об ограниченных эталонах и оценочных метрических показателях для эволюционно-алгоритмических конкурсов на Конгрессе IEEE по эволюционным вычислениям 2006 и 2010 годов можно найти в публикациях [Liang и соавт., 2006] и [Mallipeddi и Suganthan, 2010]. Отметим, что книга [Floudas и Pardalos, 1990] целиком посвящена определению ограниченных оптимизационных эталонов. Ограниченные многокритериальные эталоны можно найти в публикации [Deb и соавт., 2001].

Ограниченные эталоны в данном разделе взяты из публикации [Mallipeddi и Suganthan, 2010] и использовались в конкурсе по ограниченным эволюционным алгоритмам на Конгрессе по эволюционным вычислениям 2010 года (CEC). В приведенных ниже постановках задачи мы используем o_i для обозначения случайного сдвига и M для обозначения случайной матрицы поворота (см. раздел С.7).

С.2.1. *Функция C01*

Функция C01 имеет вид:

$$\begin{aligned} f(x) &= -\left|\frac{\sum_{i=1}^{n}\cos^4 z_i - 2\prod_{i=1}^{n}\cos^2 z_i}{\sum_{i=1}^{n} iz_i^2}\right| \\ g_1(x) &= 0.75 - \prod_{i=1}^{n} z_i \le 0 \\ g_2(x) &= \sum_{i=1}^{n} z_i - 7.5n \le 0 \\ x_i &\in [0, 10] \end{aligned} \tag{С.27}$$

где $z_i = x_i - o_i$ для $i \in [1, n]$.

С.2.2. *Функция C02*

Функция C02 задается в виде:

$$\begin{aligned} f(x) &= \max_x z_i \\ g_1(x) &= 10 - \frac{1}{n}\sum_{i=1}^{n}(z_i^2 - 10\cos(2\pi z_i) + 10) \le 0 \\ g_2(x) &= \frac{1}{n}\sum_{i=1}^{n}(z_i^2 - 10\cos(2\pi z_i) + 10) - 15 \le 0 \\ h(x) &= \frac{1}{n}\sum_{i=1}^{n}(y_i^2 - 10\cos(2\pi y_i) + 10) - 20 = 0 \\ x_i &\in [-5.12, +5.12] \end{aligned} \tag{С.28}$$

где $z_i = x_i - o_i$ и $y_i = z_i - 0.5$ для $i \in [1, n]$.

С.2.3. *Функция C03*

Функция C03 имеет вид:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n-1} [100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2] \\ h(x) &= \sum_{i=1}^{n-1} (z_i - z_{i+1})^2 = 0 \\ x_i &\in [-1000, 1000] \end{aligned} \qquad \text{(C.29)}$$

где $z_i = x_i - o_i$ для $i \in [1, n]$.

С.2.4. *Функция C04*

Функция C04 задана в виде:

$$\begin{aligned} f(x) &= \max_x z_i \\ h_1(x) &= \frac{1}{n}\sum_{i=1}^{n} z_i \cos\sqrt{|z_i|} = 0 \\ h_2(x) &= \sum_{i=1}^{n/2-1} (z_i - z_{i+1})^2 = 0 \\ h_3(x) &= \sum_{i=n/2+1}^{n/2-1} (z_i^2 - z_{i+1}) = 0 \\ h_4(x) &= \sum_{i=1}^{n} z_i = 0 \\ x_i &\in [-50, 50] \end{aligned} \qquad \text{(C.30)}$$

где $z_i = x_i - o_i$ для $i \in [1, n]$.

С.2.5. *Функция C05*

Функция C05 задана в виде:

$$\begin{aligned} f(x) &= \max_x z_i \\ h_1(x) &= \frac{1}{n}\sum_{i=1}^{n} [-z_i \sin(\sqrt{|z_i|})] = 0 \end{aligned}$$

$$\begin{aligned} h_2(x) &= \frac{1}{n}\sum_{i=1}^{n}\left[-z_i \cos\left(0.5\sqrt{|z_i|}\right)\right] = 0 \\ x_i &\in [-600, 600] \end{aligned} \quad \text{(C.31)}$$

где $z_i = x_i - o_i$ для $i \in [1, n]$.

С.2.6. *Функция C06*

Функция C06 задана в виде:

$$\begin{aligned} f(x) &= \max_x z_i \\ y_i &= (z_i + 483.6106156535)M - 483.6106156535 \\ h_1(x) &= \frac{1}{n}\sum_{i=1}^{n}\left[-y_i \sin\left(\sqrt{|y_i|}\right)\right] = 0 \\ h_2(x) &= \frac{1}{n}\sum_{i=1}^{n}\left[-y_i \cos\left(0.5\sqrt{|y_i|}\right)\right] = 0 \\ x_i &\in [-600, 600] \end{aligned} \quad \text{(C.32)}$$

где $z_i = x_i - o_i$ для $i \in [1, n]$.

С.2.7. *Функция C07*

Функция C07 имеет вид:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n-1}\left[100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2\right] \\ g(x) &= 0.5 - \exp\left[\frac{0.1}{n}\sum_{i=1}^{n} y_i^2\right] - 3\exp\left[\frac{1}{n}\sum_{i=1}^{n}\cos(0.1 y_i)\right] + \exp(1) \le 0 \\ x_i &\in [-140, 140] \end{aligned} \quad \text{(C.33)}$$

где $y_i = x_i - o_i$ и $z_i = x_i - o_i + 1$ для $i \in [1, n]$.

С.2.8. *Функция C08*

Функция C08 задается в виде:

$$f(x) = \sum_{i=1}^{n-1}\left[100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2\right]$$

$$\begin{aligned} g(x) &= 0.5 - \exp\left[\frac{0.1}{n}\sum_{i=1}^{n} y_i^2\right] - 3\exp\left[\frac{1}{n}\sum_{i=1}^{n}\cos(0.1 y_i)\right] + \exp(1) \le 0 \\ x_i &\in [-140, 140] \end{aligned} \quad \text{(C.34)}$$

где $y_i = (x_i - o_i)M$ и $z_i = x_i - o_i + 1$ для $i \in [1, n]$.

C.2.9. *Функция C09*

Функция C09 имеет вид:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n-1} [100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2] \\ h(x) &= \frac{1}{n}\sum_{i=1}^{n} y \sin\sqrt{|y_i|} = 0 \\ x_i &\in [-500, 500] \end{aligned} \quad \text{(C.35)}$$

где $y_i = x_i - o_i$ и $z_i = x_i + 1 - o_i$ для $i \in [1, n]$.

C.2.10. *Функция C10*

Функция C10 имеет вид:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n-1} [100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2] \\ h(x) &= \frac{1}{n}\sum_{i=1}^{n} y_i \sin\sqrt{|y_i|} = 0 \\ x_i &\in [-500, 500] \end{aligned} \quad \text{(C.36)}$$

где $y_i = (x_i - o_i)M$ и $z_i = x_i + 1 - o_i$ для $i \in [1, n]$.

C.2.11. *Функция C11*

Функция C11 задана в виде:

$$\begin{aligned} f(x) &= \frac{1}{n}\sum_{i=1}^{n} \left[-z_i \cos\left(2\sqrt{|z_i|}\right)\right] \\ h(x) &= \sum_{i=1}^{n-1} [100(y_i^2 - y_{i+1})^2 + (y_i - 1)^2] = 0 \\ x_i &\in [-100, 100] \end{aligned} \quad \text{(C.37)}$$

где $y_i = x_i + 1 - o_i$ и $z_i = (x_i - o_i)M$ для $i \in [1, n]$.

С.2.12. *Функция C12*

Функция C12 имеет вид:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n} z_i \sin\sqrt{|z_i|} \\ h(x) &= \sum_{i=1}^{n} (z_i^2 - z_{i+1})^2 = 0 \\ g(x) &= \sum_{i=1}^{n} [z_i - 100\cos(0.1 z_i) + 10] \le 0 \\ x_i &\in [-1000, 1000] \end{aligned} \quad \text{(C.38)}$$

где $z_i = x_i - o_i$ для $i \in [1, n]$.

С.2.13. *Функция C13*

Функция C13 задана в виде:

$$\begin{aligned} f(x) &= \frac{1}{n}\sum_{i=1}^{n} \left[-z_i \sin\sqrt{|z_i|}\right] \\ g_1(x) &= -50 + \frac{1}{100n}\sum_{i=1}^{n} z_i^2 \le 0 \\ g_2(x) &= \frac{50}{n}\sum_{i=1}^{n} \sin\left(\frac{\pi z_i}{50}\right) \le 0 \\ g_3(x) &= 75 - 50\left[\sum_{i=1}^{n} \frac{z_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{z_i}{\sqrt{i}}\right) + 1\right] \le 0 \\ x_i &\in [-500, 500] \end{aligned} \quad \text{(C.39)}$$

где $z_i = x_i - o_i$ для $i \in [1, n]$.

С.2.14. *Функция C14*

Функция C14 имеет вид:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n-1} [100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2] \\ g_1(x) &= \sum_{i=1}^{n} \left[-y_i \cos\sqrt{|y_i|}\right] - n \le 0 \end{aligned}$$

$$
\begin{aligned}
g_2(x) &= \sum_{i=1}^{n} \left[y_i \cos \sqrt{|\, y_i \,|} \right] - n \le 0 \\
g_3(x) &= \sum_{i=1}^{n} \left[y_i \sin \sqrt{|\, y_i \,|} \right] - 10n \le 0 \\
x_i &\in [-1000, 1000]
\end{aligned} \tag{C.40}
$$

где $y_i = x_i - o_i$ и $z_i = x_i - o_i + 1$ для $i \in [1, n]$.

C.2.15. *Функция C15*

Функция C15 имеет вид:

$$
\begin{aligned}
f(x) &= \sum_{i=1}^{n-1} [100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2] \\
g_1(x) &= \sum_{i=1}^{n} \left[-y_i \cos \sqrt{|\, y_i \,|} \right] - n \le 0 \\
g_2(x) &= \sum_{i=1}^{n} \left[y_i \cos \sqrt{|\, y_i \,|} \right] - n \le 0 \\
g_3(x) &= \sum_{i=1}^{n} \left[y_i \sin \sqrt{|\, y_i \,|} \right] - 10n \le 0 \\
x_i &\in [-1000, 1000]
\end{aligned} \tag{C.41}
$$

где $y_i = (x_i - o_i)M$ и $z_i = x_i - o_i + 1$ для $i \in [1, n]$.

C.2.16. *Функция C16*

Функция C16 задана в виде:

$$
\begin{aligned}
f(x) &= \sum_{i=1}^{n} \frac{z_i^2}{4000} - \prod_{i=1}^{D} \cos\left(\frac{z_i}{\sqrt{i}} \right) + 1 \\
g_1(x) &= \sum_{i=1}^{n} [z_i^2 - 100\cos(\pi z_i) + 10] \le 0 \\
g_2(x) &= \prod_{i=1}^{n} z_i \le 0 \\
h_1(x) &= \sum_{i=1}^{n} \left[z_i \sin \sqrt{|\, z_i \,|} \right] = 0
\end{aligned}
$$

$$\begin{aligned} h_2(x) &= \sum_{i=1}^{n} \left[-z_i \sin \sqrt{|z_i|} \right] = 0 \\ x_i &\in [-10, 10] \end{aligned} \tag{C.42}$$

где $z_i = x_i - o_i$ для $i \in [1, n]$.

С.2.17. *Функция C17*

Функция C17 имеет вид:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n-1} (z_i - z_{i+1})^2 \\ g_1(x) &= \prod_{i=1}^{n} z_i \le 0 \\ g_2(x) &= \sum_{i=1}^{n} z_i \le 0 \\ h(x) &= \sum_{i=1}^{n} z_i \sin\left(4\sqrt{|z_i|}\right) = 0 \\ x_i &\in [-10, 10, \end{aligned} \tag{C.43}$$

где $z_i = x_i - o_i$ для $i \in [1, n]$.

С.2.18. *Функция C18*

Функция C18 имеет вид:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n-1} (z_i - z_{i+1})^2 \\ g(x) &= \frac{1}{n} \sum_{i=1}^{n} \left[-z_i \sin \sqrt{|z_i|} \right] = 0 \\ h(x) &= \frac{1}{n} \sum_{i=1}^{n} \left[z_i \sin \sqrt{|z_i|} \right] = 0 \\ x_i &\in [-50, 50, \end{aligned} \tag{C.44}$$

где $z_i = x_i - o_i$ для $i \in [1, n]$.

С.2.19. *Резюме ограниченных эталонов*

Здесь мы приводим резюме представленных выше 18 эталонов CEC 2010. Оценочное соотношение ρ между размером допустимого

множества и размером поискового пространства указывает на то, насколько трудно удовлетворить ограничения (см. уравнение (19.53)). В табл. С.1. обобщены 18 ограниченных эталонов.

Таблица С1. Резюме 18 ограниченных оптимизационных эталонов CEC 2010. N_e – это число ограничений в виде равенства, N_i – число ограничений в виде неравенства, ρ – отношение размера допустимого множества к размеру поискового пространства для 10-мерной и 30-мерной версий каждой задачи

Функция	N_e	N_i	ρ (n = 10)	ρ (n = 30)
C01	0	2	0.997689	1.000000
C02	1	2	0.000000	0.000000
C03	1	0	0.000000	0.000000
C04	4	0	0.000000	0.000000
C05	2	0	0.000000	0.000000
C06	2	0	0.000000	0.000000
C07	0	1	0.505123	0.503725
C08	0	1	0.379512	0.375278
C09	1	0	0.000000	0.000000
C10	1	0	0.000000	0.000000
C11	1	0	0.000000	0.000000
C12	1	1	0.000000	0.000000
C13	0	3	0.000000	0.000000
C14	0	3	0.003112	0.006123
C15	0	3	0.003210	0.006023
C16	2	2	0.000000	0.000000
C17	1	2	0.000000	0.000000
C18	1	1	0.000000	0.000000

С.3. Многокритериальные эталоны

Многокритериальная оптимизационная задача предусматривает минимизацию $f(x)$ по всем x, где $f(x)$ – это вектор и x – n-мерный вектор решения. Векторная минимизация не определена в нормальном смысле этого слова, и поэтому мы определяем множество Парето P_s и фронт Парето P_f в разделе 20.1. Тогда мы можем представить многокритериаль-

ную оптимизационную задачу как задачу поиска «лучших» возможных P_s и P_f. Мы можем определить понятие «лучший» несколькими разными способами, как показано в разделе 20.2.

Подробную информацию о многокритериальных эталонах и оценочных метрических показателях для эволюционно-алгоритмических конкурсов на Конгрессе IEEE по эволюционным вычислениям 2007 и 2009 годов можно найти в публикациях [Huang и соавт., 2007] и [Zhang и соавт., 2009]. Дополнительные многокритериальные эталонные задачи можно найти в публикации [Zitzler и соавт., 2000]. Ограниченные многокритериальные эталоны можно найти в публикации [Deb и соавт., 2001]. Подходы к проектированию новых многокритериальных эталонных задач можно найти в публикациях [Deb и соавт., 2002b] и [Zhang и соавт., 2009]. В литературе предлагается ряд многокритериальных эталонов, и постоянно появляются новые. В этом разделе мы показываем только неограниченные многокритериальные оптимизационные задачи из конкурса CEC 2009 [Zhang и соавт., 2009]. Читатель может найти дополнительные многокритериальные эталоны (ограниченные и неограниченные) в приведенных выше ссылках. Размерность независимой переменной в приведенных ниже эталонах является переменной, но в конкурсе CEC 2009 использовалась $n = 30$.

С.3.1. *Неограниченная многокритериальная оптимизационная задача 1*

Данная двухкритериальная задача определяется в виде:

$$\begin{aligned} f_1(x) &= x_1 + \frac{2}{|J_1|}\sum_{j\in J_1}[x_j - \sin(6\pi x_1 + j\pi/n)]^2 \\ f_2(x) &= 1 - \sqrt{x_1} + \frac{2}{|J_2|}\sum_{j\in J_2}[x_j - \sin(6\pi x_1 + j\pi/n)]^2 \end{aligned} \qquad \text{(С.45)}$$

где множества J_1 и J_2 определяются в виде:

$$\begin{aligned} J_1 &= \{j \in [2, n] : j \text{ нечетное}\} \\ J_2 &= \{j \in [2, n] : j \text{ четное}\} \end{aligned} \qquad \text{(С.46)}$$

Поисковое пространство равно

$$\begin{aligned} x_1 &\in [0, 1] \\ x_j &\in [-1, 1] \quad \text{для } j \in [2, n] \end{aligned} \qquad \text{(С.47)}$$

Фронт Парето равен

$$\begin{aligned} f_1^* &\in [0, 1] \\ f_2^* &= 1 - \sqrt{f_1^*} \end{aligned} \tag{C.48}$$

Множество Парето равно

$$\begin{aligned} x_1^* &\in [0, 1] \\ x_j^* &= \sin(6\pi x_1 + j\pi/n) \quad \text{для } j \in [2, n] \end{aligned} \tag{C.49}$$

С.3.2. *Неограниченная многокритериальная оптимизационная задача 2*

Данная двухкритериальная задача определяется в виде:

$$\begin{aligned} f_1 &= x_1 + \frac{2}{|J_1|}\sum_{j\in J_1} y_j^2 \\ f &= 1 - \sqrt{x_1} + \frac{2}{|J_2|}\sum_{j\in J_2} y_j^2 \end{aligned} \tag{C.50}$$

где J_1 и J_2 такие же, как и в неограниченной многокритериальной оптимизационной задаче 1, и y_j определяется в виде:

$$y_j = \begin{cases} x_j - [0.3x_1^2 \cos(24\pi x_1 + 4j\pi/n) + 0.6x_1] \cos(6\pi x_1 + j\pi/n) & \text{если } j \in J_1 \\ x_j - [0.3x_1^2 \cos(24\pi x_1 + 4j\pi/n) + 0.6x_1] \sin(6\pi x_1 + j\pi/n) & \text{если } j \in J_2 \end{cases} \tag{C.51}$$

Поисковое пространство равно

$$\begin{aligned} x_1 &\in [0, 1] \\ x_j &\in [-1, 1] \quad \text{для } j \in [2, n] \end{aligned} \tag{C.52}$$

Фронт Парето равен

$$\begin{aligned} f_1^* &\in [0, 1] \\ f_2^* &= 1 - \sqrt{f_1^*} \end{aligned} \tag{C.53}$$

Множество Парето равно

$$x_1^* \in [0, 1] \tag{C.54}$$

$$x_1^* = \begin{cases} [0.3(x_1^*)^2 \cos(24\pi x_1^* + 4j\pi/n) + 0.6x_1^*] \cos(6\pi x_1^* + j\pi/n) & \text{если } j \in J_1 \\ [0.3(x_1^*)^2 \cos(24\pi x_1^* + 4j\pi/n) + 0.6x_1^*] \sin(6\pi x_1^* + j\pi/n) & \text{если } j \in J_2 \end{cases}$$

С.3.3. *Неограниченная многокритериальная оптимизационная задача 3*

Данная двухкритериальная задача определяется в виде:

$$f_1 = x_1 + \frac{2}{|J_1|}\left[4\sum_{j\in J_1} y_j^2 - 2\prod_{j\in J_1}\cos\left(20y_j\pi/\sqrt{j}\right)+2\right]$$

$$f_2 = 1-\sqrt{x_1} + \frac{2}{|J_2|}\left[4\sum_{j\in J_2} y_j^2 - 2\prod_{j\in J_2}\cos\left(20y_j\pi/\sqrt{j}\right)+2\right] \qquad \text{(С.55)}$$

где J_1 и J_2 такие же, как и в неограниченной многокритериальной оптимизационной задаче 1, и y_j определяется как

$$y_j = x_j - x_1^{0.5\,[1+3(j-2)/(n-2)]} \quad \text{для } j \in [2, n]. \qquad \text{(С.56)}$$

Поисковое пространство равно

$$x_j \in [-1, 1] \quad \text{для } j \in [1, n]. \qquad \text{(С.57)}$$

Фронт Парето равен

$$f_1^* \in [0, 1]$$
$$f_2^* = 1 - \sqrt{f_1^*} \qquad \text{(С.58)}$$

Множество Парето равно

$$x_1^* \in [0, 1]$$
$$x_j^* = (x_1^*)^{0.5\,[1+3(j-2)/(n-2)]} \quad \text{для } j \in [2, n] \qquad \text{(С.59)}$$

С.3.4. *Неограниченная многокритериальная оптимизационная задача 4*

Данная двухкритериальная задача определяется в виде:

$$f_1 = x_1 + \frac{2}{|J_1|}\sum_{j\in J_1} h(y_j) \qquad \text{(С.60)}$$

$$f = 1 - \sqrt{x_1} + \frac{2}{|J_2|}\sum_{j\in J_2} h(y_j) \qquad \text{(С.61)}$$

где J_1 и J_2 такие же, как и в неограниченной многокритериальной оптимизационной задаче 1, y_j определяется в виде:

$$y_j = x_j - \sin(6\pi x_1 + j\pi/n) \quad \text{для } j \in [2, n], \qquad \text{(С.62)}$$

и $h(\cdot)$ определяется в виде:

$$h(t) = \frac{|t|}{1 + e^{2|t|}}. \tag{C.63}$$

Поисковое пространство равно

$$\begin{aligned} x_1 &\in [0, 1] \\ x_j &\in [-2, 2] \quad \text{для } j \in [2, n] \end{aligned} \tag{C.64}$$

Фронт Парето равен

$$\begin{aligned} f_1^* &\in [0, 1] \\ f_2^* &= 1 - (f_1^*)^2 \end{aligned} \tag{C.65}$$

Множество Парето равно

$$\begin{aligned} x_1^* &\in [0, 1] \\ x_j^* &= \sin(6\pi x_1^* + j\pi/n) \quad \text{для } j \in [2, n] \end{aligned} \tag{C.66}$$

С.3.5. *Неограниченная многокритериальная оптимизационная задача 5*

Данная двухкритериальная задача определяется в виде:

$$f_1 = x_1 + \left(\frac{1}{2N} + \varepsilon\right)\left|\sin(2N\pi x_1)\right| + \frac{2}{|J_1|}\sum_{j \in J_1} h(y_j) \tag{C.67}$$

$$f_2 = 1 - x_1 + \left(\frac{1}{2N} + \varepsilon\right)\left|\sin(2N\pi x_1)\right| + \frac{2}{|J_2|}\sum_{j \in J_2} h(y_j) \tag{C.68}$$

где J_1 и J_2 такие же, как и в неограниченной многокритериальной оптимизационной задаче 1, N – это целое число ($N = 10$ в конкурсе CEC 2009), ϵ – положительное вещественное число ($\epsilon = 0.5$ в конкурсе CEC 2009), y_j определяется в виде:

$$y_j = x_j - \sin(6\pi x_1 + j\pi/n) \quad \text{для } j \in [2, n], \tag{C.69}$$

и $h(\cdot)$ определяется в виде:

$$h(t) = 2t^2 - \cos(4\pi t) + 1. \tag{C.70}$$

Поисковое пространство равно

$$\begin{aligned} x_1 &\in [0, 1] \\ x_j &\in [-1, 1] \quad \text{для } j \in [2, n] \end{aligned} \tag{C.71}$$

Фронт Парето содержит $(2N + 1)$ дискретных точек:

$$(f^*_{1i}, f^*_{2i}) = (i/(2N), 1 - i/(2N)) \quad \text{для } i \in [1, 2N + 1]. \tag{C.72}$$

Множество Парето также содержит $(2N + 1)$ дискретных точек, но они не могут быть выражены аналитически, и поэтому мы их здесь не показываем.

С.3.6. *Неограниченная многокритериальная оптимизационная задача 6*

Данная двухкритериальная задача определяется в виде:

$$\begin{aligned} f_1 &= x_1 + \max\left\{0,\ 2\left(\frac{1}{2N} + \varepsilon\right)\sin(2N\pi x_1)\right\} + z_1 \\ f_2 &= 1 - x_1 + \max\left\{0,\ 2\left(\frac{1}{2N} + \varepsilon\right)\sin(2N\pi x_1)\right\} + z_2 \end{aligned} \tag{C.73}$$

где N – это целое число ($N = 2$ в конкурсе CEC 2009), ϵ – положительное вещественное число ($\epsilon = 0.1$ в конкурсе CEC 2009), z_i определяется в виде:

$$z_i = \frac{2}{|J_i|}\left(4\sum_{j\in J_i} y_j^2 - 2\prod_{j\in J_i}\cos\left(20y_j\pi/\sqrt{j}\right) + 2\right) \quad \text{для } i \in [1, 2], \tag{C.74}$$

и множества J_1 и J_2 такие же, как в неограниченной многокритериальной оптимизационной задаче 1, и y_j определяется в виде:

$$y_j = x_j - \sin(6\pi x_1 + j\pi/n) \quad \text{для } j \in [2, n]. \tag{C.75}$$

Поисковое пространство равно

$$\begin{aligned} x_1 &\in [0, 1] \\ x_j &\in [-1, 1] \quad \text{для } j \in [2, n] \end{aligned} \tag{C.76}$$

Фронт Парето содержит одну дискретную точку (0, 1) и следующие N несвязных сегментов:

$$\begin{aligned} f_1^* &= \bigcup_{i=1}^{N}\left[\frac{2i-1}{2N}, \frac{2i}{2N}\right] \\ f_2^* &= 1 - f_1^* \end{aligned} \tag{C.77}$$

Множество Парето состоит из дискретных точек, но они не могут быть выражены аналитически, поэтому мы их здесь не показываем.

С.3.7. *Неограниченная многокритериальная оптимизационная задача 7*

Данная двухкритериальная задача определяется в виде:

$$\begin{aligned} f_1 &= x_1^{1/5} + \frac{2}{|J_1|}\sum_{j\in J_1} y_j^2 \\ f_2 &= 1 - x_1^{1/5} + \frac{2}{|J_2|}\sum_{j\in J_2} y_j^2 \end{aligned} \qquad \text{(С.78)}$$

где J_1 и J_2 такие же, как в неограниченной многокритериальной оптимизационной задаче 1, и y3 определяется как

$$y_j = x_j - \sin(6\pi + j\pi/n) \quad \text{для } j \in [2, n]. \qquad \text{(С.79)}$$

Поисковое пространство равно

$$\begin{aligned} x_1 &\in [0, 1] \\ x_j &\in [-1, 1] \quad \text{для } j \in [2, n] \end{aligned} \qquad \text{(С.80)}$$

Фронт Парето равен

$$\begin{aligned} f_1^* &\in [0, 1] \\ f_2^* &= 1 - f_1^* \end{aligned} \qquad \text{(С.81)}$$

Множество Парето равно

$$\begin{aligned} x_1^* &\in [0, 1] \\ x_j^* &= \sin(6\pi x_1 + j\pi/n) \quad \text{для } j \in [2, n] \end{aligned} \qquad \text{(С.82)}$$

С.3.8. *Неограниченная многокритериальная оптимизационная задача 8*

Данная трехкритериальная задача определяется в виде:

$$\begin{aligned} f_1 &= \cos(0.5x_1\pi)\cos(0.5x_2\pi) + \frac{2}{|J_1|}\sum_{j\in J_1}\left[x_j - 2x_2\sin(2\pi x_1 + j\pi/n)\right]^2 \\ f_2 &= \cos(0.5x_1\pi)\sin(0.5x_2\pi) + \frac{2}{|J_2|}\sum_{j\in J_2}\left[x_j - 2x_2\sin(2\pi x_1 + j\pi/n)\right]^2 \\ f_3 &= \sin(0.5x_1\pi) + \frac{2}{|J_3|}\sum_{j\in J_3}\left[x_j - 2x_2\sin(2\pi x_1 + j\pi/n)\right]^2 \end{aligned} \qquad \text{(С.83)}$$

где множества J_1, J_2 и J_3 определяются как

$$\begin{aligned} J_1 &= \{j \in [3,n] : j-1 \quad \text{кратное } 3\} \\ J_2 &= \{j \in [3,n] : j-2 \quad \text{кратное } 3\} \\ J_3 &= \{j \in [3,n] : j \quad \text{кратное } 3\} \end{aligned} \tag{С.84}$$

Поисковое пространство равно

$$\begin{aligned} x_1 &\in [0,1] \\ x_2 &\in [0,1] \\ x_j &\in [-2,2] \quad \text{для } j \in [3,n] \end{aligned} \tag{С.85}$$

Фронт Парето равен

$$(f_1^*, f_2^*, f_3^*) \quad \text{такие, что} \quad f_1^* \in [0,1], \quad f_2^* \in [0,1], \quad f_3^* \in [0,1], \quad \text{и} \quad (f_1^*)^2 + (f_2^*)^2 + (f_3^*)^2 = 1. \tag{С.86}$$

Множество Парето равно

$$\begin{aligned} x_1^* &\in [0,1] \\ x_2^* &\in [0,1] \\ x_j^* &= 2x_2^* \sin(2\pi x_1^* + j\pi/n) \quad \text{для } j \in [3,n] \end{aligned} \tag{С.87}$$

С.3.9. *Неограниченная многокритериальная оптимизационная задача 9*

Данная трехкритериальная задача определяется в виде:

$$\begin{aligned} f_1 &= 0.5\left[\max\{0, (1+\epsilon)(1-4(2x_1-1)^2)\} + 2x_1\right] x_2 + z_1 \\ f_2 &= 0.5\left[\max\{0, (1+\epsilon)(1-4(2x_1-1)^2)\} - 2x_1 + 2\right] x_2 + z_2 \\ f_3 &= 1 - x_2 + \frac{2}{|J_3|} \sum_{j \in J_3} \left[x_j - 2x_2 \sin(2\pi x_1 + j\pi/n)\right]^2 \end{aligned} \tag{С.88}$$

где ϵ – это положительное вещественное число ($\epsilon = 0.1$ в конкурсе CEC 2009), z_i определяется в виде:

$$z_i = \frac{2}{|J_1|} \sum_{j \in J_1} \left[x_j - 2x_2 \sin(2\pi x_1 + j\pi/n)\right]^2 \quad \text{для } j \in [2,n], \tag{С.88}$$

и множества J_1, J_2 и J_3 такие же, как в неограниченной многокритериальной оптимизационной задаче 8.

Поисковое пространство равно

$$\begin{aligned} x_1 &\in [0,1] \\ x_2 &\in [0,1] \\ x_j &\in [-2,2] \quad \text{для } j \in [3,n] \end{aligned} \tag{С.90}$$

Фронт Pareto имеет два раздела. Первый раздел равен

$$\begin{aligned} f_3^* &\in [0, 1] \\ f_1^* &\in [0, 1] \\ f_2^* &= 1 - f_1^* - f_3^* \end{aligned} \tag{C.91}$$

И второй раздел равен

$$\begin{aligned} f_3^* &\in [0, 1] \\ f_1^* &\in [3(1 - f_3)/4, 1] \\ f_2^* &= 1 - f_1^* - f_3^* \end{aligned} \tag{C.91}$$

Множество Парето равно

$$\begin{aligned} x_1^* &\in [0, 0.25] \cup [0.75, 1] \\ x_2^* &\in [0, 1] \\ x_j^* &= 2x_2 \sin(2\pi x_1 + j\pi/n) \quad \text{для } j \in [3, n] \end{aligned} \tag{C.93}$$

С.3.10. *Неограниченная многокритериальная оптимизационная задача 10*

Данная трехкритериальная задача определяется в виде:

$$\begin{aligned} f_1 &= \cos(0.5x_1\pi)\cos(0.5x_2\pi) + \frac{2}{|J_1|}\sum_{j\in J_1}\left[4y_j^2 - \cos(8\pi y_i) + 1\right] \\ f_2 &= \cos(0.5x_1\pi)\sin(0.5x_2\pi) + \frac{2}{|J_2|}\sum_{j\in J_2}\left[4y_j^2 - \cos(8\pi y_i) + 1\right] \\ f_3 &= \sin(0.5x_1\pi) + \frac{2}{|J_3|}\sum_{j\in J_3}\left[4y_j^2 - \cos(8\pi y_i) + 1\right] \end{aligned} \tag{C.94}$$

где множества J_1, J_2 и J_3 такие же, как и в неограниченной многокритериальной оптимизационной задаче 8, а y_j определяется в виде:

$$y_j = x_j - 2x_2 \sin(2\pi x_1 + j\pi/n) \quad \text{для } j \in [3, n]. \tag{C.95}$$

Поисковое пространство, фронт Парето и множество Парето такие же, как и в неограниченной многокритериальной оптимизационной задаче 8.

С.4. Динамические эталоны

За многие годы исследователи предложили разные динамические эталонные задачи [Branke, 1999]. Несколько ограниченных динамических задач приводится в публикации [Nguyen и Yao, 2009], и еще несколько многокритериальных динамических задач имеется в публикации [Ray и соавт., 2009a]. В публикации [Yang, 2008a] было предложено несколько комбинаторных динамических эталонов, включая динамические задачи о ранце и задачи коммивояжера [Branke и соавт., 2006], [Mavrovouniotis и Yang, 2011]. Тем не менее мы ограничим наше обсуждение непрерывными динамическими эталонами.

В данном разделе обобщаются оптимизационные задачи из публикации [Li и соавт., 2008], содержащей непрерывные эталоны и оценочные метрические показатели, которые использовались для конкурса по динамической оптимизации на Конгрессе IEEE по эволюционным вычислениям 2009 года (CEC 2009). Динамические эталоны основываются на некоторых неограниченных задачах приложения С.1. и включают смещения и матрицы поворота (см. приложение С.7), встраивание варьирующихся со временем функций и суммирование (или «композицию») нескольких таких функций. Мы приводим полное описание динамических эталонов CEC 2009 в разделе С.4.1, а затем предлагаем упрощенный вариант эталонов в разделе С.4.2.

С.4.1. *Полное описание динамического эталона*

Рассмотрим одну из n-мерных функций $f(x)$ раздела С.1. Сначала нормализуем величину эталона. Мы делаем это для обеспечения того, чтобы варьирующаяся со временем функция, которую мы добавим позже, имела желаемый относительный эффект. Мы нормализуем величину эталона, шкалируя ее следующим образом:

$$f'(x) = \frac{Cf(x)}{f_{\max}}, \quad \text{где } C = 2000. \tag{С.96}$$

Константа C выбрана для придания одинаковой величины всем шкалируемым эталонам, чтобы эффект варьирующегося со временем компонента, который мы добавим позже, имел одинаковое влияние на все шкалируемые эталоны.

Теперь обсудим определение $f_{\max}$ в уравнении (С.96). Для динамических эталонов мы обычно используем базовую функцию $f(x)$, которая чаще всего увеличивается вместе с увеличением x. Хотя многие функ-

ции раздела С.1 имеют большое число локальных пиков и впадин, многие функции близки к своему максимуму, когда каждый элемент x находится на своем максимальном значении. Следовательно, величина / max в уравнении (С.96) оценивается в виде:

$$f_{\max} \approx f(x_{\max} Q), \tag{С.97}$$

где Q – это матрица поворота, о которой мы поговорим ниже, а $x_{\max}$ определяется поэлементно:

$$\begin{aligned} x &= [\, x_1 \ldots x_n \,] \quad \text{где } x_i \in [\, x_{i,\min}, x_{i,\max} \,] \\ \Rightarrow \; x_{\max} &= [\, x_{1,\max} \ldots x_{n,\max} \,]. \end{aligned} \tag{С.98}$$

Затем мы сдвигаем $f'(x)$ и получаем $f'(x - \theta)$, где θ – это случайный n-элементный вектор смещения. Каждый элемент вектора смещения равномерно распределен таким образом, что оптимум $f'(x - \theta)$ равномерно распределен на области x. Например, предположим, что в качестве базовой функции мы используем функцию Экли.

Область функции Экли – это $x_i \in [-30, 30]$ для $i \in [1, n]$. Оптимум несмещенной функции Экли находится в $x_i^* = 0$ для $i \in [1, n]$. Следовательно, каждый элемент θ должен быть равномерно распределен на $[-30, 30]$ для всех i. Таким образом, каждый элемент оптимизирующего значения $f'(x - \theta)$ равномерно распределен на $[-30, 30]$, что помогает обеспечить равномерное игровое поле при сравнении разных эволюционных алгоритмов, как описано в приложении С.7.1.

Затем мы поворачиваем шкалированный и сдвинутый эталон и получаем $f'((x - \theta)Q)$, где Q – это случайная ортогональная матрица поворота. Этот шаг является дополнительным, и он обеспечивает равные условия при сравнении разных эволюционных алгоритмов, как описано в приложении С.7.2. Обратите внимание, что Q также используется в уравнении (С.97) для аппроксимирования $f_{\max}$.

Затем мы добавляем варьирующуюся во времени функцию $\phi(t)$ и получаем $f'((x - \theta)Q) + \phi(t)$. Функция $\phi(t)$ может модифицироваться из поколения в поколение следующим образом:

$$\begin{aligned} \phi(t) &\leftarrow \phi(t) - 1) + \Delta\phi \\ \phi(t) &\leftarrow \min(\phi(t), \phi_{\max}) \\ \phi(t) &\leftarrow \max(\phi(t), \phi_{\min}) \end{aligned} \tag{С.99}$$

где t – это номер итерации обновления функции (не обязательно номер поколения эволюционного алгоритма), а $\phi_{\min}$ и $\phi_{\max}$ определяют минимально и максимально допустимые значения $\phi(t)$. Вариация $\Delta\phi$ может

принимать несколько форм. Сначала мы обсудим динамику, о которой говорится в публикациях [Li и соавт., 2008], [Li и Yang, 2008], как малошаговую динамику:

$$\text{малый шаг: } \Delta\phi = \alpha\phi_{\text{диапазон}} r(t-1)\,\phi_s, \tag{С.100}$$

где α – это константа, $\phi_{\text{диапазон}}$ – допустимый диапазон $\phi(t)$, 0S – константа, определяющая серьезность изменения $\phi(t)$, $r(t-1)$ – случайное число, равномерно распределенное на [–1, 1]. В публикации [Li и соавт., 2008] используется

$$\begin{aligned} \alpha &= 0.04 \\ \phi_s &= 5 \\ \phi_{\min} &= 10 \\ \phi_{\max} &= 100 \\ \phi_{\text{диапазон}} &= \phi_{\max} - \phi_{\min} \end{aligned} \tag{С.101}$$

Исходное значение $\phi(t)$ в $t = 0$ является случайным числом, взятым из равномерного распределения между $\phi_{\min}$ и $\phi_{\max}$. Из уравнений (С.99)–(С.101) мы видим, что при малошаговой динамике каждое поколение $\phi(t)$ изменяется не более чем на 18. Из уравнения(С.96) мы видим, что $f'(x) \in [-2000, 2000]$ (приблизительно). Следовательно, максимальное изменение $\phi(t)$ в одном поколении относительно максимального значения $f'(x)$ для малошагового изменения составляет 18/2000 = 0.9 %.

Обратите внимание, что уравнение (С.99) не применяется в каждом поколении; оно применяется лишь изредка. В публикации [Li и соавт., 2008] предлагается реализовывать уравнение (С.99) один раз в 10 000 оцениваний функции и чтобы эволюционный алгоритм в общей сложности выполнял 600 000 оцениваний функции. Вдобавок мы используем матрицу поворота Q для поворота θ (вектора смещения) каждые 10 000 оцениваний функции:

$$\theta(t) \leftarrow \theta(t-1)Q. \tag{С.102}$$

Наконец, мы генерируем m из этих шкалированных, сдвинутых, повернутых, варьирующихся со временем функций и складываем их вместе, чтобы получить динамическую составную функцию:

$$F(x,t) = \sum_{i=1}^{m} w_i\,[f'((x) - \theta_i(t))Q_i) + \phi_i(t)], \tag{С.103}$$

где каждый w_i – это взвешивающее значение, определяемое путем выполнения следующих ниже четырех выражений в указанном порядке:

$$
\begin{aligned}
w_i &\leftarrow \exp\left[-\left(\frac{\sum_{k=1}^{n}(x_k-\theta_{ik}(t))^2}{2n}\right)^{1/2}\right] \\
w_{\max} &\leftarrow \max\{w_i\} \\
w_i &\leftarrow \begin{cases} w_i & \text{если } w_i = w_{\max} \\ w_i(1-w_{\max}) & \text{если } w_i \neq w_{\max} \end{cases} \\
w_i &\leftarrow \frac{w_i}{\sum_{j=1}^{m} w_j}
\end{aligned}
\qquad \text{(C.104)}
$$

для $i \in [1, m]$. Обратите внимание, что $w_i \in [0, 1]$, и по мере удаления x от θ_i (оптимума i-й сдвинутой функции) w_i уменьшается. В публикации [Li и соавт., 2008] используется $m = 10$. Каждый вектор $\theta_i(t)$ в уравнении (C.103) является случайным n-элементным вектором, который поворачивается каждые 10 000 оцениваний функций, и $\theta_{ik}(t)$ в уравнении (C.104) является k-м элементом $\theta_i(t)$. Каждая матрица Q_i – это случайная, но инвариантная по времени $n \times n$-матрица поворота, и каждая функция $\phi_i(t)$ – это случайная скалярная функция, определяемая уравнением (C.100) и обновляемая каждые 10 000 оцениваний функции. Каждая из m функций, которые суммируются в уравнении (C.103), имеет разную варьирующуюся со временем составляющую. Поэтому когда мы складываем эти m функций вместе, то получаем составную функцию, минимизирующее значение которой вполне может меняться из поколения в поколение.

Подытоживая результаты в вышеприведенных пунктах, получаем алгоритм рис. C.23 для генерирования динамической эталонной функции.

На рис. C.23 представлено определение динамических эталонных функций для малошаговой динамики. В публикациях [Li и соавт., 2008] и [Ли и Ян 2008] предлагаеся шесть типов динамики.

1. Малошаговая динамика обобщена выше в уравнениях (C.99)–(C.101).
2. Крупношаговая динамика описывается следующим образом:

$$\text{крупный шаг: } \Delta\phi = \phi_{\text{диапазон}}[\alpha\ \text{sign}(r(t-1)) + (\alpha_{\max} - \alpha)r(t-1)]\phi_s, \qquad \text{(C.105)}$$

где $r(t-1)$ – это случайное число, равномерно распределенное на $[-1, 1]$. Единственной новой константой в приведенном выше

уравнении является α_{max}, которая в публикации [Li и соавт., 2008] принимается равной

$$\alpha_{max} = 0.1. \qquad \text{(С.106)}$$

Из уравнений (С.101), (С.105) и (С.106) мы видим, что при крупношаговой динамике $\phi(t)$ изменяется не более чем на 45 в одном поколении. Из уравнения (С.96) мы видим, что $f'(x) \in [-2000, 2000]$ (приблизительно). Следовательно, максимальное изменение $\phi(t)$ в одном поколении относительно максимального значения $f'(x)$ для крупношагового изменения составляет 45/2000 = 2.25 %.

Begin инициализация
  $f(\cdot)$ = базовая функция из раздела С.1
  $[x_{min}, x_{max}]$ = n-мерная поисковая область
  x^* = n-мерное оптимизирующее значение функции $f(x)$
  E_{update} = число оцениваний функции между динамическими обновлениями
    (как правило, E_{update} = 10,000)
  m = число функций, объединяемых в эталоне (как правило, m = 10)
  For i = 1 **to** m
    Сгенерировать случайную матрицу поворота Q_i (см. раздел С.7.2)
    Сгенерировать случайный вектор смещения θ_i – такой, что $x^* + \theta_i \in [x_{min}, x_{max}]$
  Next i
  $f_{max} = f(x_{max}Q)$
  C = 2000
  Определение функции: $f'(x) - Cf(x)/f_{max}$
  $\phi(0) \leftarrow U[\phi_{min}, \phi_{max}]$
  $E \leftarrow 0$ = число оцениваний функции
End инициализация
When мы готовы выполнить оценивание эталонной функции
    для кандидатного решения x
  Применить уравнение (С.104) для расчета w_i для $i \in [1, m]$
  $E \leftarrow E + 1$
  If (E mod E_{update}) = 0 **then**
    Применить уравнения (С.99)–(С.101) для обновления $\phi_i(t)$ для $i \in [1, m]$
    Применить уравнение (С.102) для обновления $\theta_i(t)$ для $i \in [1, m]$
  End if
  Применить уравнение (С.103) для оценивания кандидатного решения x
Next оценивание эталона

Рис. С.23. Определение функции для n-мерной динамической функции на основе стандартного эталона $f(\cdot)$ с малошаговой динамикой. (E mod E_{update}) – это остаток после целочисленного деления E / E_{update}

3. Случайная динамика описывается следующим образом:

$$\text{случайная:}\quad \Delta\phi = \phi_s \rho(t-1), \tag{C.107}$$

где $\rho(t-1)$ – это случайное число, взятое из гауссова распределения с нулевым средним и единичной дисперсией. Поскольку гауссово случайное число не ограничено, $\phi(t)$ может изменяться от минимального к максимальному значению (или наоборот) за одно поколение. Однако в 99.7 % случаев изменение $\phi(t)$ будет происходить в пределах 3σ, что равняется $3\phi_s = 15$. При случайной динамике 3σ изменение в $\phi(t)$ за одно поколение относительно максимального значения $f'(x)$ составляет 15/2000 = 0.75 %.

4. Хаотическая динамика описывается следующим образом:

$$\text{хаотическая:}\ \phi(t) = [A\phi(t-1) - \phi_{\min}]\left[1 - \frac{\phi(t-1) - \phi_{\min}}{\phi_{\text{диапазон}}}\right], \tag{C.108}$$

где $\phi(t-1)$ – это случайное число, равномерно распределенное на [–1, 1]. Единственной новой константой в приведенном выше уравнении является A, которая в публикации [Li и соавт., 2008] определяется как

$$A = 3.67. \tag{C.109}$$

5. Рекуррентная динамика описывается следующим образом:

$$\text{рекуррентная:}\ \phi(t) = \phi_{\min} + \frac{\phi_{\text{диапазон}}\,[\sin(2\pi(t-1)/P + \varsigma) + 1]}{2} + \rho_s \rho(t-1). \tag{C.110}$$

Это единственная детерминированная динамика, определенная в публикации [Li и соавт., 2008]. Единственными новыми константами в приведенном выше уравнении являются P (период) и ς (начальная фаза), которые в публикации [Li и соавт., 2008] определяются как

$$\begin{aligned} P &= 12E_{\text{update}} \\ \varsigma &= U[0, 2\pi] \end{aligned} \tag{C.111}$$

где E_{update} – это число оцениваний функции между динамическими обновлениями и $U[0, 2\pi]$ – случайное число, равномерно распределенное между 0 и 2π.

6. Шумная рекуррентная динамика описывается следующим образом:

$$\text{шумная рекуррентная: } \phi(t) = \phi_{\min} + \frac{\phi_{\text{диапазон}}\,[\sin(2\pi(t-1)/P + \varsigma) + 1]}{2} + \rho_s \rho(t-1), \tag{C.112}$$

где $\rho(t-1)$ – это случайное число, взятое из гауссова распределения с нулевым средним и единичной дисперсией. Единственной новой константой в приведенном выше уравнении является ρ_s, серьезность шумовой динамики, которая в пуликации [Li et al. 2008] определяется как

$$\rho_s = 0.8. \tag{C.113}$$

Мы можем модифицировать рис. С.23 и реализовать любой из вышеперечисленных типов динамики. Для того чтобы изменить тип динамики, нам нужно изменить только одну строку на рис. С.23. Мы модифицируем строку, которая говорит «Применить уравнения (С.99)–(С.101) для обновления $\phi_i(t)$ для $i \in [1, m]$».

1. Если нам нужна малошаговая динамика, то мы реализуем рис. С.23, как написано.
2. Если нам нужна крупношаговая динамика, то для обновления $\phi_i(t)$ мы используем уравнение (С.105).
3. Если нам нужна случайная динамика, то для обновления $\phi_i(t)$ мы используем уравнение (С.107).
4. Если нам нужна хаотическая динамика, то для обновления $\phi_i(t)$ мы используем уравнение (С.108).
5. Если нам нужна рекуррентная динамика, то для обновления $\phi_i(t)$ мы используем уравнение (С.110).
6. Если нам нужна шумная рекуррентная динамика, то для обновления $\phi_i(t)$ мы используем уравнение (С.112).

В публикации [Li и соавт., 2008] для использования в качестве базисных функций $f(\cdot)$ на рис. С.23 предлагается пять разных функций: сферическая функция (раздел С.1.1), функция Растригина (раздел С.1.11), функция Вейерштрасса (см. раздел С.1.22), функция Гриванка (раздел С.1.6) и функция Экли (см. раздел С.1.2). Обратите внимание, что каждая из этих функций в оригинальной версии имеет оптимизирующее решение $x^* = 0$.

С.4.2. *Упрощенное описание динамического эталона*

На рис. С.23 показано, что в эталонных функциях существует несколько взаимодействующих динамик, включая веса $\{w_i\}$, которые сами являются функциями кандидатного решения x, динамические переменные $\phi_i(t)$ и динамические переменные смещения $\theta_i(t)$. Однако, по всей видимости, суть динамики можно уловить с помощью переменных смещения; другие переменные обеспечивают только эффекты второго порядка. Кроме того, нет необходимости складывать несколько функций вместе, для того чтобы получить функцию с большой динамикой; другими словами, мы можем использовать $m = 1$ на рис. С.23 и по-прежнему получать хорошие динамические эталоны. В результате получается рис. С.24, который представляет собой простой, но эффективный генератор динамических эталонных функций.

Begin инициализация
  $f(\cdot)$ = базовая функция из раздела С.1
  $[x_{min}, x_{max}]$ = n-мерная поисковая область
  x^* = n-мерное оптимизирующее значение функции $f(x)$
  E_{update} = число оцениваний функции между динамическими обновлениями
  Сгенерировать случайную матрицу поворота Q (см. раздел С.7.2)
  Сгенерировать случайное смещение θ – такое, что $x^* + \theta \in [x_{min}, x_{max}]$
End инициализация
When мы готовы выполнить оценивание эталонной функции
      для кандидатного решения x
  $E \leftarrow E + 1$
  If (E mod E_{update}) = 0 **then**
    Применить уравнение (С.102) для обновления $\theta(t)$
  End if
  Применить $F(x, t) = f((x - \theta(t))Q)$ для оценивания кандидатного решения x
Next оценивание эталона

Рис. С.24. Упрощенное определение функции для n-мерной динамической функции на основе стандартного эталона $f(\cdot)$

Наконец, мы упомянем, что для обновления смещения $\theta(t)$ мы можем использовать методы, отличные от уравнения (С.102). Уравнение (С.102) состоит из поворота $\theta(t)$ вокруг начала поискового пространства. Однако есть много других разумных способов обновить $\theta(t)$. Мы можем изменять $\theta(t)$ каким-то другим предсказуемым образом (к примеру, линейно или периодически), либо мы можем генерировать случайные $\theta(t)$ при каждом динамическом изменении. Для изменения $\theta(t)$ мы можем

использовать разные методы, которые будут представлять динамику в конкретных реальных задачах.

С.5. Шумные эталоны

Шумные эталонные задачи создаются легко. Мы просто берем стандартную, нешумную эталонную функцию и добавляем шум. Мы можем добавлять разные типы шумов: шум со статистическими показателями, которые не зависят от заданного кандидатного решения x, как показано в уравнении (21.39); шум со статистическими показателями, которые так или иначе варьируются вместе с x, как показано в уравнении (21.42), гауссов шум, равномерный шум либо любой другой тип шума, который мы хотим использовать с нашим эволюционным алгоритмом.

С.6. Задачи коммивояжера

Веб-сайт TSPLIB имеет коллекцию, состоящую из более чем 100 эталонов задачи коммивояжера [Reinelt, 2008]. Простейшей задачей коммивояжера в коллекции является эталон Ulysses16, основанный на 16 городах, которые легендарный греческий король Улисс посетил в Средиземноморье во время своих путешествий [Grötschel и Padberg, 2001]. Самой большой задачей коммивояжера на веб-сайте является задача для программируемой логической матрицы с 85 900 вершинами. Центр дискретной математики и теоретических компьютерных наук (The Center for Discrete Mathematics & Theoretical Computer Science) поддерживает веб-сайт с крупномасштабными эталонами для задачи коммивояжера, самый большой из которых содержит более 20 миллионов вершин [Demetrescu, 2012].

Каждая задча коммивояжера определяется файлом с расширением `TSP` – например, `ULYSSES.TSP`. Файл TSP содержит координаты широты и долготы каждого города в формате `DDD.MM`, где `DDD` обозначет градусы, а `MM` – минуты. Файл TSP также определяет «тип веса ребра» задачи, `EUC_2D` либо `GEO`, который указывает на то, как рассчитывать расстояния между городами.

Для задач `EUC_2D` необходимо вычислять евклидово расстояние $D(i, k)$ между городами i и k следующим образом:

$$\begin{aligned} \Delta B &= B_i - B_k \\ \Delta L &= L_i - L_k \\ D(i,k) &= \text{round}\ \sqrt{\Delta B^2 + \Delta L^2} \end{aligned} \qquad \text{(С.114)}$$

где B_i и L – это широта и долгота города i, и функция `round` округляет до ближайшего целого числа. Округление не является строго необходимым, но для задач TSPLIB оно выполняется традиционно, и поэтому мы можем использовать округление для справедливых сравнений между разными алгоритмами задачи коммивояжера и ранее опубликованными результатами.

Для задач `GED` нам нужно вычислить географическое расстояние между городами i и k, исходя из допущения, что Земля является идеальной сферой:

$$\begin{aligned} g_1 &= \cos(L_i - L_k) \\ g_2 &= \cos(B_i - B_k) \\ g_3 &= \cos(B_i + B_k) \\ D(i,k) &= \left\lfloor R \arccos\left\{[(1+q_1)q_2 - (1-q_1)q_3]/2\right\} + 1 \right\rfloor \end{aligned} \tag{C.115}$$

где R = 6378,388 км – это радиус Земли, функция `floor` $\lfloor \cdot \rfloor$ возвращает наибольшее целое число, меньшее или равное ее аргументу. Функция `floor` и прибавление 1 в конце вычисления $D(i, k)$ не являются необходимыми в общем случае, но используются при вычислении стандартных географических расстояний в эталонах TSPLIB для округления вверх до ближайшего целого.

Деривация географического расстояния

Мы можем вывести уравнение (C.115), сначала конвертировав широту и долготу в прямоугольную систему координат (декартовы координаты), получив координаты города i в виде:

$$\begin{aligned} x_i &= R \cos B_i \cos L_i \\ y_i &= R \cos B_i \sin L_i \\ z_i &= R \sin B_i \end{aligned} \tag{C.116}$$

Мы получаем аналогичные уравнения для прямоугольных координат x_k, y_k и z_k k-го города. Теперь вспомним, что мы можем написать скалярное произведение между двумя векторами A и B в виде:

$$A \cdot B = |A| \cdot |B| \cos\theta, \tag{C.117}$$

где θ – это угол между векторами. Следовательно, мы можем записать скалярное произведение между векторами, которое определяет город i и город k как

$$\begin{bmatrix} R\cos B_i \cos L_i \\ R\cos B_i \sin L_i \\ R\sin B_i \end{bmatrix} \cdot \begin{bmatrix} R\cos B_k \cos L_k \\ R\cos B_k \sin L_k \\ R\sin B_k \end{bmatrix} = R^2 \cos\theta, \tag{C.118}$$

где θ – это угол между городами i и k. Деление обеих сторон на R^2 и разложение приведенного выше уравнения дает

$$\cos B_i \cos L_i \cos B_k \cos L_k + \cos B_i \sin L_i \cos B_k \sin L_k + \sin B_i \sin B_k = \cos\theta. \tag{C.119}$$

Этот результат можно упростить до

$$\cos B_i \cos B_k (\cos L_i \cos L_k + \sin L_i \sin L_k) + \sin B_i \sin B_k = \cos\theta. \tag{C.120}$$

Мы можем использовать стандартные тригонометрические тождества для записи приведенного выше уравнения в виде:

$$\tfrac{1}{2}\left[\cos(B_i + B_k) + \cos(B_i - B_k)\right] \cos(L_i - L_k) + \tfrac{1}{2}\left[\cos(B_i - B_k) - \cos(B_i + B_k)\right] = \cos\theta. \tag{C.121}$$

Решение для θ дает

$$\theta = \arccos\left\{\tfrac{1}{2}\left[q_1(q_2 + q_3) + q_2 - q_3\right]\right\}. \tag{C.122}$$

Две точки на сфере радиуса R, разделенные углом θ, имеют между собой расстояние на поверхности сферы $R\theta$, в результате получаем уравнение (C.115)[7].

Другие метрические показатели расстояния

Другие метрические показатели можно найти в публикации [Reinelt, 2008], в том числе трехмерное евклидово расстояние, манхэттенское расстояние, которое исходит из того, что маршрут проходит по дорогам, проложенным на ортогональной решетке, максимальное расстояние, измеряющее расстояние вдоль координаты, которая требует расстояния самой дальней поездки, псевдоевклидово расстояние, то есть такое же, как уравнение (C.114), за исключением того, что вместо округления до ближайшего целого числа мы округляем до следующего наибольшего целого числа, и, наконец, специальная функция расстояния, которая имеет отношение к рентгеновской кристаллографии.

[7] Данный расчет основан на веб-сайте Джаспера Спаана (Jasper Spaans) по адресу http://jsp.vsl9.net/lr/spheredistance.php.

Другие комбинаторные задачи

Как симметричные, так и асимметричные задачи коммивояжера можно найти в упомянутой публикации [Reinelt, 2008]. Веб-сайт также содержит родственные типы задач, включая следующие.

1. Задача последовательного упорядочения является асимметричной задачей коммивояжера, которая имеет ограничения по приоритетности [Dorigo и Stützle, 2004], то есть при наличии n городов найти маршрут, который в результате дает кратчайшее расстояние, при этом требуется, чтобы город i_m был посещен перед городом k_m для $M \in [1, M]$, где M – это число ограничений.
2. Задача ограниченной по емкости маршрутизации транспортных средств включает $(n - 1)$ вершин и один склад [Toth и Vigo, 2002]. Задача состоит в том, чтобы использовать грузовики для осуществления необходимых поставок со склада в вершины, исходя из допущения, что каждая вершина имеет определенный спрос на доставку и что все грузовики имеют одинаковые емкости. Каждый маршрут начинается на складе, производит поставки в определенное число вершин, а затем возвращается на склад. Стоимостная функция может представлять собой общее расстояние, пройденное всеми грузовиками, либо общее время, потребовавшееся для доставки.
3. Задача гамильтонова пути – это задача обнаружения пути, который проходит через каждую вершину графа ровно один раз [Balakrishnan, 1997]. Задача гамильтонова цикла включает дополнительное требование о том, что путь возвращается к своей исходной точке. На рис. С.25 показан пример двух задач гамильтонова пути. Связный граф слева имеет гамильтонов путь: путь $1 \rightarrow 3 \rightarrow 2 \rightarrow 5 \rightarrow 4$ является решением. Однако граф справа не имеет гамильтонова пути. На рисунке справа мы можем найти путь, который проходит через все вершины один раз, но этот путь также будет проходить через некоторые вершины более одного раза (например, $5 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 3 \rightarrow 4$).

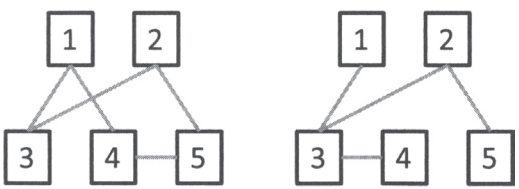


Рис. С.25. Два связных графа. У графа слева есть гамильтонов путь, а у графа справа нет

С.7. Устранение смещения в поисковом пространстве

В данном разделе мы вернемся к обсуждению непрерывных оптимизационных задач. Некоторые эволюционные алгоритмы естественным образом хорошо работают на определенных эталонах просто из-за случайного совпадения характерных особенностей эталонов с характерными особенностями эволюционных алгоритмов. Это не говорит о результативности эволюционных алгоритмов, а лишь указывает на то, что признаки многих искусственных эталонов не отражают реальных задач. В данном разделе рассматривается использование сдвигов и матриц поворота в оптимизационных эталонах, которые позволяют делать их гораздо сложнее и реалистичнее.

С.7.1. *Сдвиги*

Отдельные эволюционные алгоритмы естественным образом смещаются в сторону определенных типов поисковых пространств. Например, в разделе 16.2 мы увидели, что некоторые типы оппозиционного самообучения (OBL) имеют тенденцию сдвигать кандидатные решения ближе к центру поисковой области. Поэтому алгоритм оппозиционного самообучения естественным образом работает хорошо на задачах, решение которых находится возле центра поискового пространства. Однако такая хорошая результативность алгоритма OBL вводит в заблуждение; это искусственный побочный эффект от того факта, что многие эталоны имеют решения вблизи центра поискового пространства. Еще одним примером является дифференциальная эволюция (DE), которая модифицирует кандидатные решения на основе разностных векторов. Поэтому алгоритм DE будет хорошо работать на ограниченных задачах, допустимые участки которых параллельны друг другу. Опять же, такая хорошая результативность алгоритма DE может вводить в заблуждение; это искусственный побочный эффект от того факта, что многие эталоны имеют параллельные допустимые участки. Многие эталоны в этом приложении имеют решения, которые находятся строго в центре поискового пространства. Будет совершенно несправедливым использовать такие эталоны для оценивания результативности эволюционных алгоритмов.

Иногда, хотя и не всегда, мы можем непосредственно увидеть, что алгоритм смещен и легче находит точку решения, когда он находится в центре поискового пространства [Clerc, 2012b]. Например, мы можем

выполнить наш алгоритм на двумерной задаче с поисковым пространством $x_1 \in [-1, +1]$ и $x_2 \in [-1, +1]$ и со стоимостной функцией $f(x) = 1$ для всех x и построить график популяции в двумерной поисковой области после большого числа поколений. Если алгоритм является несмещенным, то распределение будет равномерным. Однако в случае многих алгоритмов распределение будет более плотным вокруг точки $x = (0, 0)$. В этом случае можно сделать вывод, что алгоритм смещен. Правда, разница в плотности может быть не видна, и поэтому единственный безопасный способ оценивания оптимизационных алгоритмов – никогда не использовать стоимостные функции, решение которых находится в центре поисковой области (или, по сути дела, даже на диагонали).

Мы можем модифицировать смещенные эталоны и сделать их несмещенными, добавив в независимые переменные задачи сдвиги [Liang и соавт., 2005], [Suganthan и соавт., 2005]. Рассмотрим сферическую функцию уравнения (С.2), повторенную здесь:

$$f(x) = \sum_{i=1}^{n} x_i^2 \qquad \text{(С.123)}$$

с оптимумом $x^* = 0$. Если поисковое пространство равно $x_i \in [-C, C]$ для всех i, то оптимум находится в центре поискового пространства. Поэтому мы модифицируем уравнение (С.123) следующим образом:

$$f(x) = \sum_{i=1}^{n} (x_i - o_i)^2, \quad \text{где } o_i \sim U[-C, C] \quad \text{для } i \in [1, n]. \qquad \text{(С.124)}$$

Для генерирования o_i мы можем также использовать распределение, отличное от равномерного распределения, но мы обычно ограничиваем o_i областью $[-C, C]$, для того чтобы гарантировать, что глобальный оптимум уравнения (С.124) лежит в поисковом пространстве. Сдвинутая сферическая функция уравнения (С.124) имеет ту же форму, что и исходная сферическая функция, но ее решение находится в случайной точке поискового пространства. Это помогает гарантировать, что ни один конкретный эволюционный алгоритм не будет иметь несправедливого преимущества при оценивании эталонов. На рис. С.26 показан сдвинутый вариант сферической функции.

При сравнении эволюционных алгоритмов на сдвинутой сферической функции мы должны выполнить несколько симуляций Монте-Карло, каждую с разным сдвигом. Такой подход, описанный на рис. С.27, позволяет определять лучший эволюционный алгоритм для сферических функций, избегая при этом смещения, обусловленного располо-

жением оптимума. После завершения цикла на рис. С.27 мы получим M результатов для первого эволюционного алгоритма, M результатов для второго и т. д. Мы можем сравнить результативности эволюционных алгоритмов, взяв среднее значение каждого множества из M результатов либо лучшую или худшую (или любую другую) размерность, в зависимости от того, какое число представляет наибольший интерес (см. приложение В.2).

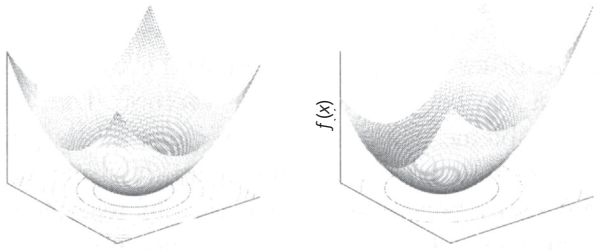


Рис. С.26. На рисунке слева показана оригинальная, несдвинутая двумерная сферическая функция. На рисунке справа показана та же самая функция, сдвинутая вдоль обеих независимых переменных

P = число эволюционных алгоритмов для оценивания
M = число симуляций Монте-Карло
For $j = 1$ **to** M
    Сгенерировать случайный o_i для $i \in [1, n]$
    For $p = 1$ **to** P
        Оценить результативность p-го эволюционного алгоритма на $f(x - o)$
    Next эволюционный алгоритм
Next симуляция Монте-Карло

Рис. С.27. Схематичное описание симуляции Монте-Карло для оценивания результативности эволюционных алгоритмов на n-мерных сдвинутых задачах. Смещение, которое связано с расположением оптимума, было удалено

С.7.2. *Матрицы поворота*

Поисковый процесс некоторых эволюционных алгоритмов естественным образом смещен в сторону поиска вдоль одной независимой переменной. Например, стратегия мутации или восхождения на вершину холма, изменяющая одну независимую переменную за раз, выполняет поиск вдоль одной размерности задачи на каждой итерации. Эти типы оптимизационных алгоритмов хорошо работают на задачах, градиенты которых параллельны независимым переменным. Одна-

ко такая хорошая результативность может вводить в заблуждение; это искусственный побочный эффект от того факта, что многие эталоны имеют градиенты, параллельные единичным векторам поискового пространства. Многие эталоны в настоящем приложении имеют такие градиенты. Будет несправедливым использовать такие эталоны для оценивания результативности эволюционных алгоритмов.

По этой причине важно модифицировать эталоны путем встраивания матриц поворота в задачу [Salomon, 1996], [Suganthan и соавт., 2005]. Рассмотрим функцию max Швефеля уравнения (С.15), повторенную здесь для удобства:

$$f(x) = \max_i(|x_i| : i \in \{1, ..., n\}), \qquad \text{(С.125)}$$

где наш целевой критерий состоит в том, чтобы минимизировать $f(x)$. Поисковый процесс, который просто уменьшает один элемент x за один раз, будет выполняться на этой задаче довольно хорошо. Такой простой поисковый процесс также вполне может хорошо работать на реальных задачах, однако существует ряд реальных задач, которые требуют более сложной поисковой стратегии. Поэтому мы модифицируем уравнение (С.125) следующим образом:

$$f(x) = \max_i(|y_i| : i \in \{1, ..., n\}), \quad \text{где } y = xQ, \qquad \text{(С.126)}$$

где n-элементные векторы x и y – это векторы-строки и Q – $n \times n$-матрица поворота. Матрица поворота представляет собой матрицу, которая при умножении на вектор вращает этот вектор в своей n-мерной области [Golan, 2007]. Матрица поворота эквивалентна ортогональной матрице, а ортогональная матрица определяется как матрица, транспонирование которой равно определителю и определитель которой равен 1:

$$Q^{-1} = Q^T \quad \text{и} \quad Q = 1. \qquad \text{(С.127)}$$

Случайная матрица поворота может быть сгенерирована с помощью QR-разложения [Golan, 2007]. QR-разложение влечет за собой нахождение ортогональной матрицы Q и верхней треугольной матрицы R – таких, что $QR = D$ для заданной матрицы D. Любая вещественная квадратная матрица D имеет QR-разложение. Если сгенерировать $n \times n$-матрицу D со случайными записями и найти ее QR-разложение, то Q-матрица будет равна матрице случайного поворота. Следовательно, мы можем сгенерировать случайную $n \times n$-матрицу поворота Q в среде MATLAB следующим образом:

```
D = randn(n);
[Q, R] = QR(D);
```

где `randn(n)` – функция языка MATLAB, генерирующая $n \times n$-матрицу, в которой все записи взяты из гауссова распределения с нулевым средним и единичной дисперсией, а `QR` – функция языка MATLAB *QR*-разложения. Другие линейно-алгебраические библиотеки и программные пакеты имеют аналогичные функции. Сдвинутая функция max Швефеля уравнения (С.126) имеет ту же форму, что и исходная функция max Швефеля, но только повернута относительно начала поискового пространства. Следовательно, градиент целевой функции больше не параллелен размерностям независимых переменных. Это помогает обеспечить, чтобы ни один конкретный эволюционный алгоритм не имел несправедливого преимущества при оценивании эталонов. На рис. С.28 показана функция max Швефеля, повернутая на несколько градусов против часовой стрелки (глядя вниз на графике).

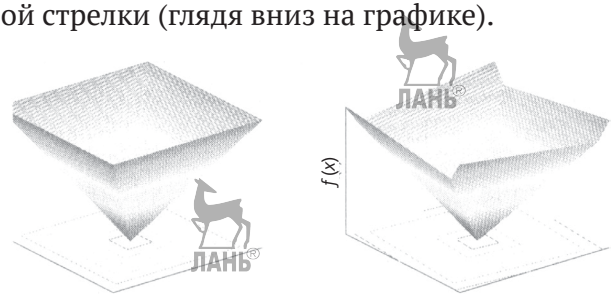


Рис. С.28. Слева изображена двумерная функция max Швефеля. На рисунке справа та же самая функция повернута на несколько градусов против часовой стрелки

При сравнении эволюционных алгоритмов на повернутой функции max Швефеля мы должны выполнить несколько симуляций Монте-Карло, каждую с другой матрицей поворота. Этот подход аналогичен подходу, показанному на рис. С.27, и схематично описан на рис. С.29. Такой подход позволяет определить лучшую результативность среди эволюционных алгоритмов, избегая при этом смещения, которое может присутствовать из-за параллельного характера градиента исходной функции. После завершения цикла на рис. С.29 мы получим M результатов для первого эволюционного алгоритма, M результатов для второго и т. д. Мы можем сравнить результативности эволюционных алгоритмов, взяв среднее значение каждого множества из M результатов, либо взяв лучшую или худшую (или любую другую) размерность, в зависимости от того, какое число представляет наибольший интерес (см. приложение В.2). Мы можем объединить логику рис. С.27 и С.29,

с тем чтобы получить эталонные сравнения на сдвинутых и повернутых функциях $f((x - o)Q)$.

P = число эволюционных алгоритмов для оценивания
M = число симуляций Монте-Карло
For *j* = 1 **to** *M*
    Сгенерировать случайную матрицу поворота *Q*
    For *p* – 1 **to** *P*
        Оценить результативность *p*-го эволюционного алгоритма на *f*(*xQ*)
    Next эволюционный алгоритм
Next симуляция Монте-Карло

Рис. С.29. Схематичное описание симуляции Монте-Карло для оценивания результативности эволюционных алгоритмов на повернутых задачах. Смещение, вызванное параллельным выравниванием градиента с системой координат, было удалено

Список литературы

1. *Aarts, E. and Korst, J.* (1989). Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing. John Wiley & Sons.
2. *Aarts, E., Lenstra, J., and van Laarhoven, P.* (2003). Simulated annealing. In Aarts, E. and Lenstra, J., editors, Local Search in Combinatorial Optimization, pages 91–120. Princeton University Press.
3. *Ackley, D.* (1987a). A Connectionist Machine for Genetic Hillclimbing. Kluwer Academic Publishers.
4. *Ackley, D.* (1987b). An empirical study of bit vector function optimization. In Davis, L., editor, Genetic Algorithms and Simulated Annealing, pages 170–215. Pitman Publishing.
5. *Adami, C.* (1997). Introduction to Artificial Life. Springer.
6. *Adler, F. and Nuernberger, B.* (1994). Persistence in patchy irregular landscapes. Theoretical Population Biology, 45(1): 41–75.
7. *Aguirre, A., Rionda, S., Coello Coello, C., Lizarraga, G., and Mezura-Montes, E.* (2004). Handling constraints using multiobjective optimization concepts. International Journal for Numerical Methods in Engineering, 59(15): 1989–2017.
8. *Ahn, C. and Ramakrishna, R.* (2007). Multiobjective real-coded Bayesian optimization algorithm revisited: Diversity preservation. Genetic and Evolutionary Computation Conference, London, England, pages 593–600.
9. *Akat, S. and Gazi, V.* (2008). Particle swarm optimization with dynamic neighborhood topology: Three neighborhood strategies and preliminary results. IEEE Swarm Intelligence Symposium, St. Louis, Missouri, pages 1–8.
10. *Alami, J. and El lmrani, A.* (2008). Using cultural algorithm for the fixed-spectrum frequency assignment problem. Journal of Mobile Communication, 2(1): 1–9.
11. *Alami, J., El Imrani, A., and Bouroumi, A.* (2007). A multipopulation cultural algorithm using fuzzy clustering. Applied Soft Computing, 7(2): 506–519.

12. *Alexander, R*. (1996). Optima for Animals. Princeton University Press.
13. *Ali, M., Khompatraporn, C., and Zabinsky, Z*. (2005). A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. Journal of Global Optimization, 31(4): 635–672.
14. *Allenson, R*. (1992). Genetic algorithms with gender for multi–function optimisation. Technical report, Edinburgh Parallel Computing Centre. EPCC–SS92–0l.
15. *Altenberg, L*. (1994). Emergent phenomena in genetic programming. Conference on Evolutionary Programming, San Diego, California, pages 233–241.
16. *Anderson, M. and Oates, T*. (2007). A review of recent research in metareasoning and metalearning. AI Magazine, 28(1): 7–16.
17. *Andre, D., Bennett, F., and Koza, J*. (1996). Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification roblem. Genetic Programming Conference, Palo Alto, California, pages 28–31.
18. *Angeline, P*. (1996a). An investigation into the sensitivity of genetic programming to the frequency of leaf selection during subtree crossover. Genetic Programming Conference, Palo Alto, California, pages 21–29.
19. *Angeline, P*. (1996b). Two self–adaptive crossover operators for genetic programming. In Angeline, P. and Kinnear, K., editors, Advances in Genetic Programming: Volume 2, pages 89–110. The MIT Press.
20. *Angeline, P*. (1997). Subtree crossover: Building block engine or macromutation? Genetic rogramming Conference, Palo Alto, California, pages 9–17.
21. *Applegate, D., Bixby, R., Chvatal, V., and Cook, W*. (2007). The Traveling Salesman Problem. Princeton University Press.
22. *Araujo, M., Wanner, E., Guimaraes, F., and Takahashi, R*. (2009). Constrained optimization based on quadratic approximations in genetic algorithms. In Mezura-Montes, E., editor, Constraint-Handling in Evolutionary Optimization, pages 193–217. Springer.
23. *Arnold, D*. (2002). Noisy Optimization with Evolution Strategies. Kluwer Academic Publishers.
24. *Ashlock, D*. (2009). Evolutionary Computation for Modeling and Optimization. Springer.

25. *Äström, K. and Wittenmark, B*. (2008). Adaptive Control. Dover Publications.

26. *Atashpaz-Gargari, E. and Lucas, C*. (2007). Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition. IEEE Congress on Evolutionary Computation, Singapore, pages 4661–4667.

27. *Auger, A., Bader, J., Brockhoff, D., and Zitzler, E*. (2012). Hypervolume-based multiobjective optimization: Theoretical foundations and practical implications. Theoretical Computer Science, 425:75–103.

28. *Axelrod, R*. (1997). The dissemination of culture: A model with local convergence and global polarization. Journal of Conflict Resolution, 41(2): 203–226.

29. *Axelrod, R*. (2006). The Evolution of Cooperation: Revised Edition. Basic Books. First published in 1984.

30. *Bäck, T*. (1996). Evolutionary Algorithms in Theory and Practice. Oxford University Press.

31. *Bäck, T., Fogel, D., and Michalewicz, Z*. (1997a). Handbook of Evolutionary Computation. Taylor and Francis.

32. *Bäck, T., Hammel, U., and Schwefel, H*. (1997b). Evolutionary computation: Comments on the history and current state. IEEE Transactions on Evolutionary Computation, 1(1):3–17.

33. *Bäck, T. and Schwefel, H.-P*. (1993). An overview of evolutionary algorithms for parameter optimization. Evolutionary Computation, 1(1): 1–23.

34. *Baker, J*. (1987). Reducing bias and inefficiency in the selection algorithm. International Conference on Genetic Algorithms and Their Application, Cambridge, Massachusetts, pages 14–21.

35. *Balakrishnan, V*. (1997). Schaum's Outline of Graph Theory. McGraw-Hill, 13th edition.

36. *Balasubramaniam, P. and Kumar, A*. (2009). Solution of matrix Riccati differential equation for nonlinear singular system using genetic programming. Genetic Programming and Evolvable Machines, 10(1): 71–89.

37. *Ball, W. and Coxeter, H*. (2010). Mathematical Recreations and Essays. Dover, 13th edition.

38. *Baluja, S*. (1994). Population-based incremental learning. Technical report, Carnegie Mellon University. CMU–CS–94–163.

39. *Baluja, S. and Caruana, R*. (1995). Removing the genetics from the standard genetic algorithm. 12th International Conference on Machine Learning, Tahoe City, California, pages 38–46.
40. *Baluja, S. and Davies, S*. (1998). Fast probabilistic modeling for combinatorial optimization. Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence, pages 469–4 76.
41. *Bandyopadhyay, S., Saha, S., Maulik, U., and Deb, K*. (2008). A simulated annealingbased multiobjective optimization algorithm: AMOSA. IEEE Transactions on Evolutionary Computation, 12(3): 269–283.
42. *Banks, A., Vincent, J., and Anyakoha, C*. (2007). A review of particle swarm optimization. Part I: Background and development. Natural Computing, 6(4): 467–484.
43. *Banks, A., Vincent, J., and Anyakoha, C*. (2008). A review of particle swarm optimization. Part II: Hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications. Natural Computing, 7(1): 09–124.
44. *Bankston, J*. (2005). Gregor Mendel and the Discovery of the Gene. Mitchell Lane Publishers.
45. *Banzhaf, W*. (1990). The «molecular» traveling salesman. Biological Cybernetics, 64(1): 7–14.
46. *Banzhaf, W., Nordin, P., Keller, R., and Francone, F*. (1998). Genetic Programming. Morgan Kauffman Publishers.
47. *Barr, R., Golden, B., Kelly, J., Resende, M., and Stewart, W*. Designing and reporting on computational experiments with heuristic methods. Journal of Metaheuristics, 1(1).
48. *Barricelli, N*. (1954). Esempi numerici di processi di evoluzione. Methodos, 6: 45–68. The English translation of the title is Numerical models of evolutionary processes.
49. *Basturk, B. and Karaboga, D*. (2006). An artificial bee colony (ABC) algorithm for numeric function optimization. IEEE Swarm Intelligence Symposium, Indianapolis, Indiana.
50. *Becerra, R. and Coello Coello, C*. (2004). A cultural algorithm with differential evolution to solve constrained optimization problems. In *Lemaitre, C., Reyes, C., and González, J., editors*, Advances in Artificial Intelligence – IBERAMIA 2004: 9th Ibero–American Conference on AI, Puebla, Mexico, November 22–26, 2004, pages 881–890. Springer.

51. *Bellman, R*. (1961). Adaptive Control Processes: A Guided Tour. Princeton University Press.
52. *Benatchba, K., Admane, L., and Koudil, M*. (2005). Using bees to solve a data-mining problem expressed as a max-sat one. In *Mira, J. and Alvarez, J., editors*, Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach, pages 212–220. Springer.
53. *Bernstein, D*. (2006). Optimization r us. IEEE Control Systems Magazine, 26(5): 6–7.
54. *Betts, J*. (2009). Practical Methods for Optimal Control and Estimation Using Nonlinear Programming. Society for Industrial & Applied Mathematics, 2nd edition.
55. *Beveridge, W*. (2004). The Art of Scientific Investigation. Blackburn Press.
56. *Beyer, H.–G*. (1998). On the dynamics of EAs without selection. Foundations of Genetic Algorithms, Amsterdam, The Netherlands, pages 5–26.
57. *Beyer, H.-G*. (2010). The Theory of Evolution Strategies. Springer.
58. *Beyer, H.-G. and Deb, K*. (2001). On self-adaptive features in real-parameter evolutionary algorithms. IEEE Transactions on Evolutionary Computation, 5(3): 250–269.
59. *Beyer, H.-G. and Schwefel, H.-P*. (2002). Evolution strategies: A comprehensive introduction. Natural Computing, 1(1): 3–52.
60. *Beyer, H.-G. and Sendhoff, B*. (2008). Covariance matrix adaptation revisited: The CMSA evolution strategy. In *Rudolph, G., Jansen, T., Lucas, S., Poloni, C., and Beume, N., editors*, Parallel Problem Solving from Nature – PPSN X, pages 123–132. Springer.
61. *Bhattacharya, M*. (2008). Reduced computation for evolutionary optimization in noisy environment. Genetic and Evolutionary Computation Conference, Atlanta, Georgia, pages 2117–2122.
62. *Bishop, C*. (2006). Pattern Recognition and Machine Learning. Springer.
63. Bishop, J. (1989). Stochastic searching networks. First IEE Conference on Artificial Neural Networks, London, England, pages 329–331.
64. *Biswas, A., Dasgupta, S., Das, S., and Abraham, A*. (2007a). A synergy of differential evolution and bacterial foraging optimization for global optimization. Neural Network World, 17(6): 607–626.
65. *Biswas, A., Dasgupta, S., Das, S., and Abraham, A*. (2007b). Synergy of PSO and bacterial foraging optimization – A comparative study on numeri-

cal benchmarks. In Corchado, E., Corchado, J., and Abraham, A., editors, Innovations in Hybrid Intelligent Systems, pages 255–263. Springer.

66. *Blum, C.* (2005a). Ant colony optimization: Introduction and recent trends. Physics of Life Reviews, 2(4): 353–373.
67. *Blum, C.* (2005b). Beam-ACO – Hybridizing ant colony optimization with beam search: An application to open shop scheduling. Computers f3 Operations Research, 32(6): 1565–1591.
68. *Blum, C.* (2007). Ant colony optimization: Introduction and hybridizations. Seventh International Conference on Hybrid Intelligent Systems, Kaiserlautern, Germany, pages 24–29.
69. *Blum, C. and Dorigo, M.* (2004). The hypercube framework for ant colony optimization. IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics, 34(2): 1161–1172.
70. *Bonabeau, E., Theraulaz, G., and Dorigo, M.* (1999). Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press.
71. *Bonacich, P., Shure, G., Kahan, J., and Meeker, R.* (1976). Cooperation and group size in then-person prisoners' dilemma. The Journal of Conflict Resolution, 20(4): 687–706.
72. *Boslaugh, S. and Watters, P.* (2008). Statistics in a Nutshell. O'Reilly Media.
73. *Bosman, P. and Thierens, D.* (2003). The balance between proximity and diversity in multiobjective evolutionary algorithms. IEEE Transactions on Evolutionary Computation, 7(2): 174–188.
74. *Box, G.* (1957). Evolutionary operation: A method for increasing industrial productivity. Journal of the Royal Statistical Society, Series C (Applied Statistics), 6(2): 81–101.
75. *Box, J.* (1987). Guinness, Gosset, Fisher, and small samples. Statistical Science, 2(1): 45–52.
76. *Branke, J.* (1998). Creating robust solutions by means of evolutionary algorithms. In E*iben, A., Bäck, T., Schoenauer, M., and Schwefel, H.-P., editors*, Parallel Problem Solving from Nature – PPSN V, pages 119–128. Springer.
77. *Branke, J.* (1999). Efficient fitness estimation in noisy environments. Memory enhanced evolutionary algorithms for changing optimization problems, Washington, District of Columbia, pages 1875–1882.
78. *Branke, J.* (2002). Evolutionary Optimization in Dynamic Environments. Kluwer Academic Publishers.

79. *Branke, J.* (2012). Evolutionary Algorithms for Dynamic Optimization Problems (EvoDOP). http://people.aifb.kit.edu/jbr/EvoDOP.

80. *Branke, J., Orbayi, M., and Uyar, S.* (2006). The role of representations in dynamic knapsack problems. In Rothlauf, F., editor, Applications of Evolutionary Computing, pages 764–775. Springer.

81. *Branke, J., Schmidt, C., and Schmec, H.* (2001). Efficient fitness estimation in noisy environments. Genetic and Evolutionary Computation Conference, San Francisco, California, pages 243–250.

82. *Bratton, D. and Kennedy, J.* (2007). Defining a standard for particle swarm optimization. IEEE Swarm Intelligence Symposium, Honolulu, Hawaii, pages 120–127.

83. *Bremermann, H., Rogson, M., and Salaff, S.* (1966). Global properties of evolution processes. In *Pattee, H., Edlsack, E., Fein, L., and Callahan, A., editors*, Natural Automata and Useful Simulations, pages 3–41. Spartan Books.

84. *Brest, J.* (2009). Constrained real-parameter optimization with E-self-adaptive differential evolution. In *Mezura-Montes, E., editor*, Constraint-Handling in Evolutionary Optimization, pages 73–93. Springer.

85. *Brest, J., Zamuda, A., Boskovic, B., Maucec, M., and Zumer, V.* (2009). Dynamic optimization using self-adaptive differential evolution. IEEE Congress on Evolutionary Computation, Trondheim, Norway, pages 415–422.

86. *Bringmann, K. and Friedrich, T.* (2010). An efficient algorithm for computing hypervolume contributions. Evolutionary Computation, 18(3): 383–402.

87. *Bui, L., Abbass, H., and Essam, D.* (2005). Fitness inheritance for noisy evolutionary multi-objective optmization. Genetic and Evolutionary Computation Conference, Washington, District of Columbia, pages 779–785.

88. *Bureerat, S. and Sriworamas, K.* (2007). Population-based incremental learning for multiobjective optimisation. In S*aad, A., Dahal, K., Sarfraz, M., and Roy, R., editors*, Soft Computing in Industrial Applications, pages 223–232. Springer.

89. *Burke, E.* (2003). High-Tech Cycling. Human Kinetics, 2nd edition.

90. *Cai, C. and Wang, Y.* (2006). A multiobjective optimization-based evolutionary algorithm for constrained optimization. IEEE Transactions on Evolutionary Computation, 10(6): 658–675.

91. *Cakir, B., Altiparmak, F., and Dengiz, B.* (2011). Multi-objective optimization of a stochastic assembly line balancing: A hybrid simulated annealing algorithm. Computers & Industrial Engineering, 60(3): 376–384.
92. *Carlisle, A. and Dozier, G.* (2001). An off-the-shelf PSO. Particle Swarm Optimization Workshop, Indianapolis, Indiana, pages 1–6.
93. *Carlson, S. and Shonkwiler, R.* (1998). Annealing a genetic algorith over constraints. IEEE International Conference on Systems, Man, and Cybernetics, San Diego, California, pages 3931–3936.
94. *Černý, V.* (1985). Thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. Journal of Optimization Theory and Applications, 45(1):41–51.
95. *Chafekar, D., Shi, L., Rasheed, K., and Xuan, J.* (2005). Multiobjective GA optimization using reduced models. IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews, 35(2): 261–265.
96. *Chen, S. and Montgomery, J.* (2011). Selection strategies for initial positions and initial velocities in multi-optima particle swarms. Genetic and Evolutionary Computation Conference, Dublin, Ireland, pages 53–60.
97. *Chen, Y.-L. and Liu, C.-C.* (1994). Multiobjective VAr planning using the goalattainment method. IEE Proceedings on Generation, Transmission and Distribution, 141(3): 227–232.
98. *Cheng, C., Wang, W., Xu, D., and Chau, K.* (2008). Optimizing hydropower reservoir operation using hybrid genetic algorithm and chaos. Water Resources Management, 22(7): 895–909.
99. *Choi, S. and Moon, B.* (2003). Normalization in genetic algorithms. Genetic and Evolutionary Computation Conference, Chicago, Illinois, pages 862–873.
100. *Chow, C. and Liu, C.* (1968). Approximating discrete probability distributions with dependence trees. IEEE Transactions on Information Theory, IT-14(3): 462–467.
101. *Christensen, S. and Oppacher, F.* (2001). What can we learn from no free lunch? A first attempt to characterize the concept of a searchable function. Genetic and Evolutionary
102. Computation Conference, San Francisco, California, pages 1219–1226.
103. *Chuan-Chong, C. and Khee-Meng, K.* (1992). Principles and Techniques in Combinatorics. World Scientific.

104. *Chuang, C.-L. and Jiang, J.-A*. (2007). Integrated radiation optimization: Inspired by the gravitational radiation in the curvature of spacetime. IEEE Congress on Evolutionary Computation, Singapore, pages 3157–3164.
105. *Chung, H.-S. and Alonso, J*. (2004). Multiobjective optimization using approximation model-based genetic algorithms. 10th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Albany, New York.
106. *Chung, H.-S., Choi, S., and Alonso, J*. (2003). Supersonic business jet design using a knowledge-based genetic algorithm with an adaptive, unstructured grid methodology. 21st AIAA Applied Aerodynamics Conference, Orlando, Florida.
107. *Clement, P*. (1959). A class of triple-diagonal matrices for test purposes. SIAM Review, 1(1): 50–52.
108. *Clerc, M*. (1999). The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization. In IEEE Congress on Evolutionary Computing, pages 1951–1957.
109. *Clerc, M*. (2004). Discrete particle swarm optimization, illustrated by the traveling salesman problem. In *Onwubolu, G. and Babu, R., editors*, New Optimization Techniques in Engineering, pages 219–239. Springer.
110. *Clerc, M*. (2006). Particle Swarm Optimization. John Wiley & Sons.
111. *Clerc, M*. (2012a). Particle Swarm Optimization. http://clerc.maurice.free.fr/pso.
112. *Clerc, M*. (2012b). Randomness matters. Technical report. http://clerc.maurice. free. fr/pso.
113. *Clerc, M. and Kennedy, J*. (2002). The particle swarm – Explosion, stability, and convergence in a multidimensional complex space. IEEE Transactions on Evolutionary Computation, 6(1): 58–73.
114. *Clerc, M. and Poli, R*. (2006). Stagnation analysis in particle swarm optimisation or what happens when nothing happens. Technical report, University of Essex. http://clerc.maurice.free.fr/pso.
115. *Cobb, H. and Grefenstette, J*. (1993). Genetic algorithms for tracking changing environments. International Conference on Genetic Algorithms, Urbana-Champaign, Illinois, pages 523–530.
116. *Coello Coello, C*. (1999). A comprehensive survey of evolutionary–based multiobjective optimization techniques. Knowledge and Information Systems, 1(3): 269–308.

117. *Coello Coello, C.* (2000a). Constraint-handling using an evolutionary multiobjective optimization technique. Civil Engineering and Environmental Systems, 17(4): 319–346.

118. *Coello Coello, C.* (2000b). Use of a self-adaptive penalty approach for engineering optimization problems. Computers in Industry, 41(2): 113–127.

119. *Coello Coello, C.* (2002). Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. Computer Methods in Applied Mechanics and Engineering, 191(11–12): 1245–1287.

120. *Coello Coello, C.* (2006). Evolutionary multi-objective optimization: A historical view of the field. IEEE Computational Intelligence Magazine, 1(1): 28–36.

121. *Coello Coello, C.* (2009). Evolutionary multi-objective optimization: Some current research trends and topics that remain to be explored. Frontiers of Computer Science in China, 3(1): 18–30.

122. *Coello Coello, C.* (2012a). List of references on constraint-handling techniques used with evolutionary algorithms. www.cs.cinvestav.mx/~constraint.

123. *Coello Coello, C.* (2012b). List of references on evolutionary multiobjective optimization. www.lania.mx/~ccoello/EMOO/EMOObib.html.

124. *Coello Coello, C. and Becerra, R.* (2002). Constrained optimization using an evolutionary programming-based cultural algorithm. In Parmee, I., editor, Adaptive Computing in Design and Manufacture V, pages 317–328. Springer.

125. *Coello Coello, C. and Becerra, R.* (2003). Evolutionary multiobjective optimization using a cultural algorithm. Swarm Intelligence Symposium, Indianapolis, Indiana, pages 6–13.

126. *Coello Coello, C., Lamont, G., and Van Veldhuizen, D.* (2007). Evolutionary Algorithms for Solving Multi-Objective Problems. Springer.

127. *Coello Coello, C. and Mezura-Montes, E.* (2011). Constraint-handling in nature-inspired numerical optimization: Past, present and future. Swarm and Evolutionary Computation, 1(4): 173–194.

128. *Coit, D. and Smith, A.* (1996). Penalty guided genetic search for reliability design optimization. Computers and Industrial Engineering, 30(4): 895–904.

129. *Coit, D., Smith, A., and Tate, D.* (1996). Adaptive penalty methods for genetic optimization of constrained combinatorial problems. INFORMS Journal on Computing, 8(2): 173–182.

130. *Collard, P. and Aurand, J.* (1994). DGA: An efficient genetic algorithm. 11th European Conference on Artificial Intelligence, Amsterdam, The Netherlands, pages 487–492.

131. *Collard, P. and Gaspar, A.* (1996). «Royal-road» landscapes for a dual genetic algorithm. 12th European Conference on Artificial Intelligence, Budapest, Hungary, pages 213–217.

132. *Collette, Y. and Siarry, P.* (2004). Multiobjective Optimization: Principles and Case Studies. Springer.

133. *Colorni, A., Dorigo, M., and Maniezzo, V.* (1991). Distributed optimization by ant colonies. European Conference on Artificial Life, Paris, France, pages 134–142.

134. *Corder, G. and Foreman, D.* (2009). Nonparametric Statistics for Non–Statisticians. John Wiley & Sons.

135. *Cordon, 0., Herrera, F., de Viana, F., and Moreno, L.* (2000). A new ACO model integrating evolutionary computation concepts: The best-worst ant system. Prom Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms, Brussels, Belgium, pages 22–29.

136. *Corfman, K. and Lehmann, D.* (1994). The prisoner's dilemma and the role of information in setting advertising budgets. Journal of Advertising, 23(2): 35–48.

137. *Courant, R.* (1943). Variational methods for the solution of problems of equilibrium and vibrations. Bulletin of the American Mathematical Society, 49(1): 1–23.

138. *Cover, T. and Thomas, J.* (1991). Elements of Information Theory. Wiley-lnterscience.

139. *Cramer, N.* (1985). A representation for the adaptive generation of simple sequential programs. International Conference on Genetic Algorithms and Their Application, Pittsburgh, Pennsylvania, pages 183–187.

140. *Crepinsek, M., Liu, S.-H., and Mernik, L.* (2012). A note on teaching-learning-based optimization algorithm. Information Sciences, 212: 79–93.

141. *Crepinsek, M., Liu, S.-H., and Mernik, M.* (2013). Replication and comparison of computational experiments in applied evolutionary com-

puting: Common pitfalls and guidelines to avoid them. Information Sciences, submitted for publication.

142. *Culberson, J.* (1998). On the futility of blind search. Evolutionary Computation, 6(2): 109–127.
143. *Darwin, C.* (1859). On the Origin of Species by Means of Natural Selection, or The Preservation of Favoured Races in the Struggle for Life. John Murray, Albemarle Street.
144. *Darwin, C., Neve, M., and Messenger, S.* (2002). Autobiographies. Penguin Classics.
145. *Das, S., Biswas, A., Dasgupta, S., and Abraham, A.* (2009). Bacterial foraging optimization algorithm: Theoretical foundations, analysis, and applications. In *Abraham, A., Hassanien, A.-E., Siarry, P., and Engelbrecht, A., editors*, Foundations of Computational Intelligence – Volume 3: Global Optimization, pages 23–56. Springer.
146. *Das, S. and Suganthan, P.* (2011). Differential evolution: A survey of the state-of-the-art. IEEE Transactions on Evolutionary Computation, 15(1): 4–31.
147. *Das, S., Suganthan, P., and Coello Coello, C.* (2011). Guest editorial: Special issue on differential evolution. IEEE Transactions on Evolutionary Computation, 15(1): 1–3.
148. *Dasgupta, S., Das, S., Abraham, A., and Biswas, A.* (2009). Adaptive computational chemotaxis in bacterial foraging optimization: An analysis. IEEE Transactions on Evolutionary Computation, 13(4): 919–941.
149. *Davis, L.* (1985). Job shop scheduling with genetic algorithms. International Conference on Genetic Algorithms and Their Application, Pittsburgh, Pennsylvania, pages 136–140.
150. *Davis, L. and Steenstrup, M.* (1987). Genetic algorithms and simulated annealing: An overview. In Davis, L., editor, Genetic Algorithms and Simulated Annealing, pages 1–11. Pitman Publishing.
151. *Davis, T. and Principe, J.* (1991). A simulated annealing like convergence theory for the simple genetic algorithm. International Conference on Genetic Algorithms, San Diego, California, pages 174–181.
152. *Davis, T. and Principe, J.* (1993). A Markov chain framework for the simple genetic algorithm. Evolutionary Computation, 1(3): 269–288.
153. *De Bonet, J., Isbell, C., and Viola, P.* (1997). MMIC: Finding optima by estimating probability densities. In *Mozer, M., Jordan, M., and Petsche,*

T., editors, Advances in Neural Information Processing Systems 9, pages 424–430. MIT Press.

154. *de Franca, F., Coelho, G., Von Zuben, F., and Attux, R*. (2008). Multivariate ant colony optimization in continuous search spaces. In Genetic and Evolutionary Computation Conference, pages 9–16.
155. *de Garis, H.* (1990). Genetic programming: Building artificial nervous systems with genetically programmed neural network modules. Seventh International Conference on Machine Leaming, Austin, Texas, pages 132–139.
156. *De Jong, K. (*1975). An Analysis of the Behaviour of a Class of Genetic Adaptive Systems. PhD thesis, University of Michigan.
157. *De Jong, K.* (1992). Genetic algorithms are NOT function optimizers. Second Workshop on Foundations of Genetic Algorithms, Vail, Colorado, pages 5–17.
158. *De Jong, K.* (2002). Evolutionary Computation. The MIT Press.
159. *De Jong, K., Fogel, D., and Schwefel, H.-P.* (1997). A history of evolutionary computation. In *Bäck, T., Fogel, D., and Michalewicz, Z., editors*, Handbook of Evolutionary Computation, pages A2.3:1–12. Oxford University Press.
160. *de Oca, M. and Stützle, T.* (2008). Convergence behavior of the fully informed particle swarm optimization algorihtm. Genetic and Evolutionary Computation Conference, Atlanta, Georgia, pages 71–78.
161. *Deb, K*. (2000). An efficient constraint handling method for genetic algorithms. Computer Methods in Applied Mechanics and Engineering, 186(2–4): 311–338.
162. *Deb, K.* (2009). Multi-Objective Optimization using Evolutionary Algorithms. John Wiley & Sons.
163. *Deb, K. and Agrawal, R*. (1995). Simulated binary crossover for continuous search space. Complex Systems, 9(2): 115–148.
164. *Deb, K. and Agrawal, S*. (1999). A niched-penalty approach for constraint handling in genetic algorithms. International Conference on Artificial Neural Nets and Genetic Algorithms, Portoroz, Slovenia, pages 235–242.
165. *Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T*. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J., and Schwefel, H.-P., editors*, Parallel Problem Solving from Nature – PPSN VI, pages 849–858. Springer.

166. *Deb, K. and Goldberg, D*. (1989). An investigation of niche and species formation in genetic function optimization. International Conference on Genetic Algorithms, Fairfax, Virginia, pages 42–50.

167. *Deb, K., Mohan, M., and Mishra, S*. (2005). Evaluating the ϵ-domination based multiobjective evolutionary algorithm for a quick computation of Pareto-optimal solutions. Evolutionary Computation, 13(4): 501–525.

168. *Deb, K., Pratap, A., Agarwal, S., and Meyarivan, T*. (2002a). A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, 6(2): 182–197.

169. *Deb, K., Pratap, A., and Meyarivan, T.* (2001). Constrained test problems for multiobjective evolutionary optimization. In *Zitzler, E., Deb, K., Thiele, L., Coello Coello, C., and Corne, D., editors*, Evolutionary Multi-Criterion Optimization: First International Conference, EMO 2001, pages 284–298. Springer.

170. *Deb, K., Thiele, L., Laumanns, M., and Zitzler, E*. (2002b). Scalable multi-objective optimization test problems. World Congress on Computational Intelligence, Honolulu, Hawaii, pages 825–830.

171. *Dechter, R*. (2003). Constraint Processing. Morgan Kaufmann.

172. *Deep, K. and Thakur, M.* (2007). A new crossover operator for real coded genetic algorithms. Applied Mathematics and Computation, 188(1): 895–911.

173. *del Valle, Y., Venayagamoorthy, G., Mohagheghi, S., Hernandez, J.-C., and Harley, R.* (2008). Particle swarm optimization: Basic concepts, variants and applications in power systems. IEEE Transactions on Evolutionary Computation, 12(2): 171–195.

174. *Delahaye, J.-P. and Mathieu, P.* (1995). Complex strategies in the iterated prisoner's dilemma. In Albert, A., editor, Chaos and Society, pages 283–292. IOS Press.

175. *Delsuc, F.* (2003). Army ants trapped by their evolutionary history. Public Library of Science Biology, 1(2): e37.

176. *Dembski, W. and Marks, R*. (2009a). Bernoulli's principle of insufficient reason and conservation of information in computer search. IEEE Conference on Systems, Man and Cybernetics, San Antonio, Texas, pages 2647–2652.

177. *Dembski, W. and Marks, R*. (2009b). Conservation of information in search: Measuring the cost of success. IEEE Transactions on Systems, Man, and Cybernetics – Part A: Systems and Humans, 39(5): 1051–1060.

178. *Dembski, W. and Marks, R.* (2010). The search for a search: Measuring the information cost of higher level search. Journal of Advanced Computational Intelligence and Intelligent Informatics, 14(5): 475–486.

179. *Demetrescu, C.* (2012). 9th DIMACS Implementation Challenge – Shortest Paths. www.dis.uniroma1.it/challenge9.

180. *Deneubourg, J.-L., Aron, S., Goss, S., and Pasteels, J.* (1990). The self–organizing exploratory pattern of the Argentine ant. Journal of Insect Behavior, 3(2): 159–168.

181. *DePaulo, B., Kashy, D., Kirkendol, S., Wyer, M., and Epstein, J.* (1996). Lying in everyday life. Journal of Personality and Social Psychology, 70(5): 979–995.

182. *Devroye, L.* (1978). Progressive global random search of continuous functions. Mathematical Programming, 15(1): 330–342.

183. *Di Pietro, A., While, L., and Barone, L.* (2004). Applying evolutionary algorithms to problems with noisy, time-consuming fitness functions. IEEE Congress on Evolutionary Computation, Portland, Oregon, pages 1254–1261.

184. *Dominguez, J. and Pulido, G.* (2011). A comparison on the search of particle swarm optimization and differential evolution on multi-objective optimization. IEEE Congress on Evolutionary Computation, New Orleans, Louisiana, pages 1978–1985.

185. *Doran, R.* (2007). The gray code. Journal of Universal Computer Science, 13(11): 1573–1597.

186. *Dorigo, M., Birattari, M., and Stützle, T.* (2006). Ant colony optimization: Artificial ants as a computational intelligence technique. IEEE Computational Intelligence Magazine, 1(4): 28–39.

187. *Dorigo, M. and Gambardella, L.* (1997a). Ant colonies for the traveling salesman problem. BioSystems, 43(2): 73–81.

188. *Dorigo, M. and Gambardella, L.* (1997b). Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation, 1(1): 53–66.

189. *Dorigo, M., Maniezzo, V., and Colorni, A.* (1996). Ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics, 26(1): 29–41.

190. *Dorigo, M. and Stützle, T.* (2004). Ant Colony Optimization. The MIT Press.

191. *Dorigo, M. and Stützle, T.* (2010). Ant colony optimization: Overview and recent advances. In Gendreau, M. and Potvin, J.-Y., editors, Handbook of Metaheuristics, pages 227–263. Springer.

192. *Droste, S., Jansen, T., and Wegener, I.* (2002). On the analysis of the (1+1) evolutionary algorithm. Theoretical Computer Science, 276(1–2): 51–81.

193. *Du, D., Simon, D., and Ergezer, M.* (2009). Biogeography-based optimization combined with evolutionary strategy and immigration refusal. IEEE Conference on Systems, Man, and Cybernetics, San Antonio, Texas, pages 1023–1028.

194. *Duan, Q., Gupta, V., and Sorooshian, S.* (1993). Shuffled complex evolution approach for effective and efficient global minimization. Journal of Optimization Theory and Applications, 76(3): 501–521.

195. *Duan, Q., Sorooshian, S., and Gupta, V.* (1992). Effective and efficient global optimization for conceptual rainfall-runoff models. Water Resources Research, 28(4): 1015–1031.

196. *Ducheyne, E., De Baets, B., and De Wulf, R.* (2003). Is fitness inheritance useful for realworld applications? Second International Conference on Evolutionary Multi-Criterion Optimization, Faro, Portugal, pages 31–42.

197. *Dueck, G.* (1993). New optimisation heuristics: The great deluge algorithm and the record-to-record travel. Journal of Computational Physics, 104(86): 86–92.

198. *Dueck, G. and Scheuer, T.* (1990). Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. Journal of Computational Physics, 90(1): 161–175.

199. *Dunham, B., Fridshal, D., Fridshal, R., and North, J.* (1963). Design by natural selection. Synthese, 15(2): 254–259.

200. *Durham, W.* (1992). Coevolution: Genes, Culture, and Human Diversity. Stanford University Press.

201. *Dyson, G.* (1998). Darwin Among the Machines. Basic Books.

202. *Eberhart, R. and Kennedy, J.* (1995). A new optimizer using particle swarm theory. International Symposium on Micro Machine and Human Science, Nagoya, Japan, pages 39–43.

203. *Eberhart, R. and Shi, Y.* (2000). Comparing inertia weights and constriction factors in particle swarm optimization. IEEE Congress on Evolutionary Computation, San Diego, California, pages 84–88.

204. *Eberhart, R. and Shi, Y.* (2001). Particle swarm optimization: Developments, applications and resources. IEEE Congress on Evolutionary Computation, Seoul, Korea, pages 81–86.

205. *Edgeworth, F.* (1881). Mathematical Physics. Kegan Paul.

206. *Ehrgott, M.* (2005). Multicriteria Optimization. Springer.

207. *Ehrnborg, C. and Rosén, T.* (2009). The psychology behind doping in sport. Growth Hormone & IGF Research, 19(4): 285–287.

208. *Eiben, A.* (2000). Multiparent recombination. In *Bäck, T., Fogel, D., and Michalewicz, Z., editors*, Evolutionary Computation 1: Basic Algorithms and Operators, pages 289–307. Institute of Physics Publishing.

209. *Eiben, A.* (2001). Evolutionary algorithms and cnstraint satisfaction: Definitions, survey, methodology, and research directions. In *Kallel, L., Naudts, B., and Rogers, A., editors*, Theoretical Aspects of Evolutionary Computating, pages 13–30. Springer.

210. *Eiben, A.* (2003). Multiparent recombination in evolutionary computing. In *Ghosh, A. and Tsutsui, S., editors*, Advances in Evolutionary Computing, pages 175–192. SpringerVerlag.

211. *Eiben, A. and Bäck, T.* (1998). Empirical investigation of multiparent recombination operators in evolution strategies. Evolutionary Computation, 5(3): 347–365.

212. *Eiben, A. and Schippers, C.* (1996). Multi-parent's niche: n-ary crossovers on nklandscapes. In *Ebeling, W., Rechenberg, I., Schwefel, H.-P., and Voigt, H.-M., editors*, Parallel Problem Solving from Nature – PPSN IV, pages 319–328. Springer.

213. *Eiben, A. and Smit, S.* (2011). Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm and Evolutionary Computation, 1(1): 19–31.

214. *Eiben, A. and Smith, J.* (2010). Introduction to Evolutionary Computing. Springer.

215. *Elbeltagi, E., Hegazy, T., and Grierson, D.* (2005). Comparison among five evolutionarybased optimization algorithms. Advanced Engineering Informatics, 19(1): 43–53.

216. *Ellis, T. and Yao, X.* (2007). Evolving cooperation in the non-iterated prisoner's dilemma: A social network inspired approach. IEEE Congress on Evolutionary Computation, Singapore, pages 736–743.

217. *Elton, C.* (1958). Ecology of Invasions by Animals and Plants. Chapman & Hall.

218. *Emre, E. and Knowles, G.* (1987). A Newton-like approximation algorithm for the steadystate solution of the Riccati equation for time-varying systems. Optimal Control Applications and Methods, 8(2): 191–197.

219. *Engelbrecht, A*. (2003). Computational Intelligence. John Wiley & Sons.

220. *English, T.* (1999). Some information theoretic results on evolutionary optimization. IEEE Congress on Evolutionary Computation, Washington, District of Columbia, pages 788–795.

221. *Ergezer, M.* (2011). Oppositional biogeography-based optimization. Technical report, Cleveland State University. Doctoral dissertation proposal, unpublished.

222. *Ergezer, M. and Simon, D.* (2011). Oppositional biogeography-based optimization for combinatorial problems. IEEE Congress on Evolutionary Computation, New Orleans, Louisiana, pages 1496–1503.

223. *Ergezer, M., Simon, D., and Du, D*. (2009). Oppositional biogeography-based optimization. IEEE Conference on Systems, Man, and Cybernetics, San Antonio, Texas, pages 1035–1040.

224. *Erol, 0. and Eksin, I.* (2006). New optimization method: Big bang-big crunch. Advances in Engineering Software, 37(2): 106–111.

225. *Eshelman, L., Caruana, R., and Schaffer, J*. (1989). Biases in the crossover landscape. International Conference on Genetic Algorithms, Fairfax, Virginia, pages 10–19.

226. *Eshelman, L. and Schaffer, J*. (1993). Real-coded genetic algorithms and interval schemata. In Whitley, D., editor, Foundations of Genetic Algorithms 2, pages 187–202. Morgan Kaufmann.

227. *Eskandari, H. and Geiger, C*. (2008). A fast Pareto genetic algorithm approach for solving expensive multiobjective optimization problems. Journal of Heuristics, 14(3): 203–241.

228. *Eusuff, M. and Lansey, K*. (2003). Optimization of water distribution network design using the shuffled frog leaping algorithm (SFLA). Journal of Water Resources Planning and Management, 129(3): 210–225.

229. *Eusuff, M., Lansey, K., and Pasha, F*. (2006). Shuffled frog-leaping algorithm: A memetic meta-heuristic for discrete optimization. Engineering Optimization, 38(2): 129–154.

230. *Evans, M., Hastings, N., and Peacock, B*. (2000). Statistical Distributions. Wiley-lnterscience.

231. *Farmani, R. and Wright, J.* (2003). Self-adaptive fitness formulation for constrained optimization. IEEE Transactions on Evolutionary Computation, 7(5): 445–455.

232. *Fausett, L.* (1994). Fundamentals of Neural Networks. Prentice Hall.

233. *Fealy, M.* (2006). The Great Pawn Hunter Chess Tutorial. AuthorHouse.

234. *Feoktistov, V.* (2006). Differential Evolution: In Search of Solutions. Springer.

235. *Fernandes, M., Martins, T., and Rocha, A.* (2009). Fish swarm intelligent algorithm for bound constrained global optimization. International Conference on Computational and Mathematical Methods in Science and Engineering, Gijón, Spain.

236. *Fish, F.* (1995). Kinematics of ducklings swimming in formation: Consequences of position. Journal of Experimental Zoology, 273(1): 1–11.

237. *Fleming, P., Purshouse, R., and Lygoe, R.* (2005). Many-objective optimization: An engineering design perspective. In *Coello Coello, C., Hernandez Aguirre, A., and Zitzler, E., editors*, Evolutionary Multi-Criterion Optimization, pages 14–32. Springer.

238. *Fletcher, R. and Powell, M.* (1963). A rapidly convergent descent method for minimization. The Computer Journal, 6(2): 163–168.

239. *Floudas, C. and Pardalos, P.* (1990). A Collection of Test Problems for Constrained Global Optimization Algorithms. Springer.

240. *Floudas, C., Pardalos, P., Adjiman, C., Esposito, W., Giimiis, Z., Harding, S., Klepeis, J., Meyer, C., and Schweiger, C.* (2010). Handbook of Test Problems in Local and Global Optimization. Springer.

241. *Fogel, D.* (1988). An evolutionary aproach to the traveling salesman problem. Biological Cybernetics, 60(2): 139–144.

242. *Fogel, D.* (1990). A parallel pocessing approach to a multiple traveling salesman problem using evolutionary programming. Fourth Annual Parallel Processing Symposium, Fullerton, California, pages 318–326.

243. *Fogel, D., editor* (1998). Evolutionary Computation: The Fossil Record. Wiley-IEEE Press.

244. *Fogel, D.* (2000). What is evolutionary computation? IEEE Spectrum, 37(2): 26–32.

245. *Fogel, D.* (2006). George Friedman – Evolving circuits for robots. IEEE Computational Intelligence Magazine, 1(4): 52–54.

246. *Fogel, D. and Anderson, R.* (2000). Revisiting Bremermann's genetic algorithm: I. Simultaneous mutation of all parameters. IEEE Congress on Evolutionary Computation, San Diego, California, pages 1204–1209.
247. *Fogel, L.* (1999). Intelligence through Simulated Evolution: Forty Years of Evolutionary Programming. John Wiley & Sons.
248. *Fogel, L., Owens, A., and Walsh, M.* (1966). Artificial Intelligence through Simulated Evolution. John Wiley & Sons.
249. *Fonseca, C. and Fleming, P.* (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. International Conference on Genetic Algorithms, Urbana-Champaign, Illinois, pages 416–423.
250. *Fonseca, C. and Fleming, P.* (1995). An overview of evolutionary algorithms in multiobjective optimization. Evolutionary Computation, 3(1): 1–16.
251. *Formato, R.* (2007). Central force optimization: A new metaheuristic with applications in applied electromagnetics. Progress in Electromagnetics Research, 77: 425–491.
252. *Formato, R.* (2008). Central force optimization: A new nature inspired computational framework for multidimensional search and optimization. In *Krasnogor, N., Nicosia, G., Pavone, M., and Pelta, D., editors*, Nature Inspired Cooperative Strategies for Optimization (NICSO 2007), pages 221–238. Springer.
253. *Forsyth, R.* (1981). BEAGLE – A Darwinian approach to pattern recognition. Kybernetes, 10(3): 159–166.
254. *Fourman, M.* (1985). Compaction of symbolic layout using genetic algorithms. International Conference on Genetic Algorithms, Pittsburgh, Pennsylvania, pages 141–153.
255. *Fox, B. and McMahon, M.* (1991). Genetic operators for sequencing problems. In *Rawlins, G., editor*, Foundations of Genetic Algorithms, pages 284–300. Morgan Kaufmann Publishers.
256. *Francqs, O.* (1998). An evolutionary strategy for global minimization and its Markov chain analysis. IEEE Transactions on Evolutionary Computation, 2(3): 77–90.
257. *Fraser, A.* (1957). Simulation of genetic systems by automatic digital computers: I. Introduction. Australian Journal of Biological Sciences, 10(3): 484–491.
258. *Friedberg, R.* (1958). A learning machine: Part I. IBM Journal of Research and Development, 2(1): 2–13.

259. *Friedberg, R., Dunham, B., and North, J.* (1958). A learning machine: Part II. IBM Journal of Research and Development, 3(3): 282–287.

260. *Friedman, G.* (1998). Selective feedback computers for engineering synthesis and nervous system analogy. In *Fogel, D., editor*, Evolutionary Computation: The Fossil Record, pages 30–84. Wiley-IEEE Press.

261. *Furuta, H., Maeda, K., and Watanabe, E.* (1995). Application of genetic algorithm to aesthetic eesign of bridge structures. Computer-Aided Civil and Infrastructure Engineering, 10(6): 415–421.

262. *Galinier, P., Ramiez, J.-P., Hao, J.-K., and Porumbel, D.* (2013). Recent advances in graph vertex coloring. In *Zelinka, I., Snasel, V., and Abraham, A., editors*, Handbook of Optimization. ebooks.com.

263. *Gallagher, M., Wood, I., Keith, J., and Sofronov, G.* (2007). Bayesian inference in estimation of distribution algorithms. IEEE Congress on Evolutionary Computation, Singapore, pages 127–133.

264. *Gambardella, L. and Dorigo, M.* (1995). Ant-Q: A reinforcement learning approach to the traveling salesman problem. Twelfth International Conference on Machine Learning, Tahoe City, California, pages 252–260.

265. *Gandomi, A. and Alavi, A.* (2012). Krill herd: A new bio-inspired optimization algorithm. Communications in Nonlinear Science and Numerical Simulation, 17(12): 4831–4845.

266. *Gathercole, C. and Ross, P.* (1994). Dynamic training subset selection for supervised learning in genetic programming. In *Davidor, Y., Schwefel, H.-P., and Manner, R., editors*, Parallel Problem Solving from Nature – PPSN III, pages 312–321. Springer.

267. *Gathercole, C. and Ross, P.* (1997). Small populations over many generations can beat large populations over few generations in genetic programming. Second Annual Conference on Genetic Programming, Palo Alto, California, pages 111–118.

268. *Geem, Z., editor* (2010a). Harmony Search Algorithms for Structural Design Optimization. Springer.

269. *Geem, Z., editor* (2010b). Music-Inspired Harmony Search Algorithm. Springer.

270. *Geem, Z.* (2010c). Recent Advances in Harmony Search Algorithm. Springer.

271. *Geem, Z., Kim, J.-H., and Loganathan, G.* (2001). A new heuristic optimization algorithm: Harmony search. Simulation, 76(2): 60–68.

272. *Geisser, S.* (1993). Predictive Inference. Chapman & Hall.
273. *Geman, S. and Geman, D.* (1984). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. IEEE Transactions on Pattern Analysis and Machine Intelligence, 6(6): 721–741.
274. *Gendreau, M.* (2003). An introduction to tabu search. In *Glover, F. and Kochenberger, G., editors*, Handbook of Metaheuristics, pages 37–54. Springer.
275. *Gendreau, M. and Potvin, J.-Y.* (2010). Tahu search. In Gendreau, M. and Potvin, J.-Y., editors, Handbook of Metaheuristics, pages 41–59. Springer.
276. *Giraldeau, L.-A. and Caraco, T.* (2000). Social Foraging Theory. Princeton University Press.
277. *Glover, F. and Laguna, M.* (1998). Tabu Search. Springer.
278. *Glover, F. and McMillan, C.* (1986). The general employee scheduling problem: An integration of MS and AI. Computers and Operations Research, 13(5): 563–573.
279. *Goh, C. and Tan, K.* (2007). An investigation on noisy environments in evolutionary multiobjective optimization. IEEE Transactions on Evolutionary Computation, 11(3): 354–381.
280. *Golan, J.* (2007). The Linear Algebra a Beginning Graduate Student Ought to Know. Springer.
281. *Goldberg, D.* (1989a). Genetic Algorithms in Search, Optimization, and Machine Learning. Addison Wesley.
282. *Goldberg, D.* (1989b). Messy genetic algorithms: Motivation, analysis, and first results. Complex Systems, 3(5): 493–530.
283. *Goldberg, D.* (1991). Real-coded genetic algorithms, virtual alphabets, and blocking. Complex Systems, 5(2): 139–167.
284. *Goldberg, D. and Lingle, R.* (1985). Alleles, loci, and the traveling salesman problem. International Conference on Genetic Algorithms and Their Application, Pittsburgh, Pennsylvania, pages 154–159.
285. *Gómez, J., Barrera, J., Rojas, J., Macias-Samano, J., Liedo, J., Cruz-Lopez, L., and Badü, M.* (2005). Volatile compounds released by disturbed females of Cephalonomia stephanoderis (Hymenoptera: Bethylidae): A parasitoid of the coffee berry borer Hypothenemus hampei (Coleoptera: Scolytidae). Florida Entomologist, 88(2): 180–187.
286. *González, C., Lozano, J., and Larrañaga, P.* (2000). Analyzing the PBIL algorithm by means of discrete dynamical systems. Complex Systems, 12(4): 465–479.

287. *González, C., Lozano, J., and Larrañaga, P.* (2001). The convergence behavior of the PBIL algorithm: A preliminary approach. In *Kůrková, V., Steele, N., Neruda, R., and Kárný, M., editors*, Artificial Neural Nets and Genetic Algorithms, pages 228–231. Springer-Verlag.

288. *González, C., Lozano, J., and Larrañaga, P.* (2002). Mathematical modeling of discrete estimation of distribution algorithms. In *Larrañaga, P. and Lozano, J., editors*, Estimation of Distribution Algorithms, pages 147–163. Kluwer Academic Publishers.

289. *Good, P. and Hardin, J.* (2009). Common Errors in Statistics. John Wiley & Sons, 3rd edition.

290. *Goss, S., Aron, S., Deneubourg, J., and Pasteels, J.* (1989). Self-organized shortcuts in the Argentine ant. Naturwissenschaften, 76(12): 579–581.

291. *Gotelli, N.* (2008). A Primer of Ecology. Sinauer Associates.

292. *Gray, R.* (2011). Entropy and Information Theory. Springer.

293. *Greene, M. and Gordon, D.* (2007). Structural complexity of chemical recognition cues affects the perception of group membership in the ants Linephithema humile and Aphaenogaster cockerelli. Journal of Experimental Biology, 210(5): 897–905.

294. *Grefenstette, J., Gopal, R., Rosmaita, B., and Van Gucht, D.* (1985). Genetic algorithms for the TSP. International Conference on Genetic Algorithms and Their Application, Cambridge, Massachusetts, pages 160–165.

295. *Gregory, R. and Karney, D.* (1969). A Collection of Matrices for Testing Computational Algorithms. John Wiley & Sons.

296. *Grieco, J.* (1988). Realist theory and the problem of international cooperation: Analysis with an amended prisoner's dilemma model. The Journal of Politics, 50(3): 600–624.

297. *Grinstead, C. and Snell, J.* (1997). Introduction to Probability. American Mathematical Society.

298. *Grötschel, M. and Padberg, M.* (2001). The optimized odyssey. AIROnews, 6(2): 1–7.

299. *Guntsch, M. and Middendorf, M.* (2002). Applying population based ACO to dynamic optimization problems. Third International Workshop on Ant Algorithms, Brussels, Belgium, pages 111–122.

300. *Gustafson, S. and Burke, E.* (2006). Speciating island model: An alternative parallel evolutionary algorithm. Parallel and Distributed Computing, 66(8): 1025–1036.

301. *Gutierrez, A., Lanza, M., Barriuso, I., Valle, L., Domingo, M., Perez, J., and Basterrechea, J.* (2002). Comparison of different PSO initialization techniques for high dimensional search space problems: A test with FSS and antenna arrays. 5th European Conference on Antennas and Propagation, Rome, Italy, pages 965–969.

302. *Gutin, G. and Punnen, A., editors* (2007). The Traveling Salesman Problem and Its Variations. Springer.

303. *Gutjahr, W.* (2000). A graph–based ant system and its convergence. Future Generation Computer Systems, 16(9): 873–888.

304. *Gutjahr, W.* (2008). First steps to the runtime complexity analysis of ant colony optimization. Computers & Operations Research, 35(9): 2711–2727.

305. *Hadj-Alouane, A. and Bean, J.* (1993). A genetic algorithm for the multiple choice integer program. Technical report, Department of Industrial & Operations Engineering, University of Michigan. http://ioe .engin.umich.edu/techrprt/pdf/TR92–50.pdf.

306. *Hadj-Alouane, A. and Bean, J.* (1997). A genetic algorithm for the multiple choice integer program. Operations Research, 45(1): 92–101.

307. *Hajela, P. and Lin, C.-Y.* (1997). Genetic search strategies in multicriterion optimal design. Structural and Multidisciplinary Optimization, 4(2): 99–107.

308. *Hamida, S. and Schoenauer, M.* (2000). An adaptive algorithm for constrained optimization problems. In *Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J., and Schwefel, H.-P., editors*, Parallel Problem Solving from Nature – PPSN VI, pages 529–538. Springer.

309. *Hamida, S. and Schoenauer, M.* (2002). ASCHEA: New results using adaptive segregational constraint handling. IEEE Congress on Evolutionary Computation, Honolulu, Hawaii, pages 884–889.

310. *Hamilton, W.* (1971). Geometry for the selfish herd. Journal of Theoretical Biology, 31(2): 295–311.

311. *Hansen, N.* (2010). The CMA evolution strategy: A comparing review. In *Lozano, J., Larraiiga, P., Inza, I., and Bengoetxea, E., editors*, Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms, pages 75–102. Springer.

312. *Hansen, N., Miiller, S., and Koumoutsakos, P.* (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evolutionary Computation, 11(1): 1–18.

313. *Hansen, N. and Ostermeier, A.* (2001). Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation, 9(2): 159–195.

314. *Hanski, I.* (1999). Habitat connectivity, habitat continuity, and metapopulations in dynamic landscapes. Oikos, 87(2): 209–219.

315. *Hanski, I. and Gilpin, M.* (1997). Metapopulation Biology. Academic Press.

316. *Hao, J.-K. and Middendorf, M., editors* (2012). Evolutionary Computation in Combinatorial Optimization. Springer.

317. *Harding, S.* (2006). Animate Earth. Chelsea Green Publishing Company.

318. *Harik, G.* (1995). Finding multimodal solutions using restricted tournament selection. International Conference on Genetic Algorithms, Pittsburgh, Pennsylvania, pages 24–31.

319. *Harik, G.* (1999). Linkage learning via probabilistic modeling in the ECGA. Technical report, Illinois Genetic Algorithms Laboratory, University of Illinois. IlliGAL Report No. 99010.

320. *Harik, G., Lobo, F., and Goldberg, D.* (1999). The compact genetic algorithm. IEEE Transactions on Evolutionary Computation, 3(4): 287–297.

321. *Harik, G., Lobo, F., and Sastry, K.* (2010). Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ecga). In *Pelikan, M., Sastry, K., and CantuPaz, E., editors*, Scalable Optimization via Probabilistic Modeling, pages 39–62. Springer.

322. *Harrald, P. and Fogel, D.* (1996). Evolving continuous behaviors in the iterated prisoner's dilemma. Biosystems, 37(1–2): 135–145.

323. *Hastie, T., Tibshirani, R., and Friedman, J.* (2009). The Elements of Statistical Learning. Springer, 2nd edition.

324. *Hastings, A. and Higgins, K.* (1994). Persistence of transients in spatially structured models. Science, 263(5150): 1133–1136.

325. *Hastings, W.* (1970). Monte Carlo sampling methods using Markov chains and their applications. Biometrika, 57(1): 97–109.

326. *Hatzakis, I. and Wallace, D.* (2006). Dynamic multi-objective optimization with evolutionary algorithms: A forward-looking approach. Genetic and Evolutionary Computation Conference, Seattle, Washington, pages 1201–1208.

327. *Haupt, R. and Haupt, S.* (2004). Practical Genetic Algorithms. John Wiley & Sons, 2nd edition.

328. *Hauptman, A., Elyasaf, A., Sipper, M., and Karmon, A*. (2009). GP-Rush: Using genetic programming to evolve solvers for the rush hour puzzle. Genetic and Evolutionary Computation Conference, Montreal, Canada, pages 955–962.

329. *Hauptman, A. and Sipper, M*. (2007). Evolution of an efficient search algorithm for the mate-in-n problem in chess. European Conference on Genetic Programming, Valencia, Spain, pages 78–89.

330. *He, S., Wu, Q., and Saunders, J.* (2009). Group search optimizer: An optimization algorithm inspired by animal searching behavior. IEEE Transactions on Evolutionary Computation, 13(5): 973–990.

331. *Heinrich, B.* (2002). Why We Run. Harper Perennial.

332. *Helwig, S. and Wanka, R*. (2008). Theoretical analysis of initial particle swarm behavior. In *Rudolph, G., Jansen, T., Lucas, S., Poloni, C., and Beume, N., editors*, Parallel Problem Solving from Nature – PPSN X, pages 889–898. Springer.

333. *Henderson, D., Jacobson, S., and Johnson, A*. (2003). The theory and practice of simulated annealing. In Glover, F. and Kochenberger, G., editors, Handbook of Metaheuristics, pages 287–320. Springer.

334. *Herrera, F., Lozano, M., and Verdegay, J*. (1998). Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. Artificial Intelligence Review, 12(4): 265– 319.

335. *Hofmeyr, S. and Forrest, S*. (2000). Architecture for an artificial immune system. Evolutionary Computation, 8(4): 443–4 73.

336. *Holland, J.* (1975). Adaptation in Natural and Artificial Systems. The University of Michigan Press.

337. *Hölldobler, B. and Wilson, E*. (1990). The Ants. The Belknap Press of Harvard University Press.

338. *Hölldobler, B. and Wilson, E*. (1994). Journey to the Ants. The Belknap Press of Harvard University Press.

339. *Hölldobler, B. and Wilson, E*. (2008). The Superorganism: The Beauty, Elegance, and Strangeness of Insect Societies. W. W. Norton & Company.

340. *Homaifar, A., Qi, C., and Lai, S.* (1994). Constrained optimization via genetic algorithms. Simulation, 62(4): 242–253.

341. *Hooker, J.* (1995). Testing heuristics: We have it all wrong. Journal of Heuristics, 1(1): 33–42.

342. *Horn, J., Nafpliotis, N., and Goldberg, D.* (1994). A niched Pareto genetic algorithm for multiobjective optimization. IEEE Conference on Evolutionary Computation, Orlando, Florida, pages 82–87.

343. *Horne, E. and Jaeger, R.* (1988). Territorial pheromones of female red-backed salamanders. Ethology, 78(2): 143–152.

344. *Horoba, C. and Neumann, F.* (2010). Approximating Pareto-optimal sets using diversity strategies in evolutionary multi-objective optimization. In *Coello Coello, C., Dhaenens, C., and Jourdan, L., editors*, Advances in Multi-Objective Nature Inspired Computing, pages 23–44. Springer.

345. *Houck, C., Joines, J., and Kay, M.* (1995). A genetic algorithm for function optimization: A Matlab implementation. Technical report, North Carolina State University.

346. *Hsiao, Y.-T., Chuang, C.-L., Jiang, J.-A., and Chien, C.-C.* (2005). A novel optimization algorithm: Space gravitational optimization. IEEE International Conference on Systems, Man and Cybernetics, Waikoloa, Hawaii, pages 2323–2328.

347. *Hu, T., Harding, S., and Banzhaf, W.* (2010). Variable population size and evolution acceleration: A case study with a parallel evolutionary algorithm. Genetic Programming and Evolvable Machines, 11(2): 205–225.

348. *Huang, H. and Wang, F.* (2002). Fuzzy decision-making design of chemical plant using mixed-integer hybrid differential evolution. Computers and Chemical Engineering, 26(12): 1649–1660.

349. *Huang, V., Qin, A., Deb, K., Zitzler, E., Suganthan, P., Liang, J., Preuss, M., and Huband, S.* (2007). Problem definitions for performance assessment on multi-objective optimization algorithms. Technical report. www.ntu.edu.sg/home/EPNSugan/index_files/cec-benchmarking.htm.

350. *Huff, D. and Geis, I.* (1993). How to Lie with Statistics. W. W. Norton & Company.

351. *Iba, H. and de Garis, H.* (1996). Extending genetic programming with recombinative guidance. In *Angeline, P. and Kinnear, K., editors*, Advances in Genetic Programming: Volume 2, pages 69–88. The MIT Press.

352. *Igelnik, B. and Simon, D.* (2011). The eigenvalues of a tridiagonal matrix in biogeography. Applied Mathematics and Computation, 218(1): 195–201.

353. *Ingber, L.* (1996). Adaptive simulated annealing: Lessons learned. Control and Cybernetics, 25(1): 33–54.

354. *Ito, K., Akagi, S., and Nishikawa, M.* (1983). A multiobjective optimization approach to a design problem of heat insulation for thermal distribution piping network systems. Journal of Mechanisms, Transmissions, and Automation in Design, 105(2): 206–213.

355. *Jaszkiewicz, A. and Zielniewicz, P.* (2006). Pareto memetic algorithm with path relinking for bi–objective traveling salesperson problem. European Journal of Operational Research, 193(3): 885–890.

356. *Jayalakshmi, G., Sathiamoorthy, S., and Rajaram, R.* (2001). A hybrid genetic algorithm – A new approach to solve traveling salesman problem. International Journal of Computational Engineering Science, 2(2): 339–355.

357. *Jefferson, D., Collins, R., Cooper, C., Dyer, M., Flowers, M., Korf, R., Taylor, C., and Wang, A.* (2003). Evolution as a theme in artificial life: The genesys/tracker system. In *Langton, C., Taylor, C., Farmer, J., and Rasmussen, S., editors*, Artificial Life II, pages 549–578. Westview Press.

358. *Jensen, T. and Toft, B.* (1994). Graph Coloring Problems. John Wiley & Sons.

359. *Jin, Y.* (2005). A comprehensive survey of fitness approximation in evolutionary computation. Soft Computing, 9(1): 3–12.

360. *Jin, Y. and Branke, J.* (2005). Evolutionary optmization in uncertain environments – A survey. IEEE Transactions on Evolutionary Computation, 9(3): 303–317.

361. *Jin, Y., Hiiskin, M., and Sendhoff, B.* (2003). Quality measures for approximate models in evolutionary computation. Genetic and Evolutionary Computation Conference, Chicago, Illinois, pages 170–173.

362. *Jofré, P., Reisenegger, A., and Fernández, R.* (2006). Constraining a possible time variation of the gravitational constant through «gravitochemical heating» of neutron stars. Physical Review Letters, 97(13): 131102.

363. *Johnson, D.* (1999). The insignificance of statistical significance testing. Journal of Wildlife Management, 63(3): 763–772.

364. *Joines, J. and Houck, C.* (1994). On the use of non–stationary penalty functions to solve nonlinear constrained optimization problems with GA's. IEEE World Congress on Computational Intelligence, Orlando, Florida, pages 579–584.

365. *Jones, D., Schonlau, M., and Welch, W.* (1998). Efficient global optimization of expensive black-box functions. Journal of Global Optimization, 13(4): 455–492.

366. *Joslin, D. and Clements, D.* (1999). Squeaky wheel optimization. Journal of Artificial Intelligence Research, 10: 353–373.

367. *Kanji, G.* (2006). 100 Statistical Tests. Sage Publications.

368. *Karaboga, D. and Akay, B.* (2009). A comparative study of artificial bee colony algorithm. Applied Mathematics and Computation, 214(1): 108–132.

369. *Karaboga, D. and Basturk, B.* (2007). A powerful and efficient agorithm for numerical function optimization: Artificial bee colony (ABC) algorithm. Journal of Global Optimization, 39(3): 459–471.

370. *Karaboga, D. and Basturk, B.* (2008). On the performance of artificial bee colony (ABC) algorithm. Applied Soft Computing, 8(1): 687–697.

371. *Karaboga, D., Gorkemli, B., Ozturk, C., and Karaboga, N.* (2013). A comprehensive survey: Artificial bee colony (ABC) algorithm and applications. Artificial Intelligence Review, in print.

372. *Kaveh, A. and Talatahari, S.* (2010). A novel heuristic optimization method: Charged system search. Acta Mechanica, 213(3–4): 267–289.

373. *Kazarlis, S. and Petridis, V.* (1998). Varying fitness functions in genetic algorithms: Studying the rate of increase of the dynamic penalty terms. In *Eiben, A., Bäck, T., Schoenauer, M., and Schwefel, H.-P., editors*, Parallel Problem Solving from Nature – PPSN V, pages 211–220. Springer.

374. *Keel, L. and Bhattacharyya, S.* (1997). Robust, fragile, or optimal? IEEE Transactions on Automatic Control, 42(8): 1098–1105.

375. *Kemeny, J., Snell, J., and Thompson, G.* (1974). Introduction to Finite Mathematics. Prentice-Hall.

376. *Kennedy, J.* (1998). Thinking is social: Experiments with the adaptive culture model. Journal of Conflict Resolution, 42(1): 56–76.

377. *Kennedy, J. and Eberhart, R.* (1997). A discrete binary version of the particle swarm algorithm. IEEE Conference on Systems, Man, and Cybernetics, Orlando, Florida, pages 4104–4109.

378. *Kennedy, J. and Eberhart, R., editors* (2001). Swarm Intelligence. Morgan Kaufmann.

379. *Kern, S., Miiller, S., Hansen, N., Biiche, D., Ocenasek, J., and Koumoutsakos, P.* (2004). Learning probability distributions in continuous evolu-

tionary algorithms – A comparative review. Natural Computing, 3(1): 77–112.

380. *Keynes, R., editor* (2001). Charles Darwin's Beagle diary. Cambridge University Press.
381. *Khare, V., Yao, X., and Deb, K.* (2003). Performance scaling of multi-objective evolutionary algorithms. In *Fonseca, C., Fleming, P., Zitzler, E., Thiele, L., and Deb, K., editors*, Evolutionary Multi-Criterion Optimization: Second International Conference, EMO 2003, pages 376–390. Springer.
382. *Khatib, W. and Fleming, P.* (1998). The stud GA: A mini revolution? In *Eiben, A., Bäck, T., Schoenauer, M., and Schwefel, H.-P., editors*, Parallel Problem Solving from Nature – PPSN V, pages 683–691. Springer.
383. *Kim, D.* (2006). Memory analysis and significance test for agent behaviours. Genetic and Evolutionary Computation Conference, Seattle, Washington, pages 151–158.
384. *Kim, H.-S. and Cho, S.-B.* (2000). Application of interactive genetic algorithm to fashion design. Engineering Applications of Artificial Intelligence, 13(6):635–644.
385. *Kinnear, K.* (1993). Evolving a sort: Lessons in genetic programming. International Conference on Neural Networks, San Francisco, California, pages 881–888.
386. *Kinnear, K.* (1994). Alternatives in automatic function definition: A comparison of performance. In Kinnear, K., editor, Advances in Genetic Programming, pages 119–141. MIT Press.
387. *Kirk, D., editor* (2004). Optimal Control Theory. Dover.
388. *Kirkpatrick, S., Gelatt, C., and Vecchi, M.* (1983). Optimization by simmulated annealing. Science, 220(4598): 671–680.
389. *Kjellström, G.* (1969). Network optimization by random variation of component values. Ericsson Technics, 25(3): 133–151.
390. *Kleidon, A.* (2004). Amazonian biogeography as a test for Gaia. In Schneider, S., Miller, J., Crist, E., and Boston, P., editors, Scientists Debate Gaia, pages 291–296. MIT Press.
391. *Knowles, J.* (2005). ParEGO: A hybrid algorithm with on–line landscape approximation for expensive multiobjective optimization problems. IEEE Transactions on Evolutionary Computation, 10(1): 50–66.
392. *Knowles, J. and Corne, D.* (2001). Approximating the nondominated front using the Pareto archived evolution strategy. Evolutionary Computation, 8(2): 149–172.

393. *Knowles, J. and Nakayama, H.* (2008). Meta-modeling in multiobjective optimization. In *Branke, J., Deb, K., Miettinen, K., and Slowinski, R., editors*, Multiobjective Optimization, pages 245–284. Springer.

394. *Konak, A., Coit, D., and Smith, A.* (2006). Multi-objective optimization using genetic algorithms: A tutorial. Reliability Engineering and System Safety, 91(9): 992–1007.

395. *Kondoh, M.* (2006). Does foraging adaptation create the positive complexity-stability relationship in realistic food-web structure? Journal of Theoretical Biology, 238(3): 646–651.

396. *Koza, J., editor* (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection. The MIT Press.

397. *Koza, J., editor* (1994). Genetic Programming II: Automatic Discovery of Reusable Programs. The MIT Press.

398. *Koza, J.* (1997). Classifying protein segments as transmembrane domains using genetic programming and architecture-altering operations. In *Bäck, T., Fogel, D., and Michalewicz, Z., editors*, Handbook of Evolutionary Computation, pages G6.1: 1–5. Oxford University Press.

399. *Koza, J.* (2010). Human-competitive results produced by genetic programming. Genetic Programming and Evolvable Machines, 11(3–4): 251–284.

400. *Koza, J., Al-Sakran, L., and Jones, L.* (2008). Automated ab initio synthesis of complete designs of four patented optical lens systems by means of genetic programming. Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 22(3):249–273.

401. *Koza, J., Bennett, F., Andre, D., and Keane, M., editors* (1999). Genetic Programming Ill: Darwinian Invention and Problem Solving. Morgan Kaufmann.

402. *Koza, J., Keane, M., Streeter, M., Mydlowec, W., Yu, J., and Lanza, G., editors* (2005). Genetic Programming IV: Routine Human–Competitive Machine Intelligence. The MIT Press.

403. *Koziel, S. and Michalewicz, Z.* (1998). A decoder-based evolutionary algorithm for constrained parameter optimization problems. In *Eiben, A., Bäck, T., Schoenauer, M., and Schwefel, H.-P., editors*, Parallel Problem Solving from Nature – PPSN V, pages 231–240. Springer.

404. *Koziel, S. and Michalewicz, Z.* (1999). Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. Evolutionary Computation, 7(1): 19–44.

405. *Krause, J. and Ruxton, G., editors* (2002). Living in Groups. Oxford University Press.

406. *Krige, D.* (1951). A statistical approach to some basic mine valuation problems on the Witwatersrand. Journal of the Chemical, Metallurgical and Mining Society of South Africa, 52(6): 119–139.

407. *Krishnanand, K. and Ghose, D.* (2009). Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions. Swarm Intelligence, 3(2): 87–124.

408. *Krogh, A.* (2008). What are artificial neural networks? Nature Biotechnology, 6(2): 195–197.

409. *Kursawe, F.* (1991). A variant of evolution strategies for vector optimization. In *Schwefel, H.-P. and Manner, R., editors*, Parallel Problem Solving from Nature – PPSN I, pages 193–197. Springer.

410. *Kvasnicka, V., Pelikan, M., and Pospichal, J.* (1996). Hill climbing with learning (an abstraction of genetic algorithm). Neural Network World, 6(5): 773–796.

411. *Lam, A. and Li, V.* (2010). Chemical-reaction-inspired metaheuristic for optimization. IEEE Transactions on Evolutionary Computation, 14(3): 381–399.

412. *Lampinen, J.* (2002). A constraint handling approach for the differential evolution algorithm. IEEE Congress on Evolutionary Computation, Honolulu, Hawaii, pages 1468–1473.

413. *Langdon, W.* (2000). Size fair and homologous tree genetic programming crossovers. Genetic Programming and Evolvable Machines, 1(1–2): 95–119.

414. *Langdon, W. and Poli, R., editors* (2002). Foundations of Genetic Programming. Springer.

415. *Larrañaga, P.* (2002). A review on estimation of distribution algorithms. In *Larrañaga, P. and Lozano, J., editors*, Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation, pages 57–100. Kluwer Academic Publishers.

416. *Larrañaga, P., Etxeberria, R., Lozano, J., and Peña, J.* (1999a). Optimization by learning and simulation of Bayesian and Gaussian networks. Technical report, University of the Basque Country. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.1895.

417. *Larrañaga, P., Etxeberria, R., Lozano, J., and Peña, J.* (2000). Combinatorial optimization by learning and simulation of Bayesian networks.

Sixteenth Conference on Uncertainty in Artificial Intelligence, Stanford, California, pages 343–352.

418. *Larrañaga, P., Karshenas, H., Bielza, C., and Santana, R.* (2012). A review on probabilistic graphical models in evolutionary computation. Journal of Heuristics, 18(5): 795–819.
419. *Larrañaga, P., Kuijpers, C., Murga, R., Inza, I., and Dizdarevic, S.* (1999b). Genetic algorithms for the travelling salesman problem: A review of representations and operators. Artificial Intelligence Review, 13(2): 129–170.
420. *Larrañaga, P. and Lozano, J., editors* (2002). Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation. Kluwer Academic Publishers.
421. *Latané, B., Nowak, A., and Liu, J.* (1994). Measuring emergent social phenomena: Dynamism, polarization, and clustering as order parameters of social systems. Behavioral Science, 39(1): 1–24.
422. *Lattimore, T. and Hutter, M.* (2011). No free lunch versus Occam's razor in supervised learning. Solomonoff 85th Memorial Conference, Melbourne, Australia.
423. *Laumanns, M., Thiele, L., and Zitzler, E.* (2003). Running time analysis of evolutionary agorithms on vector-valued pseudo–Boolean functions. IEEE Transactions on Evolutionary Computation, 8(2): 170–182.
424. *Lawler, E., Lenstra, J., Rinnooy Kan, A., and Shmoys, D., editors* (1985). The Traveling Salesman Problem. John Wiley & Sons.
425. *Le Riche, R., Knopf–Lenoir, C., and Haftka, R.* (1995). A segregated genetic algorithm for constrained structural optimization. International Conference on Genetic Algorithms, Pittsburgh, Pennsylvania, pages 558–565.
426. *Lee, K. and Geem, Z.* (2006). A new meta-heuristic algorithm for continuous engineering optimization: Harmony search theory and practice. Computer Methods in Applied Mechanics and Engineering, 194(36–38): 3902–3933.
427. *Leguizamón, G. and Coello Coello, C.* (2009). Boundary search for constrained numerical optimization problems. In Mezura-Montes, E., editor, Constraint-Handling in Evolutionary Optimization, pages 25–49. Springer.
428. *Lehman, J. and Stanley, K.* (2011). Abandoning objectives: Evolution through the search for novelty alone. Evolutionary Computation, 19(2): 189–223.

429. *Lenton, T.* (1998). Gaia and natural selection. Nature, 394(6692): 439–447.

430. *Li, C. and Yang, S.* (2008). A generalized approach to construct benchmark problems for dynamic optimization. In Li, X., editor, Simulated Evolution and Leaming, pages 391–400. Springer.

431. *Li, C., Yang, S., Nguyen, T., Yu, E., Yao, X., Jin, Y., Beyer, H.-G., and Suganthan, P.* (2008). Benchmark generator for CEC'2009 competition on dynamic optimization. Technical report. www.ntu.edu.sg/home/EPNSugan/index_files/cec-benchmarking.htm.

432. *Li, X., Shao, Z., and Qian, J.* (2003). An optimizing method based on autonomous animats: Fish-swarm algorithm. Systems Engineering – Theory f'3 Practice, 22(11): 32–38.

433. *Li, Y., Zhang, S., and Zeng, X.* (2009). Research of multi-population agent genetic algorithm for feature selection. Erpert Systems with Applications, 36(9): 11570–11581.

434. *Liang, J., Runarsson, T., Mezura-Montes, E., Clerc, M., Suganthan, P., Coello Coello, C., and Deb, K.* (2006). Problem definitions and evaluation criteria for the CEC 2006 special session on constrained real-parameter optimization. Technical report. www.ntu.edu.sg/home/EPNSugan/index_files/cec-benchmarking.htm.

435. *Liang, J., Suganthan, P., and Deb, K.* (2005). Novel composition test functions for numerical global optimization. Swarm Intelligence Symposium, Pasadena, California, pages 68–75.

436. *Lim, D., Jin, Y., Ong, Y.-S., and Sendhoff, B.* (2010). Generalizing surrogate-assisted evolutionary computation. IEEE Transactions on Evolutionary Computation, 14(3): 329–355.

437. *Lim, D., Ong, Y.-S., Jin, Y., Sendhoff, B., and Lee, B.-S.* (2007). Efficient hierarchical parallel genetic algorithms using grid computing. Future Generation Computer Systems, 23(4): 658–670.

438. *Lima, C. and Lobo, F.* (2004). Parameter-less optimization with the extended compact genetic algorithm and iterated local search. Genetic and Evolutionary Computation Conference, Seattle, Washington, pages 1328–1339.

439. *Lima, S.* (1995). Back to the basics of anti–predatory vigilance: The group-size effect. Animal Behaviour, 49(1): 11–20.

440. *Lis, J. and Eiben, A.* (1997). A multi-sexual genetic algorithm for multiobjective optimization. IEEE International Conference on Evolutionary Computation, Indianapolis, Indiana, pages 59–64.

441. *Löbbing, M. and Wegener, I.* (1995). The number of knight's tours equals 33,439,123,484,294 – counting with binary decision diagrams. Electronic Journal of Combinatorics, 3(1):5.

442. *Lohn, J., Hornby, G., and Linden, D.* (2004). An evolved antenna for deployment on NASA's space technology 5 mission. In *O'Reilly, U.-M., Riolo, R., Yu, G., and Worzel, W., editors*, Genetic Programming Theory and Practice II, pages 301–315. Kluwer Academic Publishers.

443. *Lomolino, M.* (2000a). A call for a new paradigm of island biogeography. Global Ecology and Biogeography, 9(1): 1–6.

444. *Lomolino, M.* (2000b). A species-based theory of insular zoogeography. Global Ecology and Biogeography, 9(1): 39–58.

445. *López Jaimes, A., Coello Coello, C., and Urías Barrientos, J.* (2009). Online objective reduction to deal with many-objective problems. 5th International Conference on Evolutionary Multi-Criterion Optimization, Nantes, France, pages 423–437.

446. *Lovelock, J.* (1990). Hands up for the Gaia hypothesis. Nature, 344(6262): 100–102.

447. *Lovelock, J., editor* (1995). Gaia. Oxford University Press.

448. *Lozano, J., Larrañaga, P., Inza, I., and Bengoetxea, E., editors* (2006). Towards a New Evolutionary Computation: Advances on Estimation of Distribution Algorithms. Springer.

449. *Lukasik, S. and Żak, S.* (2009). Firefly algorithm for continuous constrained optimization tasks. 1st International Conference on Computational Collective Intelligence, Wroclaw, Poland, pages 97–106.

450. *Lundy, M. and Mees, A.* (1986). Convergence of an annealing algorithm. Mathematical Programming, 34(1): 111–124.

451. *Ma, H.* (2010). An analysis of the equilibrium of migration models for biogeography-based optimization. Information Sciences, 180(18): 3444–3464.

452. *Ma, H., Ni, S., and Sun, M.* (2009). Equilibrium species counts and migration model tradeoffs for biogeography-based optimization. IEEE Conference on Decision and Control, Shanghai, China, pages 3306–3310.

453. *Ma, H. and Simon, D.* (2010). Biogeography-based optimization with blended migration for constrained optimization problems. Genetic and Evolutionary Computation Conference, Portland, Oregon, pages 417–418.

454. *Ma, H. and Simon, D.* (2011a). Analysis of migration models of biogeography-based optimization using markov theory. Engineering Applications of Artificial Intelligence, 24(6): 1052–1060.

455. *Ma, H. and Simon, D.* (2011b). Blended biogeography-based optimization for constrained optimization. Engineering Applications of Artificial Intelligence, 24(3): 517–525.

456. *Ma, H. and Simon, D.* (2013). Variations ofbiogeography-based optimization and markov analysis. Information Sciences, 220: 492–506.

457. *Ma, H., Simon, D., and Fei, M.* (2013). On the statistical mechanics approximation of biogeography-based optimization. Submitted for publication.

458. *MacArthur, R.* (1955). Fluctuations of animal populations and a measure of community stability. Ecology, 36(3): 533–536.

459. *MacArthur, R. and Wilson, E.* (1963). An equilibrium theory of insular zoogeography. Evolution, 17(4): 373–387.

460. *MacArthur, R. and Wilson, E.* (1967). The Theory of Island Biogeography. Princeton University Press.

461. *Mahfoud, S.* (1992). Crowding and preselection revisited. Technical report, Illinois Genetic Algorithms Laboratory, University of Illinois. IlliGAL Report No. 92004.

462. *Mahfoud, S.* (1995a). A comparison of parallel and sequential niching methods. International Conference on Genetic Algorithms, Pittsburgh, Pennsylvania, pages 136–143.

463. *Mahfoud, S.* (1995b). Niching methods for genetic algorithms. Technical report, Illinois Genetic Algorithms Laboratory, University of Illinois. IlliGAL Report No. 95001.

464. *Mahnig, T. and Mühlenbein, H.* (2000). Mathematical analysis of optimization methods using search distributions. Genetic and Evolutionary Computation Conference, Las Vegas, Nevada, pages 205–208.

465. *Malisia, A.* (2008). Improving the exploration ability of ant–based algorithms. In Tizhoosh, H. and Ventresca, M., editors, Oppositional Concepts in Computational Intelligence, pages 121–142. Springer.

466. *Mallipeddi, R. and Suganthan, P.* (2010). Problem definitions and evaluation criteria for the CEC 2010 competition on constrained real–parameter optimization. Technical report, Nanyang Technological University. www.ntu.edu.sg/home/EPNSugan/index_files/cec-benchmarking.htm.

467. *Maniezzo, V., Gambardella, L., and de Luigi, F.* (2004). Ant colony optimization. In Onwnboln, G. and Babu, R., editors, New Optimization Techniques in Engineering, pages 101–122. Springer.

468. *Margulis, L.* (1996). Gaia is a tough bitch. In Brockman, J., editor, The Third Culture: Beyond the Scientific Revolution, pages 129–151. Touchstone.

469. *Marriott, K. and Stuckey, P.* (1998). Programming with Constraints: An Introduction. The MIT Press.

470. *Mavrovouniotis, M. and Yang, S.* (2011). Ant colony optimization with immigrants schemes in dynamic environments. In *Schaefer, R., Cotta, C., Kolodziej, J., and Rudolph, G., editors*, Parallel Problem Solving from Nature – PPSN XI, pages 371–380. Springer.

471. *May, R.* (1973). Stability and Complexity in Model Ecosystems. Princeton University Press.

472. *McCann, K.* (2000). The diversity-stability debate. Nature, 405(6783): 228–233.

473. *McConaghy, T., Palmers, P., Gielen, G., and Steyaert, M.* (2008). Genetic programming with reuse of known designs for industrially scalable, novel circuit design. In *Riolo, R., Soule, T., and Worzel, B., editors*, Genetic Programming Theory and Practice V, pages 159–184. Springer.

474. *McGill, R., Tukey, J., and Larsen, W.* (1978). Variations of box plots. The American Statistician, 32(1): 12–16.

475. *McNab, B.* (2002). The Physiological Ecology of Vertebrates. Cornell University.

476. *McTavish, T. and Restrepo, D.* (2008). Evolving solutions: The genetic algorithm and evolution strategies for finding optimal parameters. In *Smolinski, T., Milanova, M., and Hassanien, A., editors*, Applications of Computational Intelligence in Biology, pages 55–78. Springer.

477. *Mehrabian, R. and Lucas, C.* (2006). A novel numerical optimization algorithm inspired from weed colonization. Ecological Informatics, 1(4): 355–366.

478. *Melab, N., Cahan, S., and Taibi, E.-G.* (2006). Grid computing for parallel bioinspired algorithms. Journal of Parallel and Distributed Computing, 66(8): 1052–1061.

479. *Mendes, R., Kennedy, J., and Neves, J.* (2004). The fully informed particle swarm: Simpler, maybe better. IEEE Transactions on Evolutionary Computation, 8(3): 204–210.

480. *Metropolis, N.* (1987). The beginning of the Monte Carlo method. Los Alamos Science, 15: 125–130.

481. *Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E.* (1953). Equations of state calculations by fast computing machines. The Journal of Chemical Physics, 21(6): 1087–1092.

482. *Meuleau, N., Peshkin, L., Kim, K.-E., and Kaelbling, L.* (1999). Learning finite-state controllers for partially observable environments. Conference on Uncertainty in Artificial Intelligence, Stockholm, Sweden, pages 427–436.

483. *Meuleau02, N. and Dorigo, M.* (2002). Ant colony optimization and stochastic gradient descent. Artificial Life, 8(2): 103–121.

484. *Mezura-Montes, E. and Coello Coello, C.* (2005). A simple multimembered evolution strategy to solve constrained optimization problems. IEEE Transactions on Evolutionary Computation, 9(1): 1–17.

485. *Mezura-Montes, E. and Coello Coello, C.* (2008). Constrained optimization via multiobjective evolutionary algorithms. In *Knowles, J., Corne, D., and Deb, K., editors*, Multiobjective Problem Solving from Nature, pages 53–75. Springer.

486. *Mezura-Montes, E. and Palomeque-Oritiz, A.* (2009). Parameter control in differential evolution for constrained optimization. IEEE Congress on Evolutionary Computation, Trondheim, Norway, pages 1375–1382.

487. *Mezura-Montes, E., Reyes-Sierra, M., and Coello Coello, C.* (2008). Multi-objective optimization using differential evolution: A survey of the state-of-the-art. In Chakraborty, U., editor, Advances in Differential Evolution, pages 173–196. Springer.

488. *Michalewicz, Z.* (1996). Genetic Algorithms + Data Structures = Evolution Programs. Springer.

489. *Michalewicz, Z. and Attia, N.* (1994). Evolutionary optimization of constrained problems. Third Annual Conference on Evolutionary Programming, San Diego, California, pages 98–108.

490. *Michalewicz, Z., Dasgupta, D., Riche, R. L., and Schoenauer, M.* (1996). Evolutionary algorithms for constrained engineering problems. Computers & Industrial Engineering, 30(4): 851–870.

491. *Michalewicz, Z. and Janikow, C.* (1991). Handling constraints in genetic algorithms. International Conference on Genetic Algorithms, Breckenridge, Colorado, pages 151–157.

492. *Michalewicz, Z. and Nazhiyath, G.* (1995). Genocop III: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. IEEE Conference on Evolutionary Computation, Perth, Western Australia, pages 647–651.

493. *Michalewicz, Z. and Schoenauer, M.* (1996). Evolutionary algorithms for constrained parameter optimization problems. Evolutionary Computation, 4(1): 1–32.

494. *Michiels, W., Aarts, E., and Korst, J.* (2007). Theoretical Aspects of Local Search. Springer.

495. *Milinski, H. and Heller, R.* (1978). Influence of a predator on the optimal foraging behavior of sticklebacks. Nature, 275(5681): 642–644.

496. *Miller, J. and Smith, S.* (2006). Redundancy and computational efficiency in Cartesian genetic programming. IEEE Transactions on Evolutionary Computation, 10(2): 167–174. 2006.

497. *Mitchell, M.* (1998). An Introduction to Genetic Algorithms. The MIT Press.

498. *Mitzenmacher, M. and Upfal, E.* (2005). Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press.

499. *Morales, A. and Quezada, C.* (1998). A univeral eclectic genetic algorithm for constrained optimization. Sixth European Congress on Intelligent Techniques and Soft Computing, Aachen, Germany, pages 518–522.

500. *Morrison, R.* (2004). Designing Evolutionary Algorithms for Dynamic Environments. Springer.

501. *Mühlenbein, H., Mahnig, T., and Ochoa, A.* (1999). Schemata, distributions and graphical models in evolutionary optimization. Journal of Heuristics, 5(2): 215–247.

502. *Mühlenbein, H. and Paa,B, G.* (1996). From recombination of genes to the estimation of distributions: I. Binary parameters. In *Voigt, H.-M., Ebeling, W., Rechenberg, I., and Schwefel, H.-P., editors*, Parallel Problem Solving from Nature – PPSN IV, pages 178–187. Springer.

503. *Mühlenbein, H. and Schlierkamp-Voosen, D.* (1993). Predictive models for the breeder genetic algorithm: I. Continuous parameter optimization. Evolutionary Computation, 1(1): 25–49.

504. *Mühlenbein, H. and Schlierkamp-Voosen, D.* (1997). The equation for response to selection and its use for prediction. Evolutionary Computation, 5(3): 303–346.

505. *Mühlenbein, H. and Voigt, H.-M.* (1995). Gene pool recombination for the breeder genetic algorithm. First Metaheuristics International Conference, Breckenridge, Colorado, pages 19–25.

506. *Muller, S., Marchetta, J., Airaghi, S., and Kournoutsakos, P.* (2002). Optimization based on bacterial chemotaxis. IEEE Transactions on Evolutionary Computation, 6(1): 16–29.

507. *Munroe, E.* (1948). The Geographical Distribution of Butterflies in the West Indies. PhD thesis, Cornell University.

508. *Nemhauser, G. and Wolsey, L.* (1999). Integer and Combinatorial Optimization. John Wiley & Sons.

509. *Neri, F. and Tirronen, V.* (2010). Recent advances in differential evolution: A survey and experimental analysis. Artificial Intelligence Review, 33(1): 61–106.

510. *Neshat, M., Adeli, A., Sepidnam, G., Sargolzaei, M., and Toosi, A*. (2012). A review of artificial fish swarm optimization methods and applications. International Journal on Smart Sensing and Intelligent Systems, 5(1): 107–148.

511. *Neumann, F. and Witt, C.* (2009). Runtime analysis of a simple ant colony optimization algorithm. Algorithmica, 54(2): 243–255.

512. *Newton, M.* (2004). Savage Girls and Wild Boys. Picador.

513. *Nguyen, T., Yang, S., and Branke, J*. (2012). Evolutionary dynamic optimization: A survey of the state of the art. Swarm and Evolutionary Computation, 6: 1–24.

514. *Nguyen, T. and Yao, X.* (2009). Benchmarking and solving dynamic constrained problems. IEEE Congress on Evolutionary Computation, Trondheim, Norway, pages 690–697.

515. *Nierhaus, G.* (2010). Algorithmic Composition: Paradigms of Automated Music Generation. Springer.

516. *Niknam, T. and Amiri, B*. (2010). An efficient hybrid approach based on PSO, ACO and k–means for cluster analysis. Applied Soft Computing, 10(1): 183–197.

517. *Nix, A. and Vose, M*. (1992). Modeling genetic algorithms with Markov chains. Annals of Mathematics and Artificial Intelligence, 5(1): 79–88.

518. *Noel, M. and Jannett, T*. (2005). A new continuous optimization algorithm based on sociological models. American Control Conference, Portland, Oregon, pages 237–242.

519. *Noman, N. and Iba, H.* (2008). Accelerating differential evolution using an adaptive local search. IEEE Transactions on Evolutionary Computation, 12(1): 107–125.

520. *Nordin, P., Francone, F., and Banzhaf, W.* (1996). Explicitly defined intrans and destructive crossover in genetic programming. In Angeline, P. and Kinnear, K., editors, Advances in Genetic Programming: Volume 2, pages 111–134. The MIT Press.

521. *Noren, S., Biedenbach, G., Redfern, J., and Edwards, E.* (2008). Hitching a ride: The formation locomotion strategy of dolphin calves. Functional Ecology, 22(2): 278–283.

522. *Nourani, Y. and Andresen, B.* (1998). A comparison of simulated annealing cooling strategies. Journal of Physics A: Mathematical and General, 31(41): 8373–8385.

523. *Okubo, A. and Levin, S.* (2001). Diffusion and Ecological Problems. Springer.

524. *Oliver, I., Smith, D., and Holland, J.* (1987). A study of permutation crossover operators on the traveling salesman problem. International Conference on Genetic Algorithms, Cambridge, Massachusetts, pages 224–230.

525. *Omran, M.* (2008). Using opposition-based learning with particle swarm optimization and barebones differential evolution. In *Lazinica, A., editor*, Particle Swarm Optimization, pages 373–384. InTech.

526. *Omran, M., Engelbrecht, A., and Salman, A.* (2009). Bare bones differential evolution. European Journal of Operational Research, 196(1): 128–139.

527. *Omran, M. and Mahdavi, M.* (2008). Global-best harmony search. Applied Mathematics and Computation, 198(2): 643–656.

528. *Omran, M., Simon, D., and Clerc, M.* (2013). Linearized biogeography–based optimization. Submitted for publication.

529. *O'Neill, M. and Ryan, C.* (2003). Grammatical Evolution. Springer.

530. *Ong, Y., Nair, P., Keane, A., and Wong, K.* (2004). Surrogate-assisted evolutionary optimization frameworks for high-fidelity engineering design problems. In *Jin, Y., editor*, Knowledge Incorporation in Evolutionary Computation, pages 307–332. Springer.

531. *Ong, Y.-S., Krasnogor, N., and Ishibuchi, H.* (2007). Special issue on memetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics, 37(1): 2–5.

532. *Onwubolu, G. and Davendra, D., editors* (2009). Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization. Springer.

533. *Opitz, D. and Maclin, R.* (1999). Popular ensemble methods: An empirical study. Journal of Artificial Intelligence Research, 11: 169–198.

534. *O'Reilly, U. and Oppacher, F.* (1995). The troubling aspects of a building block hypothesis for genetic programming. In *Whitley, L. and Vose, M., editors*, Foundations of Genetic Algorithms, Volume 3, pages 73–88. Morgan Kaufmann.

535. *Orvosh, D. and Davis, L.* (1993). Shall we repair? Genetic algorithms, combinatorial optimization, and feasibility constraints. International Conference on Genetic Algorithms, Urbana-Champaign, Illinois, page 650.

536. *Otten, R. and van Ginneken, L.* (1989). The Annealing Algorithm. Kluwer Academic Publishers.

537. *Palmer, C. and Kershenbaum, A.* (1994). Representing trees in genetic algorithms. IEEE Conference on Evolutionary Computation, Orlando, Florida, pages 379–384.

538. *Pan, Q., Tasgetiren, M., and Liang, Y.* (2008). A discrete differential evolution algorithm for the permutation flowshop scheduling problem. Computers f3 Industrial Engineering, 55(4): 795–816.

539. Paquet, U. and Engelbrecht, A. (2003). A new particle swarm optimiser for linearly constrained optimisation. IEEE Congress on Evolutionary Computation, Canberra, Australia, pages 227–233.

540. *Pardalos, P. and Mavridou, T.* (1998). The graph coloring problem: A bibliographic survey. In *Zhu, D.-Z. and Pardalos, P., editors*, Handbook of Combinatorial Optimization, pages 331–395. Kluwer Academic Publishers.

541. *Paredis, J.* (2000). Coevolutionary algorithms. In Bäck, T., Fogel, D., and Michalewicz, Z., editors, Evolutionary Computation 2, pages 224–238. Institute of Physics.

542. *Pareto, V.* (1896). Cours d'Economie Politique. Guillaumin.

543. *Parks, G. and Miller, I.* (1998). Selective breeding in a multiobjective genetic algorithm. In *Eiben, A., Bäck, T., Schoenauer, M., and Schwefel, H.-P., editors*, Parallel Problem Solving from Nature – PPSN V, pages 250–259. Springer.

544. *Passino, K.* (2002). Biomimicry of bacterial foraging. IEEE Control Systems Magazine, 22(3): 52–67.

545. *Paul, T. and Iba, H.* (2003). Optimization in continuous domain by real-coded estimation of distribution algorithm. In *Abraham, A., Köppen, M., and Franke, K., editors*, Design and Application of Hybrid Intelligent Systems, pages 262–271. IOS Press.

546. *Peña, J., Robles, V., Larrañaga, P., Herves, V., Rosales, F., and Perez, M.* (2004). GA-EDA: Hybrid evolutionary algorithm using genetic and estimation of distribution algorithms. In *Orchard, B., Yang, C., and Ali, M., editors*, Innovations in Applied Artificial Intelligence, pages 361–371. Springer.

547. *Pedersen, M.* (2010). Good parameters for particle swarm optimization. Technical report, Hvass Laboratories. www.hvass-labs.org.

548. *Pedersen, M. and Chipperfield, A.* (2010). Simplifying particle swarm optimization. Applied Soft Computing, 10(2): 618–628.

549. *Pelikan, M.* (2005). Hierarchical Bayesian Optimization Algorithm. Springer.

550. *Pelikan, M., Goldberg, D., and Cantu-Paz, E.* (1999). BOA: The Bayesian optimization algorithm. Genetic and Evolutionary Computation Conference, Orlando, Florida, pages 525–532.

551. *Pelikan, M., Goldberg, D., and Lobo, F.* (2002). A survey ofoptimization by building and using probabilistic models. Computational Optimization and Applications, 21(1): 5–20.

552. *Pelikan, M. and Mühlenbein, H.* (1998). The bivariate marginal distribution algorithm. In *Benitez, J., Cordon, 0., Hoffmann, F., and Roy, R., editors*, Advances in Soft Computing: Engineering Design and Manufacturing, pages 521–535. Springer.

553. *Pelikan, M. and Sastry, K.* (2004). Fitness inheritance in the Bayesian optimization algorithm. Genetic and Evolutionary Computation Conference, Seattle, Washington, pages 48–59.

554. *Petroski, H.* (1992). To Engineer Is Human: The Role of Failure in Successful Design. Vintage.

555. *Pétrowski, A.* (1996). A clearing procedure as a niching method for genetic algorithms. IEEE Conference on Evolutionary Computation, Nagoya, Japan, pages 798–803.

556. *Pham, D., Ghanbarzadeh, A., Koç, E., Otri, S., Rahim, S., and Zaidi, M.* (2006). The bees algorithm – A novel tool for complex optimisation problems. 2nd International Virtual Conference on Intelligent Production Machines and Systems, pages 454–459.

557. *Pincus, M.* (1968a). A closed form solution of certain programming problems. Operations Research, 16(3): 690–694.

558. *Pincus, M.* (1968b). A Monte Carlo method for the approximate solution of certain types of constrained optimization problems. Operations Research, 18(6): 1225–1228.

559. *Pitcher, T. and Parrish, J.* (1993). Functions of shoaling behaviour in teleosts. In Pitcher, T., editor, Behaviour of Teleost Fishes, pages 363–439. Chapman & Hall.

560. *Poli, R.* (2003). A simple but theoretically-motivated method to control bloat in genetic programming. Sixth European Conference on Genetic Programming, Essex, England, pages 211–223.

561. *Poli, R.* (2008). Dynamics and stability of the sampling distribution of particle swarm optimisers via moment analysis. Journal of Artificial Evolution and Applications, 2008. Article ID 761459, 10 pages, doi:10.1155/2008/761459.

562. *Poli, R., Kennedy, J., and Blackwell, T.* (2007). Particle swarm optimization: An overview. Swarm Intelligence, 1(1):33–57.

563. *Poli, R., Langdon, W., and McPhee, N.* (2008). A Field Guide to Genetic Programming. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk.

564. *Poli, R., McPhee, N., and Rowe, J.* (2004). Exact schema theory and Markov chain models for genetic programming and variable-length genetic algorithms with homologous crossover. Genetic Programming and Evolvable Machines, 5(1): 31–70.

565. *Poli, R., Rowe, J., and McPhee, N.* (2001). Markov chain models for GP and variablelength GAs with homologous crossover. Genetic and Evolutionary Computation Conference, San Francisco, California, pages 112–119.

566. *Poundstone, W.* (1993). Prisoner's Dilemma. Anchor.

567. *Powell, D. and Skolnick, M.* (1993). Using genetic algorithms in engineering design optimization with non-linear constraints. International Conference on Genetic Algorithms, Urbana-Champaign, Illinois, pages 424–431.

568. *Preble, S., Lipson, M., and Lipson, H.* (2005). Two-dimensional photonic crystals designed by evolutionary algorithms. Applied Physics Letters, 86(6): 061111.

569. *Price, K.* (1997). Differential evolution versus the functions of the 2nd ICEO. IEEE Conference on Evolutionary Computation, Indianapolis, Indiana, pages 153–157.

570. *Price, K.* (2013). Differential evolution. In *Zelinka, I., Snasel, V., and Abraham, A., editors*, Handbook of Optimization. ebooks.com.

571. *Price, K. and Storn, R.* (1997). Differential evolution: Numerical optimization made easy. Dr. Dobb's Journal, pages 18–24.

572. *Price, K., Storn, R., and Lampinen, J.* (2005). Differential Evolution. Springer.

573. *PSC* (2012). Particle Swarm Central. www.particleswarm.info.

574. *Păun, G.* (2003). Membrane computing. In *Lingas, A. and Nilsson, B., editors*, Fundamentals of Computation Theory, pages 177–220. Springer.

575. *Pyke, G.* (1978). Optimal foraging in bumblebees and coevolution with their plants. Oecologia, 36(3): 281–293.

576. *Qin, A., Huang, V., and Suganthan, P.* (2009). Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Transactions on Evolutionary Computation, 13(2): 39–417.

577. *Qing, A.* (2009). Differential Evolution: Fundamentals and Applications in Electrical Engineering. John Wiley & Sons.

578. *Quammen, D.* (1997). The Song of the Dodo: Island Biogeography in an Age of Extinction. Scribner.

579. *Quijano, N., Passino, K., and Andrews, B.* (2006). Foraging theory for multizone temperature control. IEEE Computational Intelligence Magazine, 1(4): 18–27.

580. *Rabanal, P., Rodríguez, I., and Rubio, F.* (2007). Using river formation dynamics to design heuristic algorithms. In *Aki, S., Calude, C., Dinneen, M., Rozenberg, G., and Wareham, H., editors*, Unconventional Computation, pages 163–177. Springer.

581. *Radcliffe, N. and Surry, P.* (1995). Fundamental limitations on search algorithms: Evolutionary computing in perspective. In *Van Leeuwen, J., editor*, Computer Science Today: Recent Trends and Developments (Lecture Notes in Computer Science, No. 1000), pages 275–291. Springer-Verlag.

582. *Rahnamayan, S., Tizhoosh, H., and Salama, M.* (2008). Opposition–based differential evolution. IEEE Transactions on Evolutionary Computation, 12(1): 64–79.

583. *Rao, R. and Patel, V.* (2012). An elitist teaching-learning-based optimization algorithm for solving complex constrained optimization problems. International Journal of Industrial Engineering Computations, 3(4): 535–560.

584. *Rao, R. and Savsani, V.* (2012). Mechanical Design Optimization Using Advanced Optimization Techniques. Springer.

585. *Rao, R., Savsani, V., and Vakharia, D.* (2011). Teaching-learning-based optimization: A novel method for constrained mechanical design optimization problems. Computer-Aided Design, 43(3): 303–315.

586. *Rao, R., Savsani, V., and Vakharia, D*. (2012). Teaching-learning-based optimization: A novel optimization method for continuous non-linear large scale problems. Information Sciences, 183(1): 1–15.

587. *Rashedi, E., Nezamabadi-pour, H., and Saryazdi, S*. (2009). GSA: A gravitational search algorithm. Information Sciences, 179(13): 2232–2248.

588. *Rashedi, E., Nezamabadi-pour, H., and Saryazdi, S*. (2010). BGSA: Binary gravitational search algorithm. Natural Computing, 9(3): 727–745.

589. *Rasheed, K. and Hirsh, H.* (2000). Informed operators: Speeding up genetic-algorithmbased design optimization using reduced models. Genetic and Evolutionary Computation Conference, Las Vegas, Nevada, pages 628–635.

590. *Rashid, M. and Baig, A*. (2010). Improved opposition-based PSO for feedforward neural network training. International Conference on Information Science and Applications, Seoul, Korea, pages 1–6.

591. *Rastrigin, L.* (1974). Extremal Control Systems. Nauka. In Russian.

592. *Rawlins, G., editor* (1991). Foundations of Genetic Algorithms. Morgan Kaufmann Publishers.

593. *Ray, T., Isaacs, A., and Smith, W*. (2009a). A memetic algorithm for dynamic multiobjective optimization. In *Goh, C.-K., Ong, Y.-S., and Tan, K., editors*, Multi-Objective Memetic Algorithms, pages 353–367. Springer.

594. *Ray, T. and Liew, K*. (2003). Society and civilization: An optimization algorithm based on the simulation of social behavior. IEEE Transactions on Evolutionary Computation, 7(4): 386–396.

595. *Ray, T., Singh, H., Isaacs, A., and Smith, W.* (2009b). Infeasibility driven evolutionary algorithm for constrained optimization. In *Mezura-Montes, E., editor*, Constraint-Handling in Evolutionary Optimization, pages 145–165. Springer.

596. *Rechenberg, I.* (1973). Evolutionsstrategie – Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution. Frommann-Holzboog. The English translation of the title is Evolution strategy – Optimization of Technical Systems according to Principles of Biological Evolution.

597. *Rechenberg, I.* (1998). Cybernetic solution path of an experimental problem. In Fogel, D., editor, Evolutionary Computation: The Fossil Record, pages 301–310. Wiley-IEEE Press. First published in 1964.

598. *Reeves, C.* (1993). Modern Heuristic Techniques for Combinatorial Problems. John Wiley & Sons.

599. *Reeves, C. and Rowe, J.* (2003). Genetic Algorithms: Principles and Perspectives. Kluwer Academic Publishers.

600. *Reinelt, G.* (2008). TSPLIB. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95.

601. *Reynolds, R.* (1994). An introduction to cultural algorithms. Third Annual Conference on Evolutionary Computing, Madison, Wisconsin, pages 131–139.

602. *Reynolds, R.* (1999). Cultural algorithms: Theory and applications. In *Corne, D., Dorigo, M., Glover, F., Dasgupta, D., Moscato, P., Poli, R., and Price, K., editors*, New Ideas in Optimization, pages 367–378. McGraw-Hill.

603. *Reynolds, R., Ashlock, D., Yannakakis, G., Togelius, J., and Preuss, M.* (2011). Tutorials: Cultural algorithms: Incorporating social intelligence into virtual worlds. IEEE Conference on Computational Intelligence and Games, Seoul, South Korea, pages J1–J5.

604. *Reynolds, R. and Chung, C.* (1997). Knowledge-based self-adaptation in evolutionary programming using cultural algorithms. IEEE International Conference on Evolutionary Computation, Indianapolis, Indiana, pages 71–76.

605. *Ritscher, T., Helwig, S., and Wanka, R.* (2010). Design and experimental evaluation of multiple adaptation layers in self-optimizing particle swarm optimization. IEEE Congress on Evolutionary Computation, Barcelona, Spain, pages 1–8.

606. *Ritzel, B., Eheart, J., and Ranjithan, S.* (1995). Using genetic algorithms to solve a multiple objective groundwater pollution containment problem. Water Resources Research, 30(5): 1589–1603.

607. *Robert, C. and Casella, G.* (2010). Monte Carlo Statistical Methods. Springer.

608. *Ronald, S.* (1998). Duplicate genotypes in a genetic algorithm. IEEE World Congress on Computational Intelligence, Anchorage, Alaska, pages 793–798.

609. *Ros, R. and Hansen, N.* (2008). A simple modification in CMA-ES achieving linear time and space complexity. In *Rudolp, G., Jansen, T., Lucas, S., Poloni, C., and Beume, N., editors*, Parallel Problem Solving from Nature – PPSN X, pages 296–305. Springer.

610. *Rosca, J.* (1997). Analysis of complexity drift in genetic programming. Second Annual Conference on Genetic Programming, Palo Alto, California, pages 286–294.

611. *Rosenberg, R.* (1967). Simulation of genetic populations with biochemical properties. PhD thesis, University of Michigan.

612. *Rosenbrock, H.* (1960). An automatic method for finding the greatest or least value of a function. The Computer Journal, 3(3): 175–184.

613. *Rosenkrantz, D., Stearns, R., and Lewis, P.* (1977). An analysis of several heuristics for the traveling salesman problem. SIAM Journal on Computing, 6(3): 563–581.

614. *Ross, T.* (2010). Fuzzy Logic with Engineering Applications. John Wiley & Sons, 3rd edition.

615. *Rossi, F., van Beek, P., and Walsh, T.* (2006). Handbook of Constraint Programming. Elsevier.

616. *Rothlauf, F. and Goldberg, D.* (2003). Redundant representations in evolutionary computation. Evolutionary Computation, 11(4): 381–415.

617. *Rubinstein, A.* (1986). Finite automata play the repeated prisoner's dilemma. Journal of Economic Theory, 39(1): 83–96.

618. *Rudlof, S. and Köppen, M.* (1996). Stochastic hill climbing by vectors of normal distributions. First Online Workshop on Soft Computing, Nagoya, Japan, pages 60–70.

619. *Rudolph, G.* (1992). Parallel approaches to stochastic global optimization. In Joosen, W. and Milgrom, E., editors, Parallel Computing: Prom Theory to Sound Practice, pages 256–267. IOS Press.

620. *Rudolph, G. and Agapie, A.* (2000). Convergence properties of some multi-objective evolutionary algorithms. IEEE Congress on Evolutionary Computation, San Diego, California, pages 1010–1016.

621. *Rudolph, G. and Schwefel, H.-P.* (2008). Simulated evolution under multiple criteria conditions revisited. IEEE World Congress on Computational Intelligence, Hong Kong, pages 249–261.

622. *Runarsson, T. and Yao, X.* (2000). Stochastic ranking for constrained evolutionary optimization. IEEE Transactions on Evolutionary Computation, 4(3): 284–294.

623. *Sakawa, M.* (2002). Genetic Algorithms and Fuzzy Multiobjective Optimization. Springer.

624. *Salkind, N.* (2007). Statistics for People Who {Think They) Hate Statistics. Sage Publications.

625. *Salomon, R.* (1996). Reevaluating genetic algorithm performance under coordinate rotation of benchmark functions. BioSystems, 39(3): 263–278.

626. *Salustowicz, R. and Schmidhuber, J.* (1997). Probabilistic incremental program evolution. Evolutionary Computation, 5(2): 123–141.

627. *Sano, Y. and Kita, H.* (2002). Optimization of noisy fitness functions by means of genetic algorithms using history of search with test of estimation. IEEE Congress on Evolutionary Computation, Honolulu, Hawaii, pages 360–365.

628. *Santana, R.* (1998). Estimation of distribution algorithms with Kikuchi approximations. Evolutionary Computation, 13(1): 67–97.

629. *Santana, R.* (2003). A Markov network based factorized distribution algorithm for optimization. 14th European Conference on Machine Learning, Cavtat, Croatia, pages 337–348.

630. *Santana, R. and Echegoyen, C.* (2012). Matlab Toolbox for Estimation of Distribution Algorithms (MATEDA–2.0). www.sc.ehu.es/ccwbayes/members/rsantana/software/matlab/MATEDA.html.

631. *Santana, R., Larrañaga, P., and Lozano, J.* (2008). Adaptive estimation of distribution algorithms. In *Cotta, C., Sevaux, M., and Sörensen, K., editors*, Adaptive and Multilevel Metaheuristics, pages 177–197. Springer.

632. *Santana-Quintero, L., Montaño, A., and Coello Coello, C.* (2010). A review of techniques for handling expensive functions in evolutionary multi-objective optimization. In *Tenne, Y. and Goh, C.-K., editors*, Computational Intelligence in Expensive Optimization Problems, pages 29–60. Springer.

633. *Sareni, B. and Krähenbühl, L.* (1998). Fitness sharing and niching methods revisited. IEEE Transactions on Evolutionary Computation, 2(3): 97–106.

634. *Sastry, K. and Goldberg, D.* (2000). On extended compact genetic algorithm. Genetic and Evolutionary Computation Conference, Las Vegas, Nevada, pages 352–359.

635. *Sastry, K., Goldberg, D., and Pelikan, M.* (2001). Don't evaluate, inherit. Genetic and Evolutionary Computation Conference, San Francisco, California, pages 551–558.

636. *Savicky, P. and Robnik-Sikonja, M.* (2008). Learning random numbers: A Matlab anomaly. Applied Artificial Intelligence, 22(3): 254–265.

637. Savla, K., Frazzoli, E., and Bullo, F. (2008). Traveling salesperson problems for the Dubins vehicle. IEEE Transactions on Automatic Control, 53(6): 1378–1391.

638. *Sayadi, M., Ramezanian, R., and Ghaffari-Nasab, N.* (2010). A discrete firefly metaheuristic with local search for makespan minimization in permutation flow shop scheduling problems. International Journal of Industrial Engineering Computations, 1(1): 1–10.

639. *Schaffer, C.* (1994). A conservation law for generalization performance. 11th International Conference on Machine Leaming, Boca Raton, Florida, pages 259–265.

640. *Schaffer, J.* (1985). Multiple objective optimization with vector evaluated genetic algorithms. International Conference on Genetic Algorithms and Their Application, Pittsburgh, Pennsylvania, pages 93–100.

641. *Schervish, M.* (1996). P values: What they are and what they are not. The American Statistician, 50(3): 203–206.

642. *Schmidhuber, J.* (1987). Evolutionary principles in self–referential learning, or on learning how to learn: The meta-meta-... hook. PhD thesis, Technische Universitat Miinchen.

643. *Schoenauer, M. and Michalewicz, Z.* (1996). Evolutionary computation at the edge of feasibility. In *Ebeling, W., Rechenberg, I., Schwefel, H.-P., and Voigt, H.-M., editors*, Parallel Problem Solving from Nature – PPSN IV, pages 245–254. Springer.

644. *Schoenauer, M., Sebag, M., Jouve, F., Lamy, B., and Maitournam, H.* (1996). Evolutionary identification of macro-mechanical models. In Angeline, P. and Kinnear, K., editors, Advances in Genetic Programming, volume 2, pages 467–488. MIT Press.

645. *Schoenauer, M. and Xanthakis, S.* (1993). Constrained GA optimization. International Conference on Genetic Algorithms, Urbana-Champaign, Illinois, pages 573–580.

646. *Schrijver, A.* (2005). On the history of combinatorial optimization (till 1960). In *Aardal, K., Nemhauser, G., and Weismantel, R., editors*, Dis-

crete Optimization, volume 12 of Handbooks in Operations Research and Management Science, pages 1–68. Elsevier.

647. *Schultz, T.* (1999). Ants, plants and antibiotics. Nature, 398(6730): 747–748.

648. *Schultz, T.* (2000). In search of ant ancestors. Proceedings of the National Academy of Sciences, 97(26): 14028–14029.

649. *Schumacher, C., Vose, M., and Whitley, L.* (2001). The no free lunch and problem description length. Genetic and Evolutionary Computation Conference, San Francisco, California, pages 565–570.

650. *Schütze, 0., Lara, A., and Coello Coello, C.* (2011). On the influence of the number of objectives on the hardness of a multiobjective optimization problem. IEEE Transactions on Evolutionary Computation, 15(4): 444–454.

651. *Schwefel, H.-P.* (1977). Numerische Optimierung van Computer–Modellen. Birkhauser. The English translation of the title is Evolutionary Strategy and Numerical Optimization.

652. *Schwefel, H.-P.* (1981). Numerical Optimization of Computer Models. John Wiley & Sons. Translation of [Schwefel, 1977] along with some additional material.

653. *Schwefel, H.-P.* (1995). Evolution and Optimum Seeking. John Wiley & Sons. Expanded version of [Schwefel, 1981].

654. *Schwefel, H.-P. and Mendes, M.* (2010). 45 years of evolution strategies. SIGEVOlution, 4(2): 2–8.

655. *Sebag, M. and Ducoulombier, A.* (1998). Extending population-based incremental learning to continuous search spaces. In *Eiben, A., Bäck, T., Schoenauer, M., and Schwefel, H.-P., editors*, Parallel Problem Solving from Nature – PPSN V, pages 418–427. Springer.

656. *Sefrioui, M. and Périaux, J.* (2000). A hierarchical genetic algorithm using multiple models for optimization. In *Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J., and Schwefel, H.-P., editors*, Parallel Problem Solving from Nature – PPSN VI, pages 879–888. Springer.

657. *Selvakumar, A. and Thanushkodi, K.* (2007). A new particle swarm optimization solution to nonconvex economic dispatch problems. IEEE Transactions on Power Systems, 22(1): 42–51.

658. *Seneta, E.* (1966). Markov and the birth of chain dependence theory. International Statistical Review, 64(3): 255–263.

659. *Settles, B*. (2010). Active learning literature survey. Technical report, University of Wisconsin-Madison. www.cs.emu.edu/~bsettles/pub/settles.activelearning.pdf.

660. *Shah-Hosseini, H.* (2007). Problem solving by intelligent water drops. IEEE Congress on Evolutionary Computation, Singapore, pages 3226–3231.

661. *Shakya, S. and Santana, R., editors* (2012). Markov Networks in Evolutionary Computation. Springer.

662. *Shi, L. and Rasheed, K*. (2010). A survey of fitness approximation methods applied in evolutionary algorithms. In *Tenne, Y. and Goh, C.-K., editors*, Computational Intelligence in Expensive Optimization Problems, pages 3–28. Springer.

663. *Shi, Y. and Eberhart, R.* (1999). Empirical study of particle swarm optimization. IEEE Congress on Evolutionary Computation, Washington, District of Columbia, pages 1945–1950.

664. *Shibani, Y., Yasuno, S., and Ishiguro, I.* (2001). Effects of global information feedback on diversity. Advances in Genetic Programming, 45(1): 80–96.

665. *Simões, A.* (2011). Evolutionary Algorithms in Dynamic Optimization Problems. Lambert Academic Publishing.

666. *Simon, D.* (2005). Research in the balance. IEEE Potentials, 24(2): 17–21.

667. *Simon, D*. (2006). Optimal State Estimation. John Wiley & Sons.

668. *Simon, D*. (2008). Biogeography-based optimization. IEEE Transactions on Evolutionary Computation, 12(6): 702–713.

669. *Simon, D*. (2011a). A dynamic system model of biogeography-based optimization. Applied Soft Computing, 11(8): 5652–5661.

670. *Simon, D*. (2011b). A probabilistic analysis of a simplified biogeography-based optimization algorithm. Evolutionary Computation, 19(2): 167–188.

671. *Simon, D*. (2012). Biogeography-Based Optimization Web Site. http://embeddedlab.csuohio.edu/BBO.

672. *Simon, D., Ergezer, M., and Du, D*. (2009). Population distributions in biogeographybased optimization algorithms with elitism. IEEE Conference on Systems, Man, and Cybernetics, San Antonio, Texas, pages 1017–1022.

673. *Simon, D., Ergezer, M., Du, D., and Rarick, R*. (2011a). Markov models for biogeographybased optimization. IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics, 41(1): 299–306.

674. *Simon, D., Rarick, R., Ergezer, M., and Du, D.* (2011b). Analytical and numerical comparisons of biogeography-based optimization and genetic algorithms. Information Sciences, 181(7): 1224–1248.

675. *Singh, H., Ray, T., and Smith, W.* (2010). Surrogate assisted simulated annealing (SASA) for constrained multi-objective optimization. IEEE Congress on Evolutionary Computation, Barcelona, Spain, pages 1–8.

676. *Smith, A. and Tate, D.* (1993). Genetic optimization using a penalty function. International Conference on Genetic Algorithms, Urbana–Champaign, Illinois, pages 499–505.

677. *Smith, R., Dike, B., and Stegmann, S.* (1995). Fitness inheritance in genetic algorithms. Symposium on Applied Computing, Nashville, Tennessee, pages 345–350.

678. *Smith, S.* (1980). A Learning System Based on Genetic Adaptive Algorithms. PhD thesis, University of Pittsburgh.

679. *Šobotník, J., Hanus, R., Kalinová, B., Piskorski, R., Cvačka, J., Bourguignon, T., and Raisin, Y.* (2008). (E,E)-α-Farnesene, an alarm pheromone of the termite Prorhinotermes canalifrons. Journal of Chemical Ecology, 34(4): 478–486.

680. *Socha, K. and Dorigo, M.* (2008). Ant colony optimization for continuous domains. European Journal of Operational Research, 185(3): 1155–1173.

681. *Solnon, C.* (2010). Ant Colony Optimization and Constraint Programming. John Wiley & Sons.

682. *Spears, W. and De Jong, K.* (1997). Analyzing GAs using Markov models with semantically ordered and lumped states. In Belew, R. and Vose, M., editors, Foundations of Genetic Algorithms, volume 4, pages 85–100. Morgan Kaufmann.

683. *Srinivas, N. and Deb, K.* (1994). Multiobjective optimization using nondominated sorting in genetic algorithms. Evolutionary Computation, 2(3): 221–248.

684. *Stanley, K. and Mükkulainen, R.* (2002). Evolving neural networks through augmenting topologies. Evolutionary Computation, 10(2): 99–127.

685. *Stephens, D. and Krebs, J.* (1986). Foraging Theory. Princeton University Press.

686. *Sterne, J. and Smith, G.* (2001). Sifting the evidence – what's wrong with significance tests? Physical Therapy, 81(8): 1464–1469.

687. *Stone, L.* (2009). Zebras. Lerner Publications Company.

688. *Starn, R.* (1996a). Differential evolution design of an HR-filter. IEEE Conference on Evolutionary Computation, Nagoya, Japan, pages 268–273.

689. *Starn, R.* (1996b). On the usage of differential evolution for function optimization. Conference of the North American Fuzzy Information Processing Society, Berkeley, California, pages 519–523.

690. *Starn, R. and Price, K.* (1996). Minimizing the real functions of the ICEC'96 contest by differential evolution. IEEE Conference on Evolutionary Computation, Nagoya, Japan, pages 842–844.

691. *Starn, R. and Price, K.* (1997). Differential evolution – A simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization, 11(4): 341–359.

692. *Stroud, P.* (2001). Kalman-extended genetic algorithm for search in nonstationary environments with noisy fitness evaluations. IEEE Transactions on Evolutionary Computation, 5(1): 66–77.

693. *Stützle, T. and Hoos, H.* (2000). MAX-MIN ant system. Future Generation Computer Systems, 16(8): 889–914.

694. *Su, C. and Lee, C.* (2003). Network reconfiguration of distribution systems using improved mixed-integer hybrid differential evolution. IEEE Transactions on Power Delivery, 18(3): 1022–1027.

695. *Suganthan, P., Hansen, N., Liang, J., Deb, K., Chen, Y.–P., Auger, A., and Tiwari, S.* (2005). Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Technical report. www.iitk.ac.in/kangal/papers/k2005005.pdf, www.ntu.edu.sg/home/EPNSugan/index_files/cec-bench.rnarking.htm.

696. *Sun, J., Lai, C.-H., and Wu, X.-J.* (2011). Particle Swarm Optimisation: Classical and Quantum Perspectives. CRC Press.

697. *Sverdlik, W. and Reynolds, R.* (1993). Incorporating domain specific knowledge into version space search. Fifth International Conference on Tools with Artificial Intelligence, Boston, Massachusetts, pages 216–223.

698. *Syberfeldt, A., Ng, A., John, R., and Moore, P.* (2010). Evolutionary optimisation of noisy multi–objective problems using confidence–based dynamic resampling. European Journal of Operational Research, 204(3): 533–544.

699. *Syswerda, G.* (1991). Schedule optimization using genetic algorithms. In Davis, L., editor, Handbook of Genetic Algorithms, pages 332–349. Van Nostrand Reinhold.

700. *Syswerda, G.* (2010). Differential evolution research – trends and open questions. In Chakraborty, U., editor, Advances in Differential Evolution, pages 1–32. Springer.

701. *Szu, H. and Hartley, R.* (1987). Fast simulated annealing. Physics Letters A, 122(3–4): 157–162.

702. *Takahama, T. and Sakai, S.* (2009). Solving difficult constrained optimziation problems by the E constrained differential evolution with gradient-based mutation. In *MezuraMontes, E., editor*, Constraint–Handling in Evolutionary Optimization, pages 51–72. Springer.

703. *Tan, K., Khor, E., and Lee, T.* (2010). Multiobjective Evolutionary Algorithms and Applications. Springer.

704. *Tanaka, M. and Tanino, T.* (1992). Global optimization by the genetic algorithm in a multiobjective decision support system. International Conference on Multiple Criteria Decision Making, Taipei, Taiwan, pages 261–270.

705. *Tao, G. and Michalewicz, Z.* (1998). Inver-over operator for the TSP. In *Eiben, A., Bäck, T., Schoenauer, M., and Schwefel, H.-P., editors*, Parallel Problem Solving from Nature – PPSN V, pages 803–812. Springer.

706. *Taylor, J.* (1997). An Introduction to Error Analysis: The Study of Uncertainties in Physical Measurements. University Science Books, 2nd edition.

707. *Tenne, Y. and Goh, C.-K., editors* (2010). Computational Intelligence in Expensive Optimization Problems. Springer.

708. *Teodorović, D.* (2003). Transport modeling by multi-agent systems: A swarm intelligence approach. Transportation Planning and Technology, 26(4): 289–312.

709. *Tereshko, V.* (2000). Reaction-diffusion model of a honeybee colony's foraging behaviour. In *Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J., and Schwefel, H.-P., editors*, Parallel Problem Solving from Nature – PPSN VI, pages 807–816. Springer.

710. *Tessema, B. and Yen, G.* (2006). A self adaptive penalty function based algorithm for constrained optimization. IEEE Congress on Evolutionary Computation, Vancouver, Canada, pages 246–253.

711. *Thiele, L., Miettinen, K., Korhonen, P., and Molina, J.* (2009). A preference-based Evolutionary Computation, evolutionary algorithm for multi-objective optimization. 17(3): 411–436.

712. *Thomson, I.* (2010). Culture Wars and Enduring American Dilemmas. The University of Michigan Press.

713. *Tilman, D., May, R., Lehman, C., and Nowak, M.* (1994). Habitat destruction and the extinction debt. Nature, 371(3): 65–66.

714. *Tinoco, J. and Coello Coello, C.* (2013). hypDE: A hyper-heuristic based on differential evolution for solving constrained optimization problems. In *Schütze, O., Coello Coello, C., Tantar, A.-A., Tantar, E., Bouvry, P., Mora, P. D., and Legrand, P., editors*, EVOLVE – A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation II, pages 267–282. Springer.

715. *Tinós, R. and Yang, S.* (2005). Genetic algorithms with self-organized criticality for dynamic optimziation problems. IEEE Congress on Evolutionary Computation, Edinburgh, United Kingdom, pages 2816–2823.

716. *Tizhoosh, H.* (2005). Opposition-based learning: A new scheme for machine intelligence. International Conference on Computational Intelligence for Modelling, Control and Automation, Vienna, Austria, pages 695–701.

717. *Tizhoosh, H., Ventresca, M., and Rahnamayan, S.* (2008). Opposition–based computing. In *Tizhoosh, H. and Ventresca, M., editors*, Oppositional Concepts in Computational Intelligence, pages 11–28. Springer.

718. *Tomassini, M. and Vanneschi, L.* (2009). Introduction: Special issue on parallel and distributed evolutionary algorithms, Part I. Genetic Programming and Evolvable Machines, 10(4): 339–341.

719. *Tomassini, M. and Vanneschi, L.* (2010). Guest editorial: Special issue on parallel and distributed evolutionary algorithms, Part II. Genetic Programming and Evolvable Machines, 11(2): 129–130.

720. *Torregosa, R. and Kanok-Nukulchai, W.* (2002). Weight optimization of steel frames using genetic algorithm. Advances in Structural Engineering, 5(2): 99–111.

721. *Toth, P. and Vigo, D., editors* (2002). The Vehicle Routing Problem. The Society for Industrial and Applied Mathematics.

722. *Tripathi, P., Bandyopadhyay, S., and Pal, S.* (2007). Multi-objective particle swarm optimization with time variant inertia and acceleration coefficients. Information Sciences, 177(22): 5033–5049.

723. *Tsutsui, S.* (2004). Ant colony optimisation for continuous domains with aggregation pheromones metaphor. Fifth International Conference on Recent Advances in Soft Computing (RASC-04), Nottingham, United Kingdom, pages 207–212.

724. *Turing, A.* (1950). Computing machinery and intelligence. Mind, 59(236): 433–460.

725. *Turner, G. and Pitcher, T.* (1986). Attack abatement: A model for group protection by combined avoidance and dilution. The American Naturalist, 128(2): 228–240.

726. *Twain, M.* (2010). Autobiography of Mark Twain, Volume 1. University of California Press.

727. *Tylor, E.* (2009). Primitive Culture: Researches into the Development of Mythology, Philosophy, Religion, Art and Custom, volume 1. Cornell University Library. Originally published in 1871.

728. *Tylor, E.* (2011). Researches into the Early History of Mankind and the Development of Civilization. University of California Libraries. Originally published in 1878.

729. *Ufuktepe, U. and Bacak, G.* (2005). Applications of graph coloring. International Conference on Computational Science and Applications, Singapore, pages 465–477.

730. *van Laarhoven, P. and Aarts, E.* (2010). Simulated Annealing: Theory and Applications. Springer.

731. *Van Veldhuizen, D. and Lamont, G.* (2000). Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. Evolutionary Computation, 8(2): 125–147.

732. *Vavak, F. and Fogarty, T.* (1996). Comparison of steady state and generational genetic algorithms for use in nonstationary environments. IEEE Conference on Evolutionary Computation, Nagoya, Japan, pages 192–195.

733. *Ventresca, M. and Tizhoosh, H.* (2007). Simulated annealing with opposite neighbors. IEEE Symposium on Foundations of Computational Intelligence, Honolulu, Hawaii, pages 186–192.

734. *Volk, T.* (1997). Gaia's Body: Toward a Physiology of Earth. Springer.

735. *Vorwerk, K., Kennings, A., and Greene, J.* (2009). Improving simulated annealing-based FPGA placement with directed moves. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 28(2): 179–192.

736. *Vose, M.* (1990). Formalizing genetic algorithms. IEEE Workshop on Genetic Algorithms, Neural Networks, and Simulated Annealing Applied to Signal and Image Processing, Glasgow, Scotland.

737. *Vose, M.* (1999). The Simple Genetic Algorithm: Foundations and Theory. MIT Press.

738. *Vose, M. and Liepins, G.* (1991). Punctuated equilibria in genetic search. Complex Systems, 5(1): 31–44.

739. *Vose, M. and Wright, A.* (1998a). The simple genetic algorithm and the Walsh transform: Part I, Theory. Evolutionary Computation, 6(3): 253–273.

740. *Vose, M. and Wright, A.* (1998b). The simple genetic algorithm and the Walsh transform: Part II, The inverse. Evolutionary Computation, 6(3): 275–289.

741. *Waghmare, G.* (2013). Comments on «A note on teaching–learning–based optimization algorithm». Information Sciences, in print.

742. *Wallace, A.* (2006). The Geographical Distribution of Animals (two volumes). Adamant Media Corporation. First published in 1876.

743. *Wang, H., Yang, S., Ip, W., and Wang, D.* (2009). Adaptive primal-dual genetic algorithms in dynamic environments. IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics, 39(6): 1348–1361.

744. *Wedde, H., Farooq, M., and Zhang, Y.* (2004). Beehive: An efficient fault-tolerant routing algorithm inspired by honey bee behavior. In *Dorigo, M., Birattari, M., Blum, C., Gambardella, L., Mondada, F., and Stützle, T., editors*, Ant Colony Optimization and Swarm Intelligence: 4th International Workshop, ANTS 2004, pages 83–94. Springer.

745. *Welland, M.* (2009). Sand: The Never-Ending Story. University of California Press.

746. *Welsch, R. and Endicott, K.* (2005). Taking Sides: Clashing Views in Cultural Anthropology. McGraw-Hill, 2nd edition.

747. *Wesche, T., Goertler, G., and Hubert, W.* (1987). Modified habitat suitability index model for brown trout in southeastern Wyoming. North American Journal of Fisheries Management, 7(2):232–237.

748. *Wetzel, C. and Insko, C.* (1982). The similarity-attraction relationship: Is there an ideal one? Journal of Experimental Social Psychology, 18(3): 253–76.

749. *Whigham, P.* (1995). A schema theorem for context-free grammars. IEEE Conference on Evolutionary Computation, Perth, Western Australia, pages 178–181.

750. *Whitley, D.* (1989). The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. International Conference on Genetic Algorithms, Fairfax, Virginia, pages 116–121.

751. *Whitley, D.* (1994). A genetic algorithm tutorial. Statistics and Computing, 4(2): 65–85.

752. *Whitley, D.* (1999). A free lunch proof for gray versus binary encodings. Genetic and Evolutionary Computation Conference, Orlando, Florida, pages 726–733.

753. *Whitley, D.* (2001). An overview of evolutionary algorithms: Practical issues and common pitfalls. Information and Software Technology, 43(14): 817–831.

754. *Whitley, D., Rana, S., Dzubera, J., and Mathias, K.* (1996). Evaluating evolutionary algorithms. Artificial Intelligence, 85(1–2): 245–276.

755. *Whitley, D., Rana, S., and Heckendorn, R.* (1998). The island model genetic algorithm: On separability, population size and convergence. Journal of Computing and Information Technology, 7(1): 33–47.

756. *Whitley, D. and Watson, J.* (2005). Complexity theory and the no free lunch theorem. In Burke, E. and Kendall, G., editors, Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, pages Chapter 11, 317–339. Springer.

757. *Whittaker, R. and Bush, M.* (1993). Dispersal and establishment of tropical forst assemblages, Krakatoa, Indonesia. In Miles, J. and Walton, D., editors, Primary Succession on Land, pages 147–160. Blackwell Science.

758. *Wienke, D., Lucasius, C., and Kateman, G.* (1992). Multicriteria target vector optimization of analytical procedures using a genetic algorithm: Part I. Theory, numerical simulations and application to atomic emission spectroscopy. Analytica Chimica Act, 265(2): 211–225.

759. *Wilson, P. and Macleod, M.* (1993). Low implementation cost IIR digital filter design using genetic algorithms. IEE/IEEE Workshop on Natural Algorithms in Signal Processing, Chelmsford, England, pages 4/1–4/8.

760. *Winchester, S.* (2008). The Day the World Exploded. Collins.

761. *Winston, P. and Horn, B.* (1989). Lisp. Addison Wesley, 3rd edition.

762. *Woldesenbet, Y. and Yen, G.* (2009). Dynamic evolutionary algorithm with variable relocation. IEEE Transactions on Evolutfonary Computation, 13(3): 500–513.

763. *Wolpert, D. and Macready, W.* (1997). No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation, 1(1): 67–82.

764. *Wolpert, D. and Macready, W.* (2005). Coevolutionary free lunches. IEEE Transactions on Evolutionary Computation, 9(6): 721–735.

765. *Worden, L. and Levin, S.* (2007). Evolutionary escape from the prisoner's dilemma. Journal of Theoretical Biology, 245(3): 411–422.

766. *Wright, A., Poli, R., Stephens, C., Langdon, W., and Pulavarty, W.* (2004). An estimation of distribution algorithm based on maximum entropy. Genetic and Evolutionary Computation Conference, Seattle, Washington, pages 343–354.

767. *Wright, S.* (1987). Primal-Dual Interior-Point Methods. Society for Industrial Mathematics.

768. *Wu, J. and Vankat, J.* (1995). Island biogeography theory and applications. In Nierenberg, W., editor, Encyclopedia of Environmental Biology, pages 317–379. Academic Press.

769. *Wu, S. and Chow, T.* (2007). Self-organizing and self-evolving neurons: A new neural network for optimization. IEEE Transactions on Neural Networks, 18(2): 385–396.

770. *Wyatt, T.* (2003). Pheromones and Animal Behaviour: Communication by Smell and Taste. Cambridge University Press.

771. *Xinchao, Z.* (2010). A perturbed particle swarm algorithm for numerical optimization. Applied Soft Computing, 10(1): 119–124.

772. *Yang, C. and Simon, D.* (2005). A new particle swarm optimization technique. International Conference on Systems Engineering, Las Vegas, Nevada, pages 164–169.

773. *Yang, C.-H., Tsai, S.-W., Chuang, L.-Y., and Yang, C.-H.* (2011). A modified particle swarm optimization for global optimization. International Journal of Advancements in Computing Technology, 3(7): 169–189.

774. *Yang, S.* (2003a). Non-stationary problem optimization using the primal-dual genetic algorithm. IEEE Congress on Evolutionary Computation, Canberra, Australia, pages 2246–2253.

775. *Yang, S.* (2003b). PDGA: The primal-dual genetic algorithm. International Conference on Hybrid Intelligent Systems, Melbourne, Australia, pages 214–223.

776. *Yang, S.* (2008a). Genetic algorithms with memory- and elitism-based immigrants in dynamic environments. Evolutionary Computation, 16(3): 385–416.

777. *Yang, S., Ong, Y.-S., and Jin, Y., editors* (2010). Evolutionary Computation in Dynamic and Uncertain Environments. Springer.

778. *Yang, S. and Yao, X.* (2005). Experimental study on population-based incremental learning algorithms for dynamic optimization problems. Soft Computing, 9(11): 815–834.

779. *Yang, S. and Yao, X.* (2008a). Population-based incremental learning with associative memory for dynamic environments. IEEE Transactions on Evolutionary Computation, 12(5): 542–561.

780. *Yang, S. and Yao, X.* (2008b). Population-based incremental learning with associative memory for dynamic environments. IEEE Transactions on Evolutionary Computation, 12(5): 542–561.

781. *Yang, X.-S., editor* (2008b). Nature-Inspired Metaheuristic Algorithms. Luniver Press.

782. *Yang, X.-S.* (2009a). Cuckoo search via Lévy flights. World Congress on Nature f3 Biologically Inspired Computing, Coimbatore, India, pages 210–214.

783. *Yang, X.-S.* (2009b). Firefly algorithm, Lévy flights and global optimization. In Ellis, R. and Petridis, M., editors, Research and Development in Intelligent Systems XXVI, pages 209–218. Springer.

784. *Yang, X.-S.* (2010a). Firefly algorithm, stochastic test functions and design optimisation. International Journal of Bio–Inspired Computation, 2(2): 78–84.

785. *Yang, X.-S.* (2010b). Firefly algorithms for multimodal optimization. In Watanabe, O. and Zeugmann, T., editors, Stochastic Algorithms: Foundations and Applications, pages 169–178. Springer.

786. *Yang, X.-S.* (2010c). A new metaheuristic bat-inspired algorithm. In *González, J., Pelta, D., Cruz, C., Terrazas, G., and Krasnogor, N., editors*, Nature Inspired Cooperative Strategies for Optimization, pages 65–74. Springer.

787. *Yang, Z., Tang, K., and Yao, X.* (2008). Large scale evolutionary optimization using cooperative coevolution. Information Sciences, 178(15): 2985–2999.

788. *Yao, X. and Liu, Y.* (1997). Fast evolution strategies. In *Angeline, P., Reynolds, R., McDonnell, J., and Eberhart, R., editors*, Evolutionary Programming VI, pages 151–161. Springer.

789. *Yao, X., Liu, Y., and Lin, G.* (1999). Evolutionary programming made faster. IEEE Transactions on Evolutionary Programming, 3(2): 82–102.

790. *Yen, G.* (2009). An adaptive penalty function for handling constraint in multi-objective evolutionary optimization. In *Mezura-Montes, E.,*

editor, Constraint-Handling in Evolutionary Optimization, pages 121–143. Springer.

791. *Yoshida, H., Kawata, K., and Fukuyama, Y.* (2001). A particle swarm optimization for reactive power and voltage control considering voltage security assessment. IEEE Transactions on Power Systems, 15(4): 1232–1239.
792. *Yu, X., Tang, K., Chen, T., and Yao, X.* (2009). Empirical analysis of evolutionary algorithms with immigrants schemes for dynamic optimization. Memetic Computing, 1(1): 3–24.
793. *Yuan, B., Orlowska, M., and Sadiz, S.* (2007). On the optimal robot routing problem in wireless sensor networks. IEEE Transactions on Knowledge and Data Engineering, 19(9): 1251–1261.
794. *Zaharie, D.* (2002). Critical values for the control parameters of differential evolution algorithms. International Conference on Soft Computing, Brno, Czech Republic, pages 62–67.
795. *Zavala, A., Aguirre, A., and Diharce, E.* (2009). Continuous constrained optimization with dynamic tolerance using the COPSO algorithm. In *Mezura-Montes, E., editor*, Constraint-Handling in Evolutionary Optimization, pages 1–23. Springer.
796. *Zhan, Z.-H., Zhang, J., Li, Y., and Chung, H.* (2009). Adaptive particle swarm optimization. IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics, 39(6): 1362–1381.
797. *Zhang, J. and Sanderson, A., editors* (2009). Adaptive Differential Evolution. Springer.
798. *Zhang, Q., Sun, J., and Tsang, E.* (2005). An evolutionary algorithm with guided mutation for the maximum clique problem. IEEE Transactions on Evolutionary Computation, 9(2): 192–200.
799. *Zhang, Q., Zhou, A., Zhao, S., Suganthan, P., Liu, W., and Tiwari, S.* (2009). Multiobjective optimization test instances for the CEC 2009 special session and competition. Technical report. www.ntu.edu.sg/home/EPNSugan/index_files/cec-benchmarking.htm.
800. *Zhu, Y., Yang, Z., and Song, J.* (2006). A genetic algorithm with age and sexual features. International Conference on Intelligent Computing, Kunming, China, pages 634–640.
801. *Zimmerman, A. and Lynch, J.* (2009). A parallel simulated annealing architecture for model updating in wireless sensor networks. IEEE Sensors Journal, 9(11): 1503–1510.

802. *Zitzler, E., Deb, K., and Thiele, L.* (2000). Comparison of multiobjective evolutionary algorithms: Empirical results. Evolutionary Computation, 8(2): 173–195.

803. *Zitzler, E., Laumanns, M., and Bleuler, S.* (2004). A tutorial on evolutionary multiobjective optimization. In *Gandibleux, X., Sevaux, M., Sorensen, K., and T'Kindt, V., editors*, Metaheuristics for Multiobjective Optimisation, pages 3–38. Springer.

804. *Zitzler, E., Laumanns, M., and Thiele, L.* (2001). SPEA2: Improving the strength Pareto evolutionary algorithm. EUROGEN 2001: Evolutionary Methods for Design, Optimisation and Control with Applications to Industrial Problems, Athens, Greece, pages 95–100.

805. *Zitzler, E. and Thiele, L.* (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. IEEE Transactions on Evolutionary Computation, 3(4): 257–271.

806. *Zitzler, E., Thiele, L., and Bader, J.* (2010). On set-based multiobjective optimization. IEEE Transactions on Evolutionary Computation, 14(1): 58–79.

807. *Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C., and da Fonseca, V.* (2003). Performance assessment of multiobjective optimizers: An analysis and review. IEEE Transactions on Evolutionary Computation, 7(2): 117–132.

808. *Zlochin, M., Birattari, M., Meuleau, N., and Dorigo, M.* (2004). Model–based search for combinatorial optimization: A critical survey. Annals of Operations Research, 131(1–4): 373–395.

Предметный указатель

Б

Д

Е

З

И

К

Л

М

Н

О

П

Р

С

Т

У

Ф

Книги издательства «ДМК Пресс» можно заказать в торгово-издательском холдинге «Планета Альянс» наложенным платежом, выслав открытку или письмо по почтовому адресу: **115487, г. Москва, 2-й Нагатинский пр-д, д. 6А**.

При оформлении заказа следует указать адрес (полностью), по которому должны быть высланы книги; фамилию, имя и отчество получателя. Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в интернет-магазине: **www.a-planeta.ru**.

Оптовые закупки: тел**. +7(499) 782-38-89**.

Электронный адрес: **books@alians-kniga.ru**.

Дэн Саймон

Алгоритмы эволюционной оптимизации

Биологически обусловленные
и популяционно-ориентированные подходы
к компьютерному интеллекту

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com

Перевод с английского *Логунов А. В.*
Корректор *Синяева Г. И.*
Верстка *Паранская Н. В.*
Дизайн обложки *Мовчан А. Г.*

Формат 70×100$^{1}/_{16}$. Печать цифровая.
Усл. печ. л. 73,27. Тираж 200 экз.

Веб-сайт издательства: *www.dmkpress.com*