А.П. Карпенко

СОВРЕМЕННЫЕ АЛГОРИТМЫ ПОИСКОВОЙ ОПТИМИЗАЦИИ

Алгоритмы, вдохновленные природой

Допущено Учебно-методическим объединением вузов по университетскому политехническому образованию в качестве учебного пособия для студентов высших учебных заведений, обучающихся по направлению подготовки 230100 «Информатика и вычислительная техника»

2-е издание

BAUMANPRESS

Москва
ИЗДАТЕЛЬСТВО
МГТУ им. Н. Э. Баумана
2017

УДК 519.6
ББК 22.19
К26

Рецензенты:
заведующий кафедрой «Информационные технологии в управлении» Российской академии народного хозяйства и государственной службы при Президенте Российской Федерации д-р техн. наук, проф. *А.Н. Данчул;*
профессор кафедры физико-математических методов управления МГУ им. М.В. Ломоносова, Главный научный сотрудник лаборатории оптимизации управляемых систем ИПУ им. В.А. Трапезникова РАН, д-р техн. наук *Н.Б. Филимонов*

Карпенко, А. П.

К26 Современные алгоритмы поисковой оптимизации. Алгоритмы, вдохновленные природой : учебное пособие / А. П. Карпенко. — 2-е изд. — Москва : Издательство МГТУ им. Н. Э. Баумана, 2017. — 446, [2] с. : ил.

ISBN 978-5-7038-4634-6

Учебное пособие посвящено, преимущественно, рассмотрению современных стохастических популяционных алгоритмов решения однокритериальной задачи оптимизации. Рассмотрены методы повышения эффективности этих алгоритмов путем их гибридизации и метаоптимизации. Наряду с однокритериальной рассматривается задача многокритериальной оптимизации и популяционные алгоритмы ее решения. Представлены методы распараллеливания указанных алгоритмов. Содержит большое число примеров решения тестовых и практически значимых задач оптимизации.

Для студентов высших учебных заведений, обучающихся по направлению 230100 «Информатика и вычислительная техника». Может быть полезно для всех студентов, изучающих курс «Методы оптимизации» и близкие по тематике курсы. Материал пособия представляет интерес также для аспирантов и специалистов, использующих в своей работе методы, алгоритмы и программы оптимизации.

УДК 519.6
ББК 22.19

© Карпенко А. П., 2014
© Оформление. Издательство МГТУ им. Н.Э. Баумана, 2017

ISBN 978-5-7038-4634-6

Моим любимым дочерям.
Главное, что вы есть!

Предисловие

В последние годы интенсивно развиваются алгоритмы поисковой оптимизации, которые называют поведенческими, интеллектуальными, метаэвристическими, вдохновленными (инспирированными) природой, роевыми, многоагентными, популяционными и т. д. Эффективность таких алгоритмов соизмерима, а часто превосходит эффективность ставших уже классическими эволюционных алгоритмов, среди которых наиболее известен генетический алгоритм. С помощью популяционных алгоритмов успешно решаются сложные оптимизационные задачи, например, задачи автоматизированного проектирования, синтеза сложных химических соединений, оптимального управления динамическими системами.

Большое число, прежде всего, англоязычных публикаций посвящено разработке, исследованию эффективности и практическому применению популяционных алгоритмов. В то же время для многих алгоритмов полностью или практически отсутствуют даже журнальные русскоязычные публикации. Данное учебное пособие призвано восполнить этот пробел и предоставить учащемуся широкий обзор современных популяционных алгоритмов поисковой оптимизации.

Основное внимание в пособии уделено популяционным алгоритмам глобальной оптимизации. Наряду с этим достаточно широко представлены алгоритмы этого класса, ориентированные на решение задачи многокритериальной оптимизации, точнее говоря – на приближенное построение множества и фронта Парето этой задачи. Поскольку практически значимые задачи оптимизации имеют, как правило, высокую вычислительную сложность, в пособии рассмотрены также вопросы реализации популяционных алгоритмов на параллельных вычислительных системах различной архитектуры. В целом, в пособии рассмотрены все основные аспекты популяционных алгоритмов – представлены канонические алгоритмы, большое число их модификаций, методы повышения эффективности алгоритмов путем гибридизации и метаоптимизации, методы распараллеливания вычислений. В пособии также представлено большое число примеров решения тестовых и практически значимых задач однокритериальной и многокритериальной оптимизации.

Пособие ориентировано, в первую очередь, на студентов высших учебных заведений, обучающихся по направлению 230100 «Информатика и вычислительная техника». В то же время оно может быть полезным для всех студентов, изучающих курс «Методы оптимизации» и близкие по тематике курсы. Материал пособия представляет интерес также для аспирантов и специалистов, использующих в своей работе методы, алгоритмы и программы оптимизации.

Пособие в значительной степени использует материалы лекций по курсу «Методы оптимизации», который автор в течение ряда лет читает студентам старших курсов специальности «Системы автоматизированного проектирования» МГТУ им. Н. Э. Баумана.

Автор благодарит своих аспирантов и студентов Абрамова Д. О., Антух А. Э., Бензу Н. Н., Березина Д. В., Гришина А. А., Егорова А. И., Карипова Д. Р., Комарова С. В., Кудряшова А. М., Митину Е. В., Мишина С. Н., Мурсякаева М. А., Мухлисуллина Э. Т., Овешникова А. А., Плякина Д. А., Семенихина А. С., Синяговскую О. А., Сорокина А. С., Сундукова К. В., Хасанову Р. В., Чернобривченко К. А., Шурова Д. Л. за помощь в проведении вычислительных экспериментов, результаты которых представлены в книге. Автор отмечает особенно большую помощь в подготовке книги Селиверстова Е. Ю. и выражает ему отдельную благодарность.

Модель объекта управления (п. 2.13) заимствована из книги: Федоренко Р. П. Приближенное решение задач оптимального управления.– М.: Наука, 1978.– 488 с. Задача, рассмотренная в п. 3.1.6, предоставлена И. М. Губайдуллиным (Институт нефтехимии и катализа РАН, г. Уфа). Модель системы «двигатель–генератор» взята из книги Александрова А. Г. Оптимальные и адаптивные системы.– М.: Высшая школа, 1989.– 263 с. Тестовая задача оптимизации конструкции консольной балки, рассмотренная в п. 5.1.3, заимствована из работы *Lee KS., Geem ZW. A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice // Computational Methods Applied Mechanics Engineering, 2005, Vol. 194, pp. 3902–3933.*

Основные обозначения

$abs\,(\cdot)$ – символ абсолютного значения числа

$D, D_X \subset R^{|X|}$ – множества допустимых значений вектора варьируемых параметров X

$D_F \subset R^{|F|}$ – множество достижимости многоцелевой задачи оптимизации

$D_F^* \subset D_F$ – фронт Парето многоцелевой задачи оптимизации

$D_X^* \subset D_X$ – множество Парето многоцелевой задачи оптимизации

$E(X) = (e_1(X), e_2(X), ..., e_{|E|}(X))$ — $(|E| \times 1)$-ограничивающая вектор-функция, с помощью которой задаются ограничения типа равенств на вектор варьируемых параметров X

$E_\nu(a, b)$ — $(\nu \times 1)$-вектор независимых вещественных случайных чисел, распределенных по экспоненциальному закону $\xi(x) = \dfrac{1}{2b} \exp\left(-\dfrac{|x-a|}{b}\right)$, $a, b > 0$, $x \in (-\infty, \infty)$

$F(X) = \left(f_1(X), f_2(X), ..., f_{|F|}(X)\right)$ – целевая вектор-функция (векторный критерий оптимальности) со значениями в пространстве $R^{|F|}$

$F(X^*) = F^*$ – искомый целевой вектор

$f(X)$ – скалярная целевая функция (критерий оптимальности) со значениями в пространстве R^1

$f(X^*) = f^*$ – искомое оптимальное значение целевой функции $f(X)$

$\tilde{f}^*$ – приближенное значение оптимума целевой функции $f(X)$

$G(X) = (g_1(X), g_2(X), ..., g_{|G|}(X))$ – $(|G| \times 1)$-ограничивающая вектор-функция, с помощью которой задаются ограничения типа неравенств на вектор варьируемых параметров X

$H = (h_1, h_2, ..., h_{|H|})$ – хромосома (закодированный вектор X), соответствующая текущему положению агента в пространстве поиска

$H_i^g = (h_{i,j}, j \in [1 : |H_i^g|])$ – ген, соответствующий компоненте x_i вектора X

$L_\nu(\lambda)$ – $(\nu \times 1)$-вектор независимых вещественных случайных чисел, распределенных по закону Леви; используется в полетах Леви

$N_\nu(m, \sigma)$ – $(\nu \times 1)$-вектор независимых вещественных случайных чисел, распределенных по нормальному закону с математическим ожиданием m и средним квадратичным отклонением σ

n_f – число испытаний (вычислений значений целевой функции)

$\overline{n}_f$ – среднее число испытаний

$Null$ – пустое множество

$R^{|A|}$ – $|A|$-мерное арифметическое пространство

$S = S(t) = (s_i, i \in [1:|S|])$ – текущая популяция

$s_i = s_i(t)$ – агент популяции S

$S' = S(t+1) = (s'_i, i \in [1:|S|])$ – следующая популяция

$s'_i = s_i(t+1)$ – агент популяции S'

$\boldsymbol{S} = \left(S_i, i \in [1:|\boldsymbol{S}|]\right)$ – мультипопуляция

S_i – субпопуляция мультипопуляции $\boldsymbol{S}$

$sign\,(a)$ – знак числа a

^{T} – символ транспонирования

$\hat{t}$ – максимальное допустимое число поколений (итераций)

t – номер текущего поколения (итерации)

t_{end} – поколение (итерация), соответствующее началу стагнации вычислений

$\overline{t}_{end}$ – среднее по числу запусков мультистарта число поколений (итераций)

$U_\nu(a;b)$ – $(\nu \times 1)$-вектор независимых вещественных случайных чисел, равномерно распределенных в интервале $[a;b]$; a, b – вещественные числа; $b > a$

$U_\nu(n_1:n_2)$ – $(\nu \times 1)$-вектор независимых целых случайных чисел, равномерно распределенных в интервале $[n_1:n_2]$; n_1, n_2 – целые числа; $n_2 > n_1$

u^0_{sign} – случайная величина, с равной вероятностью принимающая значения (–1), (0), (1)

$u^{\pm 1}_{sign}$ – случайная величина, с равной вероятностью принимающая значения (1), (–1)

$X = (x_1, x_2, ..., x_{|X|})$ – $(|X| \times 1)$-вектор варьируемых параметров

$X_i = (x_{i,1}, x_{i,2}, ..., x_{i,|X|})$ – $(|X| \times 1)$-вектор варьируемых параметров, определяющий положение агента $s_i \in S$; $i \in [1:|X|]$

$X'_i = (x'_{i,1}, x'_{i,2}, ..., x'_{i,|X|})$ – $(|X| \times 1)$-вектор варьируемых параметров, определяющий положение агента $s'_i \in S'$; $i \in [1:|X|]$

X^* – искомый оптимальный вектор варьируемых параметров X

$\tilde{X}^*$ – приближенный вектор X^*

δ_t – критерий стагнации вычислительного процесса: если в течение данного числа поколений значение фитнес-функции не меняется, то считают, что имеет место стагнация вычислений

$\varphi(X) = \varphi(s) = \varphi(H)$ – фитнес-функция (функция приспособленности, приспособленность) агента s. Если не оговорено противное, то имеется в виду минимизация этой функции

$\chi(x) = \begin{cases} 1, & x \geq 0, \\ 0, & x < 0 \end{cases}$ – функция Хевисайда

$\{c_1, c_2, ...\}$ – совокупность (множество) элементов $c_1, c_2, ...$

$|A|$ – размерность вектора A

$|B|$ – число элементов во множестве B

$\lceil v \rceil$ – ближайшее целое большее v

$\lfloor v \rfloor$ – ближайшее целое меньшее v

$\lfloor v, w \rfloor$ – ближайшее целое меньшее v и кратное w

$\|\cdot\|$ – векторная норма

$\|\cdot\|_E$ – евклидова векторная норма

$\|A_1, A_2\|_E = \sqrt{\sum_{i=1}^{|A|} \left(a_{1,i} - a_{2,i}\right)^2}$ – евклидово расстояние между целочисленными или вещественными векторами A_1, A_2 размерности $|A|$

$\|B_1, B_2\|_H$ – расстояния Хемминга между бинарными векторами B_1, B_2, равное числу различающихся битов в одноименных разрядах указанных векторов

$\|A_1, A_2\|_M^1 = \sum_{i=1}^{|A|} abs\left(a_{1,i} - a_{2,i}\right)$ – манхэттенское расстояние между целочисленными или вещественными векторами A_1, A_2 размерности $|A|$

$\|A_1, A_2\|_M^{|A|} = \left(\sum_{i=1}^{|A|} abs\left(a_{1,i} - a_{2,i}\right)^{|A|}\right)^{\frac{1}{|A|}}$ – $L_{|A|}$-манхэттенское расстояние между целочисленными или вещественными векторами A_1, A_2 размерности $|A|$

$\nabla f\left(X'\right) = \left(\left.\frac{\partial f(X)}{\partial x_1}\right|_{X=X'} \left.\frac{\partial f(X)}{\partial x_2}\right|_{X=X'} \ldots \left.\frac{\partial f(X)}{\partial x_{|X|}}\right|_{X=X'} \right)^T$ – градиент функции $f(X)$ в точке X'

(A, B) – скалярное произведение векторов A, B, имеющих одинаковую размерность

$A \otimes B = \left(a_1 b_1 \quad a_2 b_2 \quad \ldots \quad a_{|A|} b_{|B|}\right)$ – прямое произведение векторов A, B одинаковой размерности $|A| = |B|$

$\Pi = \{X \mid X^- \le X \le X^+\} = \{X \mid x_i^- \le x_i \le x_i^+, i \in [1:|X|]\}$ – гиперпараллелепипед допустимых значений вектора варьируемых параметров

ЛПР – лицо, принимающее решения

ОДУ – система обыкновенных дифференциальных уравнений

Замечание. В силу того, что любой программный генератор случайных чисел в действительности генерирует псевдослучайные числа, вместо слов «случайный вектор», «случайная величина» и так далее следовало бы писать «псевдослучайный вектор», «псевдослучайная величина» и т. д. В книге для компактности письма этот нюанс опущен.

Введение

Многие задачи, возникающие в таких фундаментальных науках, как физика, химия и молекулярная биология, а также во многих прикладных науках, сводятся к задачам непрерывной глобальной оптимизации. Особенностями таких задач часто являются нелинейность, недифференцируемость, многоэкстремальность (мультимодальность), овражность, отсутствие аналитического выражения (плохая формализованность) и высокая вычислительная сложность оптимизируемых функций, высокая размерность пространства поиска, сложная топология области допустимых значений и т. д.

С самой общей точки зрения, именно указанные особенности задач глобальной оптимизации объясняют отсутствие универсального алгоритма их решения и, напротив, наличие чрезвычайно большого числа алгоритмов, их модификаций и гибридизаций. Число таких алгоритмов увеличивают также их параллельные версии и новые параллельные алгоритмы оптимизации, ориентированные на различные классы параллельных вычислительных систем.

Для эффективного решения задач глобальной оптимизации в 1980-х гг. начали интенсивно разрабатывать класс стохастических поисковых алгоритмов оптимизации, которые в разных публикациях называют поведенческими, интеллектуальными, метаэвристическими, вдохновленными (инспирированными) природой, роевыми, многоагентными, популяционными и т. д. Последний термин наиболее отвечает сути этих алгоритмов.

Популяционные алгоритмы (*population algorithms*) предполагают одновременную обработку нескольких вариантов решения задачи оптимизации и представляют собой альтернативу классическим «траекторным» поисковым алгоритмам, в которых в области поиска эволюционирует только один кандидат на решение этой задачи.

Подавляющее большинство рассматриваемых алгоритмов опубликовано в англоязычной литературе, в которой вместо традиционного для русскоязычного читателя термина «метод» принято использовать термин «алгоритм». Для того чтобы избежать возможной неоднозначности идентификации рассматриваемых объектов, также используется последний термин, хотя он с точки зрения русскоязычных публикаций не совсем корректен.

Задачи глобальной оптимизации делятся на два класса – детерминированные задачи и стохастические задачи. В первом случае оптимизируемая функция и функции, ограничивающие область решения задачи, являются детерминированными, то есть не содержат случайных параметров. Во втором случае одна или несколько указанных функций содержат такие параметры. В книге рассматриваются только детерминированные задачи оптимизации.

С другой точки зрения, среди задач глобальной оптимизации выделяют статические (стационарные) и динамические (нестационарные) задачи. В статических задачах оптимизируемая функция и область ее допустимых значений не меняются во времени, так что остаются неизменными положения в области

поиска ее локальных и глобальных экстремумов. В динамических задачах положения указанных экстремумов могут меняться во времени, и задача состоит в отслеживании движения глобального экстремума. Книга посвящена статическим задачам оптимизации, хотя многие из рассматриваемых популяционных алгоритмов могут быть модифицированы (и такие модификации известны) для решения также динамических задач оптимизации.

Таким образом, предметом рассмотрения являются детерминированные статические задачи глобальной безусловной (без ограничений на значения варьируемых параметров) и условной (с ограничениями на эти значения) непрерывной оптимизации. Последнее означает, что допустимыми значениями параметров являются вещественные числа.

По способу определения направления движения к экстремуму алгоритмы поисковой оптимизации разделяют на алгоритмы детерминированного (регулярного) поиска (например, классический алгоритм наискорейшего спуска) и алгоритмы стохастического (случайного) поиска. Все рассматриваемые популяционные алгоритмы относятся к классу стохастических.

Алгоритмы поисковой оптимизации разделяют еще на два следующих класса: алгоритмы, использующие как пробные, так и рабочие шаги поиска; алгоритмы, в которых пробные и рабочие шаги совмещены. Представленные в книге популяционные алгоритмы реализуют оба эти варианта поиска. Заметим, что в алгоритмах решения динамических задач оптимизации используют только совмещение пробных и рабочих шагов.

Все популяционные алгоритмы относятся к классу эвристических алгоритмов (*heuristic algorithms*), то есть алгоритмов, для которых сходимость к глобальному решению не доказана, но экспериментально установлено, что в большинстве случаев они дают достаточно хорошее решение.

В качестве общего названия членов популяции используем термин агент (*agent*). В различных популяционных алгоритмах агенты называются индивидами, особями, частицами, муравьями, пчелами и т. д.

Общая схема популяционных алгоритмов включает в себя следующие этапы.

1. *Инициализация популяции*. В области поиска тем или иным образом создаем некоторое число начальных приближений к искомому решению задачи – инициализируем популяцию агентов.

2. *Миграция агентов популяции*. С помощью некоторого набора миграционных операторов, специфических для каждого из популяционных алгоритмов, перемещаем агентов в области поиска таким образом, чтобы в конечном счете приблизиться к искомому экстремуму оптимизируемой функции.

3. *Завершение поиска*. Проверяем выполнение условий окончания итераций, и если они выполнены, завершаем вычисления, принимая лучшее из найденных положений агентов популяции за приближенное решение задачи. Если указанные условия не выполнены, возвращаемся к выполнению этапа 2.

При инициализации популяции могут быть использованы детерминированные и случайные алгоритмы. Формирование начальной популяции, агенты которой находятся вблизи глобального экстремума оптимизируемой функции, может существенно сократить время решения задачи. Однако обычно априорная

информация о местоположении этого экстремума отсутствует, поэтому агентов начальной популяции распределяют равномерно по всей области поиска.

Миграцию агентов, например, в генетическом алгоритме, который мы также относим к популяционным алгоритмам, реализуют генетические операторы скрещивания, мутации и другие.

В качестве условия окончания поиска используют, как правило, условие достижения заданного числа итераций (*поколений*). Часто используют также условие стагнации (*stagnation*) алгоритма, когда лучшее достигнутое значение оптимизируемой функции не изменяется в течение заданного числа поколений. Могут быть использованы и другие условия, например, условие исчерпания времени, отпущенного на решение задачи.

Из представленной общей схемы популяционных алгоритмов следует, что они обладают ярко выраженной модульной структурой, позволяющей легко получить большое число вариантов любого из алгоритмов путем варьирования и комбинирования правил инициализации популяции, миграционных операторов и условий завершения поиска.

Агенты популяции обладают следующими основными свойствами.

- Автономность – агенты движутся в пространстве поиска, хотя бы частично, независимо друг от друга.
- Стохастичность – процесс миграции агентов содержит случайную компоненту.
- Ограниченность представления – каждый из агентов популяции обладает информацией лишь об исследуемой им части области поиска и, быть может, об окружении некоторых других агентов.
- Децентрализация – отсутствие агентов, управляющих процессом поиска в целом.
- Коммуникабельность – агенты тем или иным способом могут обмениваться между собой информацией о топологии (ландшафте) оптимизируемой функции, выявленной в процессе исследования своей части области поиска.

Даже если стратегия поведения каждого из агентов популяции достаточно проста, указанные свойства агентов обеспечивают формирование так называемого роевого интеллекта (*swarm intelligence*) популяции, проявляющегося в самоорганизации и сложном поведении популяции в целом.

Одной из особенностей популяционных алгоритмов оптимизации является то, что в подавляющем большинстве случаев для них имеется достаточно любопытная аналогия в человеческом обществе, живой или неживой природе. Так, известны популяционные алгоритмы эволюции разума, колонии муравьев, роя медоносных пчел, светлячков, гравитационного и электромагнитного поиска и т. д.

Популяционные алгоритмы поисковой оптимизации в сравнении с классическими алгоритмами имеют неоспоримые преимущества, прежде всего, при решении задач высокой размерности, мультимодальных и плохо формализованных задач. В этих условиях популяционные алгоритмы могут обеспечить высокую вероятность локализации глобального экстремума оптимизируемой функции. Важно также, что популяционные алгоритмы позволяют эффективнее классических алгоритмов отыскать субоптимальное (близкое к оптимальному) решение. Часто достаточным является именно такое решение.

Популяционные алгоритмы почти наверняка проиграют алгоритму полного перебора при решении задачи оптимизации невысокой размерности. В случае гладкой и унимодальной оптимизируемой функции популяционные алгоритмы, как правило, менее эффективны, чем любой классический градиентный метод. К недостаткам популяционных алгоритмов следует отнести также сильную зависимость их эффективности от значений свободных параметров, число которых в большинстве алгоритмов достаточно велико.

Важно понимать, что для решения любой задачи оптимизации во множестве всех известных оптимизационных алгоритмов, включая популяционные алгоритмы, наверняка найдется хотя бы один алгоритм, который даст, как минимум, удовлетворительные результаты. Однако нет и не может быть алгоритма, который мог бы обеспечить высокую эффективность при решении всех задач оптимизации. Поэтому для каждого из популяционных алгоритмов могут быть выделены классы задач, которые он решает

– лучше других алгоритмов,
– примерно как другие алгоритмы,
– хуже других алгоритмов.

Важнейшим понятием популяционных алгоритмов является понятие фитнес-функции (*fitness-function*). Часто эту функцию называют функцией пригодности, функцией полезности, функцией приспособленности и т. д. Важность функции обусловлена тем обстоятельством, что с ее помощью оценивают «качество» агентов популяции. Стратегически, в процессе миграции агенты движутся таким образом, чтобы приблизиться к глобальному экстремуму фитнес-функции.

Часто, но далеко не всегда, фитнес-функция совпадает с оптимизируемой функцией. В общем случае в качестве фитнес-функции используют некоторое детерминированное или стохастическое, линейное или нелинейное преобразование оптимизируемой функции. Простейший пример – применение в качестве фитнес-функции инвертированной оптимизируемой функции. Более сложный пример – использование для построения фитнес-функции известного метода штрафных функций.

С использованием понятия фитнес-функции можно сказать, что суть популяционных алгоритмов состоит в обеспечении более высокой средней приспособленности положений агентов данного поколения по сравнению с их приспособленностью в предыдущем поколении.

Одна из основных проблем конструирования популяционных алгоритмов – обеспечение баланса между интенсивностью поиска (скоростью сходимости алгоритма) и широтой поиска (диверсификацией поиска). Интенсификация (*intensification*) поиска требует быстрой сходимости алгоритма, что означает быстрое уменьшение разнообразия популяции. Напротив, диверсификация (*diversification*) поиска призвана обеспечить более широкий обзор пространства поиска и более высокую вероятность локализации глобального экстремума задачи. Диверсификация требует сохранения разнообразия популяции в течение как можно большего числа поколений популяционного алгоритма.

Первые поколения популяционного алгоритма преимущественно занимаются исследованием области поиска, чтобы найти новых кандидатов на решение

задачи. Агенты последних поколений алгоритма заняты исключительно уточнением найденных ранее решений или решения. Промежуточные поколения в некоторой мере совмещают разведку и эксплуатацию разведанных решений.

Наиболее развитыми механизмами решения проблемы баланса между интенсивностью и широтой поиска являются механизмы адаптации (*adapting*) и самоадаптации (*self-adapting*) популяционных алгоритмов. Эти механизмы реализуют изменение значений свободных параметров алгоритмов так, чтобы обеспечить постепенный переход от диверсификации к интенсификации поиска.

Поскольку популяционные алгоритмы являются стохастическими, их эффективность, как правило, меняется в широких пределах в зависимости от удачности начального приближения, полученного на этапе инициализации популяции. В этой связи для оценки эффективности данных алгоритмов используют многократные прогоны алгоритма, исходя из разных начальных приближений (метод мультистарта). Основными критериями эффективности популяционных алгоритмов являются надежность алгоритма – оценка вероятности локализации глобального экстремума, а также скорость его сходимости – оценка математического ожидания необходимого числа испытаний (вычислений значения оптимизируемой функции).

Можно предложить несколько классификаций популяционных алгоритмов. На верхнем иерархическом уровне выделяют следующие классы таких алгоритмов:

– эволюционные алгоритмы, включая генетические алгоритмы;
– популяционные алгоритмы, вдохновленные живой природой;
– алгоритмы, вдохновленные неживой природой;
– алгоритмы, инспирированные человеческим обществом;
– прочие алгоритмы.

Первая глава книги содержит постановку детерминированной задачи непрерывной оптимизации и классификацию алгоритмов ее решения. Здесь же рассмотрены некоторые классические непопуляционные стохастические алгоритмы решения задачи локальной безусловной оптимизации, аналогичные алгоритмы решения задачи условной оптимизации, а также основные неклассические алгоритмы глобальной оптимизации.

Во второй главе рассмотрены генетический алгоритм и другие эволюционные алгоритмы.

Третья глава освещает наиболее известные и развитые популяционные алгоритмы, относящиеся к классу алгоритмов, вдохновленных живой природой.

В четвертой главе рассматриваются менее известные и развитые популяционные алгоритмы, вдохновленные живой природой.

Пятая глава посвящена популяционным алгоритмам, инспирированным неживой природой, человеческим обществом, и некоторым другим алгоритмам, которые трудно отнести к одному из указанных классов.

В настоящее время основным средством конструирования новых высокоэффективных популяционных алгоритмов является гибридизация (*hybridization*) популяционного алгоритма с двумя и более популяционными и/или не популяционными алгоритмами. В шестой главе рассматривается классификация ме-

тодов гибридизации, общие принципы гибридизации, приводятся примеры основных классов гибридных популяционных алгоритмов.

Как отмечалось выше, одной из особенностей популяционных алгоритмов является наличие в них значительного, а часто очень большого числа свободных параметров. От значений этих параметров может сильно зависеть эффективность алгоритма, однако формальные рекомендации по выбору значений этих параметров, исходя из особенностей решаемой задачи оптимизации, как правило, отсутствуют. В этой связи в настоящее время интенсивно развиваются методы адаптации и самоадаптации значений этих параметров, называемые методами мета-оптимизации (*meta-optimization*). Данная проблематика рассматривается в седьмой главе.

При решении практических, прежде всего, технических задач оптимизации часто возникает задача многоцелевой (*multi objective*) оптимизации, когда желательно экстремизировать не одну, а несколько целевых функций. Хорошо известно, что, поскольку нельзя добиться экстремального значения всех целевых функций одновременно, решение такой задачи представляет собой некоторый компромисс, принадлежащий так называемому множеству Парето (*Pareto set*) задачи, то есть множеству решений, не улучшаемых одновременно по всем оптимизируемым функциям. Часто решением задачи многоцелевой оптимизации называют множество Парето этой задачи, предполагая, что после построения некоторой аппроксимации данного множества лицо, принимающее решение, неформальными или формализованными методами выберет из него единственное решение. Популяционные алгоритмы решения (в указанном смысле) задачи многоцелевой оптимизации рассматриваются в восьмой главе. Глава содержит постановку задачи многоцелевой оптимизации, включая формальное определение множества Парето, классификацию алгоритмов решения этой задачи, а также обзор популяционных алгоритмов построения аппроксимации множества Парето.

В связи с высокой вычислительной сложностью практически значимых задач оптимизации и быстро возрастающей доступностью различных высоко- и сверхвысокопроизводительных вычислительных систем, все более актуальной становится проблема разработки параллельных популяционных алгоритмов, ориентированных на различные классы параллельных ЭВМ. Этим алгоритмам посвящена девятая глава, в которой рассмотрена структура основных классов параллельных ЭВМ, приведена классификация методов синтеза параллельных алгоритмов оптимизации, а также примеры таких алгоритмов.

Во всех случаях в книге изложены, по сути, не алгоритмы, а лишь их схемы. Детали алгоритмов, быть может, весьма существенные, но затеняющие их суть, опущены. Поэтому материал книги не следует использовать непосредственно для программной реализации представленных популяционных алгоритмов – следует обращаться к оригинальным публикациям авторов этих алгоритмов.

Сводка основных популяционных алгоритмов, рассмотренных в работе, приведена в Приложении А.

Основные обозначения, используемые в книге, стандартизованы и почти всюду не совпадают с исходными авторскими обозначениями.

Глава 1

ПОСТАНОВКА ЗАДАЧИ ПОИСКОВОЙ ОПТИМИЗАЦИИ И НЕПОПУЛЯЦИОННЫЕ СТОХАСТИЧЕСКИЕ АЛГОРИТМЫ ЕЕ РЕШЕНИЯ

В первом параграфе главы даны общая постановка детерминированной задачи оптимизации, классификация этих задач по различным классификационным признакам, а также классификация алгоритмов их решения. Во втором и третьем параграфах рассмотрены непопуляционные стохастические алгоритмы решения задач, соответственно, локальной безусловной оптимизации и локальной условной оптимизации. Четвертый параграф посвящен таким же алгоритмам решения задачи глобальной оптимизации.

1.1. ПОСТАНОВКА И КЛАССИФИКАЦИЯ АЛГОРИТМОВ РЕШЕНИЯ ДЕТЕРМИНИРОВАННОЙ ЗАДАЧИ ПОИСКОВОЙ ОПТИМИЗАЦИИ

1.1.1. Постановка задачи

Рассмотрим *детерминированную задачу оптимизации*

$$\min_{X \in D \subset R^{|X|}} f\left(X\right) = f\left(X^*\right) = f^*, \tag{1.1}$$

где $|X|$ – размерность *вектора варьируемых параметров* $X = (x_1, x_2, ..., x_{|X|})^T$; $R^{|X|}$ – $|X|$-мерное арифметическое пространство; $f(X)$ – *целевая функция* (*критерий оптимальности*) со значениями в пространстве R^1; X^*, f^* – искомые оптимальное решение и значение целевой функции соответственно;

$$D = \{X \mid E(X) = 0, G(X) \geq 0\} \; - \tag{1.2}$$

множество допустимых значений вектора X. Здесь $E(X)$ – *ограничивающая вектор-функция* с компонентами $e_1(X)$, $e_2(X), \ldots, e_{|E|}(X)$, а $G(X)$ – аналогичная функция с компонентами $g_1(X)$, $g_2(X), \ldots, g_{|G|}(X)$.

Детерминированность поставленной задачи оптимизации означает, что целевая функция $f(X)$ и ограничивающие функции $E(X)$, $G(X)$ не содержат в себе случайных параметров.

Целевую функцию $f(x)$, где $x \in [a; b]$ – скаляр, называют *унимодальной*, если в области ее определения $[a; b]$ существует точка $x_* \in [a; b]$ такая, что на полуинтервале $[a; x_*)$ функция $f(x)$ убывает, а на полуинтервале $(x_*; b]$ – возрастает. Точка x_* может быть как внутренней точкой отрезка $[a; b]$, так и совпадать с одним из его концов. Определение унимодальности целевой функции не требует ее непрерывности.

Непрерывную в своей области определения $[a;b]$ одномерную целевую функцию $f(x)$ называют *выпуклой* (вниз), если для любых точек $x_1, x_2 \in [a;b]$, $x_1 \neq x_2$ и для любых $\lambda \in [0;1]$ выполняется неравенство

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda) f(x_2).$$

Определение выпуклой целевой функции не требует ее унимодальности.

Непрерывную в своей области определения $[a;b]$ одномерную целевую функцию $f(x)$ называют *строго выпуклой* (вниз), если для любых точек $x_1, x_2 \in [a;b]$, $x_1 \neq x_2$ и для любого $\lambda \in [0;1]$ выполняется неравенство

$$f(\lambda x_1 + (1-\lambda)x_2) < \lambda f(x_1) + (1-\lambda) f(x_2).$$

Строго выпуклая целевая функция является унимодальной.

Если множество D представляет собой *выпуклое множество*, то в случае $n \geq 2$ также определено понятие выпуклой целевой функции. Непрерывную целевую функцию $f(X)$, где $X \in D$ и множество D является выпуклым, называют *выпуклой* (вниз), если для любых $X_1, X_2 \in D$, $X_1 \neq X_2$ и любого $\lambda \in [0;1]$ имеет место неравенство

$$f(\lambda X_1 + (1-\lambda)X_2) \leq \lambda f(X_1) + (1-\lambda) f(X_2).$$

Аналогично, целевую функцию $f(X)$ называют *строго выпуклой* (вниз), если для любых $X_1, X_2 \in D$, $X_1 \neq X_2$ и любого $\lambda \in [0;1]$ справедливо неравенство

$$f(\lambda X_1 + (1-\lambda)X_2) < \lambda f(X_1) + (1-\lambda) f(X_2).$$

Выпуклая целевая функция может иметь более одной точки локального минимума, а строго выпуклая функция – только одну точку. Например, выпуклая квадратичная функция $f(X) = f(x_1, x_2) = (x_1 + x_2)^2$ достигает в точке (0; 0) своего минимума, значение которого равно нулю. Но это же значение функция $f(X)$ принимает во всех точках вида $(x_1;\ x_2 = -x_1)$.

Целевую функцию $f(X)$, имеющую в своей области определения несколько локальных минимумов, называют *многоэкстремальной* или *мультимодальной*.

Целевую функцию $f(X)$, $|X| > 1$, называют *овражной* в своей области определения, если в этой области имеют место слабые изменения первых частных производных функции $f(X)$ по одним направлениям и значительные изменения этих производных по другим направлениям. Широко известным примером овражной функции является функция Розенброка (Приложение Б).

Если целевая функция $f(X)$ представляет собой сумму функций, каждая из которых зависит только от одной компоненты вектора X, то есть если

$$f(X) = \sum_{i=1}^{|X|} f_i(x_i),$$

то такую функцию называют *сепарабельной*.

Рис. 1.1. Классификация целевых функций

Целевую функцию $f(X)$ называют *позиномиальной*, если

$$f(X)=\sum_{i=1}^{k} c_i p_i(X),$$

где c_i и все компоненты вектора X – положительные действительные числа, а функции $p_i(X)$ имеют вид

$$p_i(X)=\prod_{j=1}^{|X|} x_j^{a_{i,j}}.$$

Здесь $a_{i,j}$ – любые действительные числа.

Приведенную классификацию целевых функций иллюстрирует рис. 1.1.

1.1.2. Классификация задач оптимизации

Классификация задач вида (1.1), (1.2) возможна по многим признакам. Рассмотрим основные из них.

Вид целевой и ограничивающих функций. Если целевая функция $f(X)$ – линейная функция, а множество D – выпуклый многогранник, то задачу (1.1), (1.2) называют задачей *линейного программирования*.

В случае, когда целевая функция $f(X)$ представляет собой отношение двух линейных функций, а множество D – выпуклый многогранник, задача (1.1), (1.2) носит название задачи *дробно-линейного программирования*.

Пусть область D определяют только ограничениями типа неравенств:

$$D = \{X \mid G(X) \geq 0\}. \tag{1.3}$$

Если целевая функция $f(X)$ и ограничивающие функции $g_i(X), i \in [1:|G|]$ являются сепарабельными, то задачу (1.1), (1.3) называют задачей *сепарабельного программирования.*

В ситуации, когда целевая функция $f(X)$ и ограничивающие функции $e_i(X)$, $g_j(X)$, $i \in [1:|E|]$, $j \in [1:|G|]$ являются позиномами, задачу (1.1), (1.2) называют задачей *геометрического программирования.*

Если целевая функция $f(X)$ является выпуклой, то задача (1.1), (1.3) принимает название задачи *выпуклого программирования.*

При квадратичной целевой функции $f(X) = X^T AX + B^T X$ и выпуклом множестве D задача (1.1), (1.2) называется задачей *квадратичного программирования*. Здесь A – $(|X| \times |X|)$ -симметричная матрица, B – $(|X| \times 1)$ -вектор.

Конечное множество D порождает задачу *дискретного программирования*, а множество D в виде множества целых чисел – аналогичную задачу *целочисленного программирования.*

В общем случае задачу (1.1), (1.2) называют задачей *нелинейного программирования*. Часто задачи выпуклого программирования также относят к задачам нелинейного программирования.

Классификацию детерминированных задач оптимизации по виду целевой и ограничивающих функций, а также по некоторым другим перечисленным ниже признакам иллюстрирует рис. 1.2.

Наличие или отсутствие ограничений. Если ограничения на вектор X отсутствуют (то есть $D = R^{|X|}$), то задачу (1.1), (1.2) называют задачей *оптимизации без ограничений* или задачей *безусловной оптимизации.*

При наличии ограничений на вектор X (когда $D \subset R^{|X|}$) задача (1.1), (1.2) называется задачей *оптимизации с ограничениями* или задачей *условной оптимизации.*

Характер ограничений. Среди задач условной оптимизации выделяют следующие классы задач.

Если множество D имеет вид (1.3), то задача (1.1), (1.3) носит название задачи условной оптимизации *с ограничениями типа неравенств.*

В ситуации, когда множество D имеет вид

$$D = \{X \mid E(X) = 0\},$$

задача (1.1), (1.2) называется задачей условной оптимизации *с ограничениями типа равенств.*

Задачу (1.1), (1.2) называют задачей условной оптимизации *с ограничениями общего вида*, если множество D формируют как ограничения типа неравенств, так и ограничения типа равенств.

Размерность вектора X. Если размерность вектора X равна единице $(|X| = 1)$, то задача (1.1), (1.2) называется *однопараметрической* (*одномерной*) задачей оптимизации.

В случае, когда размерность вектора X больше единицы $(|X| > 1)$, задачу (1.1), (1.2) называют *многопараметрической* (*многомерной*).

Число точек минимума. Если целевая функция $f(X)$ имеет в области допустимых значений D один минимум, то задача (1.1), (1.2) носит название *одноэкстремальной*.

При наличии в области D более одного минимума целевой функции $f(X)$ задачу (1.1), (1.2) называют *многоэкстремальной*.

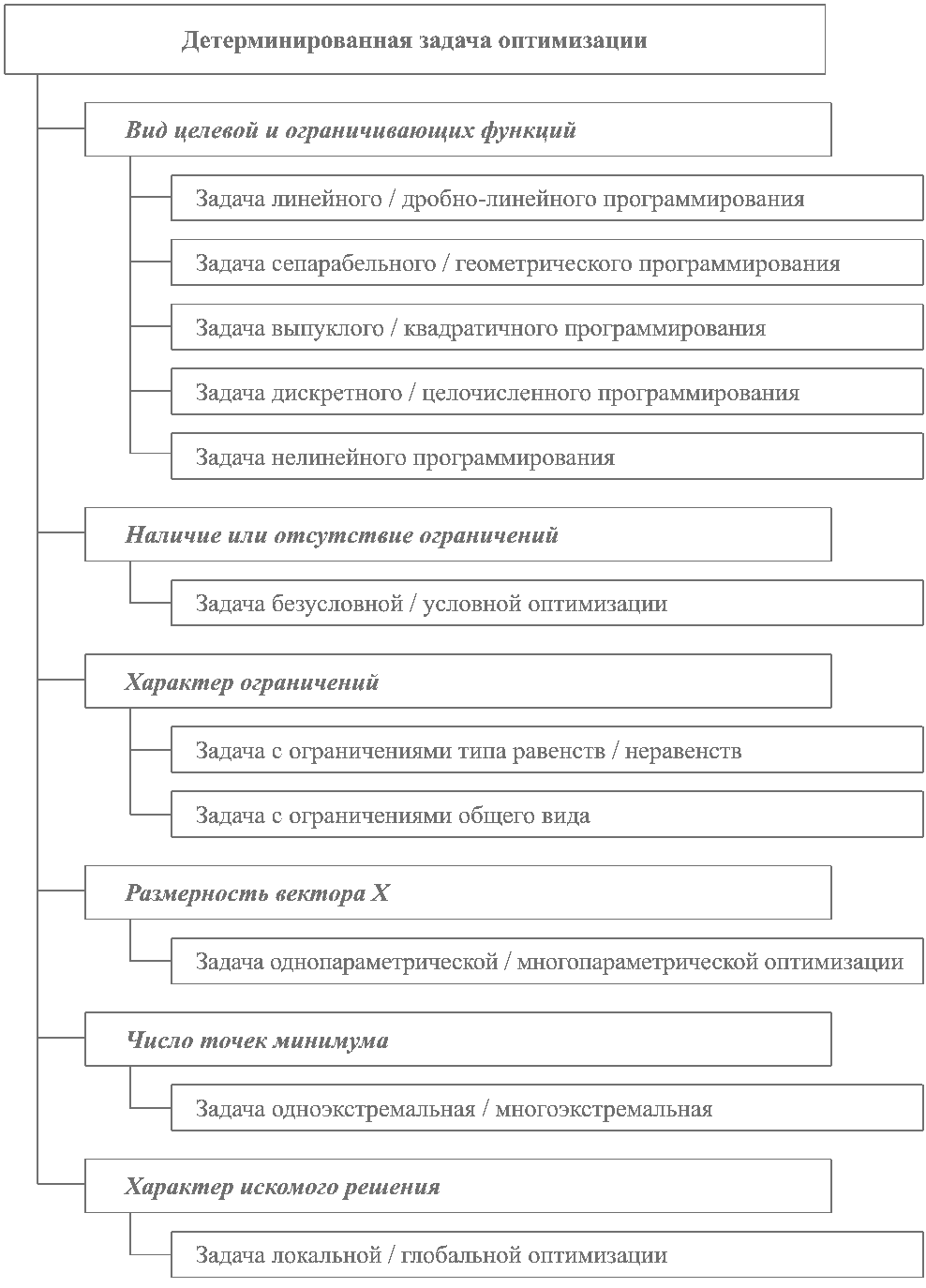


Рис. 1.2. Классификация детерминированных задач оптимизации

Характер искомого решения. Если отыскивается любой локальный минимум целевой функции $f(X)$ в области D, то задача (1.1), (1.2) называется задачей *локальной оптимизации*.

В случае, когда целью является поиск глобального минимума целевой функции $f(X)$ в той же области D, задачу (1.1), (1.2) называют задачей *глобальной оптимизации*.

1.1.3. Классификация алгоритмов оптимизации

Испытанием называют операцию однократного вычисления значений целевой функции $f(X)$ и ограничивающих функций $E(X)$, $G(X)$, а также, быть может, производных указанных функций в некоторой точке $X \in D$.

Особенность задач оптимизации, представляющих практический интерес, состоит в том, что каждое испытание может требовать больших затрат компьютерного времени. Поэтому одним из основных требований, предъявляемых к алгоритмам оптимизации, является решение задачи оптимизации при наименьшем числе испытаний.

Приближение к решению задачи (кандидата на решение) называем *агентом* и обозначаем s_i, $i \in [1:|S|]$, $|S| \geq 1$. Здесь S – используемое множество агентов (популяция агентов), $|S|$ – их общее число.

Ограничиваемся рассмотрением *итерационных алгоритмов* оптимизации, которые в вычислительной практике используют в подавляющем большинстве случаев. Общая схема итерационного алгоритма имеет следующий вид.

1) Инициализируем алгоритм – задаем начальное значение счетчика числа итераций $t = 0$, начальные положения агентов $X_i(0)$, $i \in [1:|S|]$, а также значения свободных параметров алгоритма.

2) Применяем поисковые (миграционные) операторы алгоритма к текущим положениям агентов $X_i(t) = X_i^t = X_i$, в результате чего они приобретают новые положения $X_i(t+1) = X_i^{t+1} = X_i'$; $i \in [1:|S|]$.

3) Проверяем выполнение условий завершения итераций. Если эти условия не выполнены, полагаем $t = t+1$ и возвращаемся к шагу 2. В противном случае принимаем лучшее из найденных положений агентов в качестве приближенного решения задачи.

С самой общей точки зрения миграционные операторы вычисляют новые положения агентов в соответствии с некоторой функцией $\Phi(\cdot)$:

$$X_i^{t+1} = \Phi\left(X_k^\tau, f(X_k^\tau), E(X_k^\tau), G(X_k^\tau),\ k \in [1:|S|], \tau \in [0:t]\right), i \in [1:|S|], t+1 \leq \hat{t}. \quad (1.4)$$

Здесь $\hat{t}$ – число итераций, использованных в процессе поиска. Смысл формулы заключается лишь в том, что новое положение агента s_i определяется, вообще говоря, всеми предыдущими положениями всех агентов популяции, а также соответствующими значениями целевой и ограничивающих функций.

В качестве приближенного решения задачи принимаем вектор $\tilde{X}^*$, который находится из условия

$$\tilde{f}^* = f(\tilde{X}^*) = \min_{t \in [0:\hat{t}],\ i \in [1:|S|]} f(X_i^t). \quad (1.5)$$

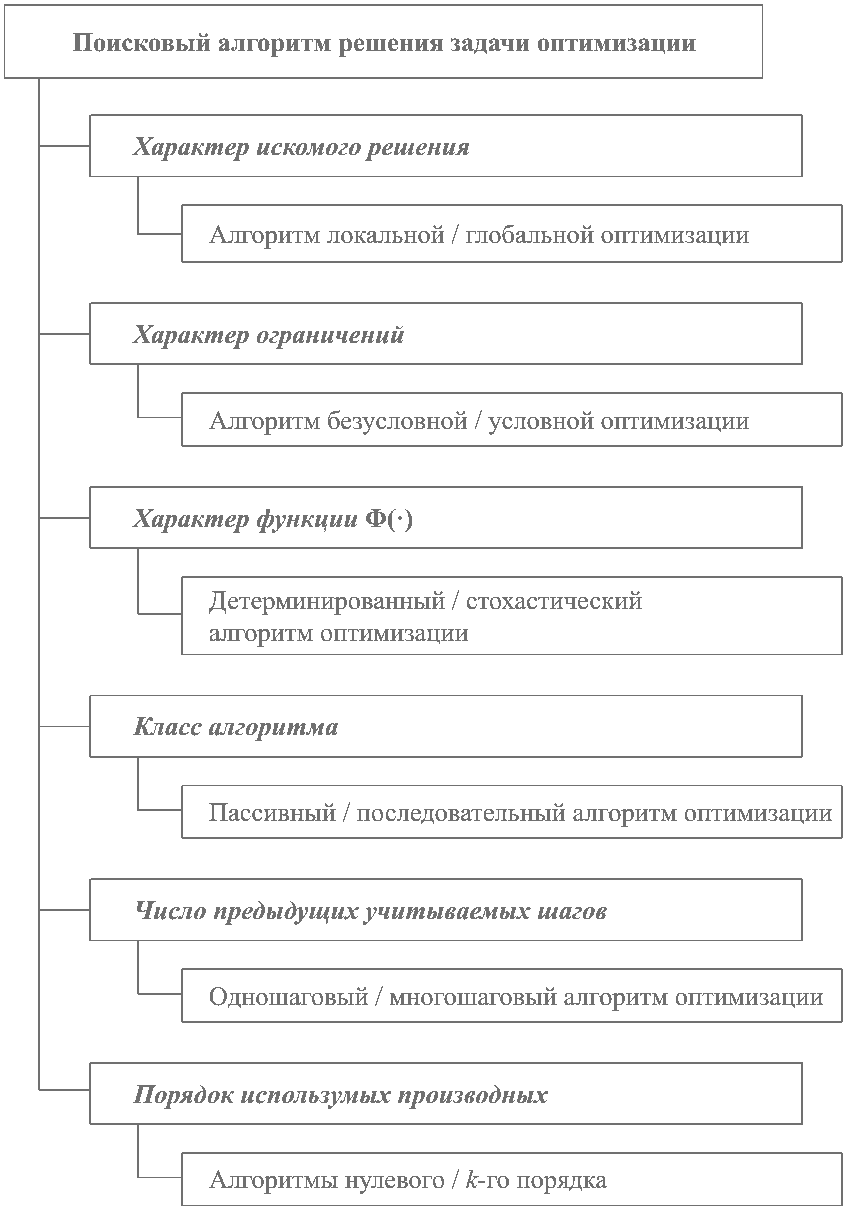


Рис. 1.3. Классификация поисковых алгоритмов оптимизации

В итерационных алгоритмах функция $\Phi(\cdot)$ является одной и той же на всех итерациях, исключая, может быть, лишь некоторое небольшое число начальных «разгонных» итераций.

Поисковые алгоритмы оптимизации можно классифицировать по следующим признакам (рис. 1.3).

Характер искомого решения. Алгоритм называют *локальным*, если схема поиска (1.4), (1.5) нацелена на отыскание локального минимума функции $f(X)$. В случае, когда имеется в виду отыскание глобального минимума этой функции, алгоритм называется *глобальным*.

Характер ограничений. Алгоритмом *безусловной оптимизации* называется алгоритм, ориентированный на решение задач безусловной оптимизации. Алгоритм *условной оптимизации* ориентирован на решение задач условной оптимизации.

Характер функции $\Phi(\cdot)$. Если функция $\Phi(\cdot)$ является детерминированной, то алгоритм называют *детерминированным*. В ситуации, когда функция $\Phi(\cdot)$ включает в себя один или несколько параметров, которые на различных итерациях принимают случайные значения с заданными законами распределения, алгоритм называется *стохастическим* или *случайным*.

Класс алгоритма. Алгоритм носит название *пассивного* алгоритма, когда все точки X_i^t, $t \in [1:\hat{t}], i \in [1:|S|]$ назначаются заранее (до проведения испытаний). Если точки X_i^t определяют на основе всей или части информации об испытаниях в предыдущих точках, то алгоритм называется *последовательным*.

Число предыдущих учитываемых шагов. В ситуации когда при вычислении координат точки X_i^{t+1} учитывают информацию только об одной (предыдущей) итерации, алгоритм поиска называется *одношаговым*. Общая схема одношагового алгоритма последовательного поиска имеет вид

$$X_i^{t+1} = \Phi\left(X_i^t, f(X_i^t), E(X_i^t), G(X_i^t)\right),\ t \in [1:\ \hat{t}-1],\ i \in [1:|S|].$$

Если при вычислении координат точки X^{t+1} учитывают информацию не более чем о $r > 1$ предыдущих испытаниях, то алгоритм поиска называется *многошаговым* (точнее, r-шаговым), или алгоритмом *с памятью*. Отметим, что это определение не требует, чтобы в r-шаговом алгоритме учитывалась информация обо *всех* r предыдущих испытаниях. Определение требует лишь, чтобы использовалась информация, как минимум, о r-ом испытании и не использовалась информация ни об одном из испытаний, предшествующих r-му.

Очевидно, что по своей сути r-шаговый алгоритм не может быть применен на первых $(r-1)$ шагах. На этих шагах (на этапе «разгона» алгоритма) должны быть использованы, например, одношаговые алгоритмы.

Порядок используемых производных. Алгоритм поиска называют *прямым,* или *алгоритмом нулевого порядка*, если при вычислении значений функции $\Phi(\cdot)$ производные целевой и ограничивающих функций не используются. Если в тех же условиях используют указанные производные k-го порядка, то алгоритм называется *алгоритмом k-го порядка*. Алгоритм поиска первого порядка называют также *градиентным*.

Выделяем еще два класса алгоритмов оптимизации – траекторные и популяционные.

Траекторные алгоритмы (*single-state algorithms*) предполагают обновление на каждой итерации положения лишь одного из агентов. При этом общее число агентов может быть больше единицы, и на разных итерациях могут перемещаться разные агенты. В соответствии с этим разделяем траекторные алгоритмы на *одноточечные и многоточечные*.

В *популяционных алгоритмах* (*population algorithms*), во-первых, число агентов больше единицы и, во-вторых, на каждой из итераций перемещаются все агенты, исключая, возможно, лишь некоторых из них (например, «лучших» агентов).

Важной проблемой при построении поисковых алгоритмов оптимизации является проблема выбора условий (критериев) окончания поиска (итераций). Простейшими, но широко используемыми в вычислительной практике являются критерии окончания поиска вида

$$\left\| X^{t+1} - X^{t} \right\| \le \varepsilon_X , \tag{1.6}$$

$$\left| f(X^{t+1}) - f(X^{t}) \right| \le \varepsilon_f , \tag{1.7}$$

где ε_X, ε_f – константы, определяющие требуемую точность решения по вектору варьируемых параметров и критерию оптимальности соответственно.

Условия окончания поиска (1.6), (1.7) называем *стандартными условиями*.

1.2. ЛОКАЛЬНАЯ БЕЗУСЛОВНАЯ ОПТИМИЗАЦИЯ

Последовательно рассматриваются три следующих класса стохастических алгоритмов локальной безусловной оптимизации: одноточечные одношаговые и многошаговые алгоритмы, многоточечные алгоритмы.

1.2.1. Одношаговые алгоритмы

Итерационную формулу одношагового алгоритма *случайного поиска* (*Random Search*, *RS*) можно записать в виде

$$X' = (1-\lambda)X + \lambda V , \tag{1.8}$$

где λ – положительная константа, имеющая смысл величины шага в направлении случайного ($|X| \times 1$)-вектора V. Точку X' принимаем, если $\varphi(X') < \varphi(X)$. В противном случае по некоторому правилу изменяем величину шага λ и/или направление поиска V.

В общем случае вектор X' является не простейшей линейной комбинацией (1.8) векторов X, V, а представляет собой некоторую детерминированную или стохастическую функцию этих векторов, так что

$$X' = \Phi(X, V) .$$

Правила выбора направления поиска

Одним из наиболее известных классов алгоритмов *RS* являются *рандомизированные алгоритмы стохастической аппроксимации* (*Simultaneous Perturbation Stochastic Approximation*, *SPSA*), в которых в качестве вектора V используют стохастическую аппроксимацию градиента фитнес-функции. Эти алгоритмы предложены в конце 80-х – начале 90-х годов прошлого века. Свойства их хорошо исследованы как в теоретическом плане, так и экспериментально. Доказана асимптотическая оптимальность скорости сходимости этих алгоритмов.

В самой общей форме итерационную формулу алгоритмов *SPSA* можно записать в виде

$$X' = X - \lambda\, \tilde{\nabla}_X\,, \tag{1.9}$$

где $\tilde{\nabla}_X = V$ – псевдоградиент (*pseudo-gradient*) фитнес-функции в точке X, то есть стохастическая аппроксимация градиента этой функции в указанной точке. Доказано, что итерационный процесс (1.9) сходится по вероятности к локальному минимуму функции $\varphi(X)$.

Различные алгоритмы *SPSA* отличаются друг от друга правилами вычисления псевдоградиента, которые имеют общий вид

$$\tilde{\nabla}_X = \frac{|X|}{n\,\varepsilon} \sum_{j=1}^{n} \Big(\varphi(X + \varepsilon\,\Psi_j) - \varphi(X)\Big)\,\Psi_j\,, \tag{1.10}$$

где Ψ_j – $(|X|\times 1)$-случайный вектор, равномерно распределенный в единичной сфере с центром в точке X; ε – фиксированная положительная константа; n – используемое число векторов Ψ_j. Важно, что величина n может превышать размерность пространства поиска $|X|$, быть равной этой размерности или меньше ее. Очевидно, что если $n = |X|$ и направления векторов Ψ_j, $j \in [1:n]$ совпадают с ортами системы координат, определяющей пространство поиска $R^{|X|}$, то вектор псевдоградиента совпадает с вектором градиента функции $\varphi(X)$.

Обычно в вычислительной практике используют простые правила вычисления псевдоградиента вида

$$(\tilde{\nabla}_X)_j = \frac{\varphi(X + \Psi) - \varphi(X - \Psi)}{2\psi_j}\,,\ \ j \in [1:|X|],$$

где $(\tilde{\nabla}_X)_j$ – j-я компонента вектора $\tilde{\nabla}_X$, Ψ – $(|X|\times 1)$-случайный вектор с компонентами $\psi_j = a\, u_{sign}^{\pm 1}$, $j \in [1:|X|]$; a – заданная положительная константа (свободный параметр алгоритма).

Замечание. Если функция $\varphi(X)$ недифференцируема в окрестности рассматриваемой точки X, то, конечно, говорить о векторе $\tilde{\nabla}_X$, как о векторе псевдоградиента этой функции, неверно. В этом случае используют, например, термины *направление убывания функции* и *аппроксимация направления убывания* функции (*Approximate Descent Direction*, *ADD*). Формально, вектор *ADD* определяется так же, как вектор псевдоградиента $\tilde{\nabla}_X$.

Иное правило определения вектора V использует *алгоритм линейного стохастического поиска* (*Random Line Search*, *RLS*). Алгоритм предложил Бремерман (*H. J. Bremermann*) в 1970 г. Компоненты указанного вектора в этом случае вычисляют по формуле вида

$$v_j = u_{sign}^{\pm 1} U_1(0;1)\,,\ \ j \in [1:|X|].$$

Величину шага λ в алгоритме *RLS* определяют по правилу

$$\lambda = \alpha \max_{j \in [1:|X|]} (x_j^+ - x_j^-)\,,$$

где x_j^-, x_j^+ – границы рассматриваемой области пространства поиска, $\alpha \in (0;1)$ – свободный параметр алгоритма.

В *алгоритме локальной унимодальной выборки* (*Local Unimodal Sampling*, *LUS*), который предложил Педерсен (*M.E.H. Pedersen*) в 2010 г., вектор V определяет формула

$$V = \frac{U}{\|U\|}, \; U = U_{|X|}(-1;1). \tag{1.11}$$

Автор позиционирует алгоритм *LUS* как простой и быстрый стохастический алгоритм численной оптимизации.

Алгоритм одномерного рандомизированного поиска (*Single-Dimension Perturbation Search*, *SDPS*), предложенный Педерсеном в 2010 г. как развитие алгоритма *LUS*, предполагает расщепление каждой из итераций схемы (1.8) на $|X|$ шагов. На шаге $j \in [1:|X|]$ вектор V в этом алгоритме имеет вид

$$V = \left(0, ..., 0, U_1(-1;1), 0, ..., 0\right),$$

где отлична от нуля только j-я компонента. Очевидно, что такое определение вектора V обеспечивает на j-м шаге поиск только вдоль j-го координатного направления (почему алгоритм и называется одномерным).

Вместо равномерного распределения в последних алгоритмах могут быть использованы нормальные распределения $N_{|X|}(0;\sigma)$, $N_1(0;\sigma)$ соответственно. Могут быть использованы и более сложные распределения. Так, *адаптивный алгоритм случайного поиска с интенсификацией и диверсификацией* (*Adaptive Random Search with Intensification and Diversification*, *ARSID*) предполагает генерацию вектора V на основе двухпараметрической функции плотности вероятности

$$\xi(v) = \begin{cases} (1-a)\, b\, e^{bv}, & v \le 0, \\ a\, b\, e^{-bv}, & v > 0, \end{cases}$$

где $a \in [0;1]$, b – свободные параметры, управляющие соотношением интенсификационных и диверсификационных свойств алгоритма.

Значительно более сложные правила формирования вектора V использует эвристический алгоритм на основе паттернов (*Heuristic Pattern Search, HPS*). Алгоритм предполагает вычисление вектора псевдоградиента по формуле, несколько отличной от формулы (1.10),

$$\tilde{\nabla}_X = \sum_{j=1}^{n} \frac{\Delta\varphi_j}{\sum_{k=1}^{n} |\Delta\varphi_k|} a_j \;,$$

где

$$\Delta\varphi_j = \varphi(X_j) - \varphi(X), \; a_j = \frac{X_j - X}{\|X_j - X\|};$$

$$X_j = X + \varepsilon\, \Psi_j .$$

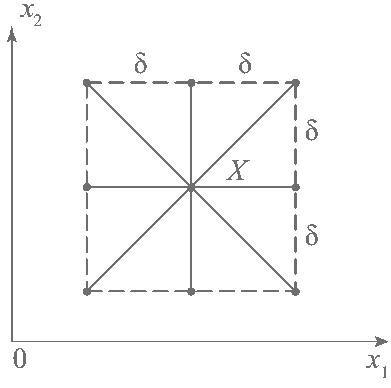


Рис. 1.4. К определению паттерна $P_\delta(X)$: $|X| = 2$

Паттерном $P_\delta(X) = P$ называем набор точек X_k^P, $k \in |P|$, расположенных на расстоянии δ от точки X вдоль каждого из координатных направлений, а также на диагоналях соответствующего куба (рис. 1.4).

Введем еще набор точек

$$P^{add} = \begin{cases} \left\{ Y \in P \mid (Y, \tilde{\nabla}_X) \le b \|Y\| \left\| \tilde{\nabla}_X \right\| \right\}, & \varphi(X - c\,\tilde{\nabla}_X) < \varphi(X), \\ \left\{ Y \in P \mid (Y, \tilde{\nabla}_X) \ge -b \|Y\| \left\| \tilde{\nabla}_X \right\| \right\}, & \varphi(X - c\,\tilde{\nabla}_X) \ge \varphi(X), \end{cases}$$

смысл которого в следующем. Если направление $\tilde{\nabla}_X$ является направлением *ADD*, то в набор P^{add} включаем все те точки Y паттерна P, для которых косинус угла между векторами Y, $\tilde{\nabla}_X$ не превышает величины b. В противном случае, то есть когда направление $\tilde{\nabla}_X$ ведет к возрастанию значения фитнес-функции, в указанный набор включаем точки паттерна P, для которых косинус угла между векторами Y, $\tilde{\nabla}_X$ отрицателен и по модулю не превышает ту же величину b. Здесь $b \in (-1; 1)$, $c > 0$ – свободные параметры алгоритма.

Итого, в качестве вектора V выбирают тот вектор текущего набора P^{add}, который обеспечивает минимальное значение функции $\varphi(X)$, т. е. полагают, что

$$V = \arg \min_{Y \in P^{add}} \varphi\left(X + \lambda Y\right).$$

Правила выбора величины шага

В случае постоянства величины шага λ алгоритм *RS* называется *неадаптивным алгоритмом случайного поиска* (*Fixed Step Size Random Search*, *FSSRS*), а в противном случае – *адаптивным алгоритмом случайного поиска* (*Adaptive Step Size Random Search*, *ASSRS*).

Известно большое число правил изменения шага поиска в алгоритмах *ASSRS*. В простейшем случае величина этого шага отыскивается из условия минимума функции $\varphi(X)$ в направлении V.

В алгоритме *LUS* в случае неудачного шага (когда не выполнено условие $\varphi(X') < \varphi(X)$) величину шага λ уменьшают по закону геометрической прогрессии $\lambda = v\,\lambda$, где $v \in (0; 1)$ – свободный параметр алгоритма.

В алгоритме *SDPS* в тех же условиях шаг уменьшают по линейному закону $\lambda = \lambda_0 \dfrac{\hat{t} - t}{\hat{t}}$, где λ_0 – свободный параметр.

Адаптивный максимизирующий алгоритм случайного поиска (*Adaptive Maxing Random Search*, *AMRS*), который предложил Забинский (*Z. B. Zabinsky*) с соавторами в 1990 г., использует двухэтапный выбор величины шага λ. На первом этапе определяют максимально допустимую величину шага $\lambda_{\max}$, обеспечивающую допустимость (принадлежность области поиска) всех точек вида $X' = X + \lambda V$, где $\lambda \in (0; \lambda_{\max}]$; V – определенный ранее вектор направления поиска. На втором этапе в качестве шага λ генерируют случайную величину $U_1(0^+; \lambda_{\max})$; 0^+ – малая положительная константа.

Алгоритм с возвратом при неудачном шаге

Рассмотрим более детально в качестве примера одношаговых алгоритмов *алгоритм с возвратом при неудачном шаге*. Алгоритм использует итерационную формулу вида

$$X' = X + \lambda V, \tag{1.12}$$

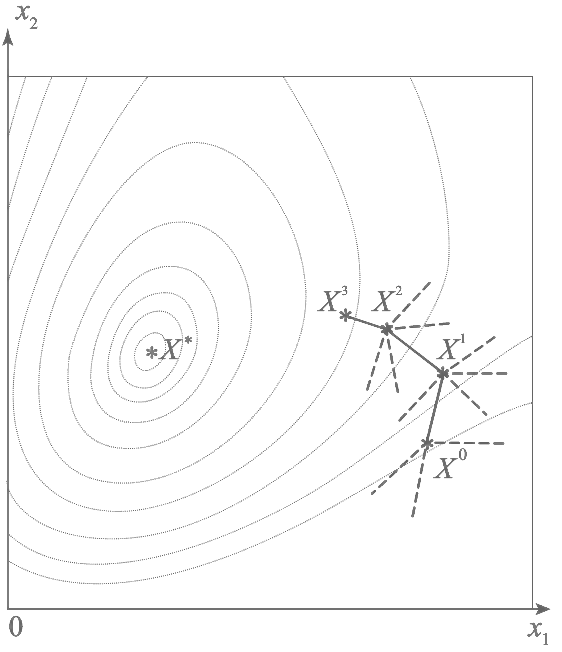


Рис. 1.5. К алгоритму с возвратом при неудачном шаге: фрагмент траектории поиска минимума функции: $|X|=2$

где вектор V определяется так же, как в алгоритме *LUS*. Свободными параметрами алгоритма являются начальная длина шага λ^0, предельное число неудачных попыток m (рекомендуемое значение равно $3|X|$), коэффициент уменьшения шага $\nu \in (0;1)$. Схема алгоритма имеет следующий вид.

1) Задаем начальную точку X^0, начальные значения параметров λ^0, m, ν и полагаем счетчик числа итераций $t=0$.

2) Задаем начальное значение счетчика числа неудачных попыток k=1.

3) Получаем реализацию вектора случайных чисел $U_{|X|}(-1;1)$, по формуле (1.12) находим пробную точку X' и вычисляем величину $\varphi(X')$.

4) Если $\varphi(X') < \varphi(X)$, то полагаем $t=t+1$ и переходим к шагу 2.

5) Если $k \le m$, то полагаем $k=k+1$ и переходим к шагу 3.

6) Проверяем условие окончания поиска. Если это условие выполнено, то полагаем $\tilde{X}^* = X^t$ и завершаем итерации, иначе полагаем $\lambda = \nu\lambda$, $t=t+1$ и переходим к шагу 2.

В качестве условия окончания итераций может быть использовано одно из стандартных условий окончания итераций. Алгоритм иллюстрирует рис. 1.5, где принято $m=5$ и пунктиром показаны отвергнутые шаги.

1.2.2. Многошаговые алгоритмы

В простейшем случае итерационную формулу r-шагового алгоритма случайного поиска определяет формула

$$X^{t+1} = \sum_{\tau=0}^{r-1} \lambda_\tau X^{t-\tau} + \lambda_\nu V^{t+1}, \; t > r,$$

где λ_τ – положительные константы, имеющие смысл весов соответствующих предыдущих шагов поиска, λ_ν – величина шага в направлении случайного $(|X|\times 1)$-вектора V^{t+1}. Как и в одношаговых алгоритмах, точку X^{t+1} принимаем, если $\varphi(X^{t+1}) < \varphi(X^t)$. В противном случае по некоторому правилу изменяем величину шага λ_ν и/или направление поиска V^{t+1}.

В общем случае вектор X^{t+1} представляет собой некоторую детерминированную или стохастическую функцию векторов $X^t, X^{t-1},..., X^{t-r+1}$, V^{t+1}:

$$X^{t+1} = \Phi(\ X^t, X^{t-1},..., X^{t-r+1}, V^{t+1})\ . \tag{1.13}$$

В качестве примера многошаговых алгоритмов рассмотрим трехшаговый *алгоритм повторяющегося случайного поиска*. Алгоритм использует итерационную схему вида (1.13)

$$X^{t+1} = X^t + \lambda\, V^{t+1}\ , \tag{1.14}$$

которую иллюстрирует рис. 1.6. Вектор направления шага V^{t+1} в этом случае определяет формула

$$V^{t+1} = \left[\ a\frac{W^{t+1}}{\left\|W^{t+1}\right\|} + (1-a)\frac{U^{t+1}}{\left\|U^{t+1}\right\|}\ \right], \tag{1.15}$$

где $W^{t+1} = b\,V^t + (1-b)\,V^{t-1}$ – $(|X|\times 1)$-вектор «предыстории» (среднее направление поиска на двух предыдущих шагах); $a \in [0;1]$ – коэффициент, задающий относительные веса детерминированной и случайной компонент в векторе V^{t+1} (свободный параметр алгоритма); $b \in [0;1]$ – коэффициент, задающий относительные веса векторов V^t, V^{t-1} в векторе W^{t+1} (свободный параметр алгоритма); $U^{t+1} = U_{|X|}(-1;1)$ – случайный $(|X|\times 1)$-вектор.

Рисунок 1.6 соответствует случаю $a = b = 0{,}5$; $\lambda = 2$, так что имеют место равенства

$$W^{t+1} = 0{,}5\,(V^t + V^{t-1})\ ,\ X^{t+1} = X^t + \left(\bar{W}^{t+1} + \bar{U}^{t+1}\right),$$

где $\bar{W}^{t+1}$, $\bar{U}^{t+1}$ – единичные векторы направлений W^{t+1}, U^{t+1} соответственно.

Суть алгоритма повторяющегося случайного поиска передает следующая последовательность его шагов.

1) Задаем начальную точку X^0, начальный шаг λ, максимально допустимое число неудачных попыток m, значения коэффициентов a, b, $\nu > 1$, $\mu < 1$ и полагаем счетчик числа итераций $t = 2$.

2) С помощью какого-либо одношагового алгоритма определяем точки X^1, X^2 – этап «разгона» алгоритма.

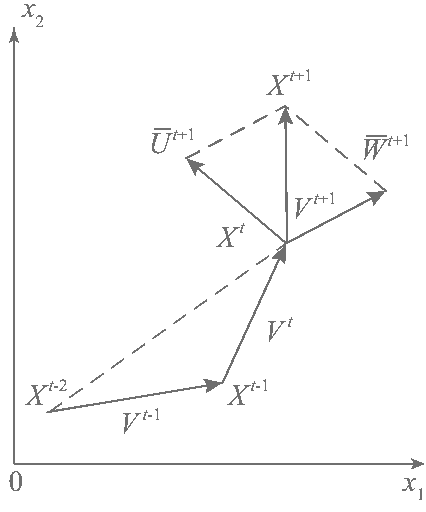


Рис. 1.6. К итерационной схеме алгоритма повторяющегося случайного поиска: $|X| = 2$

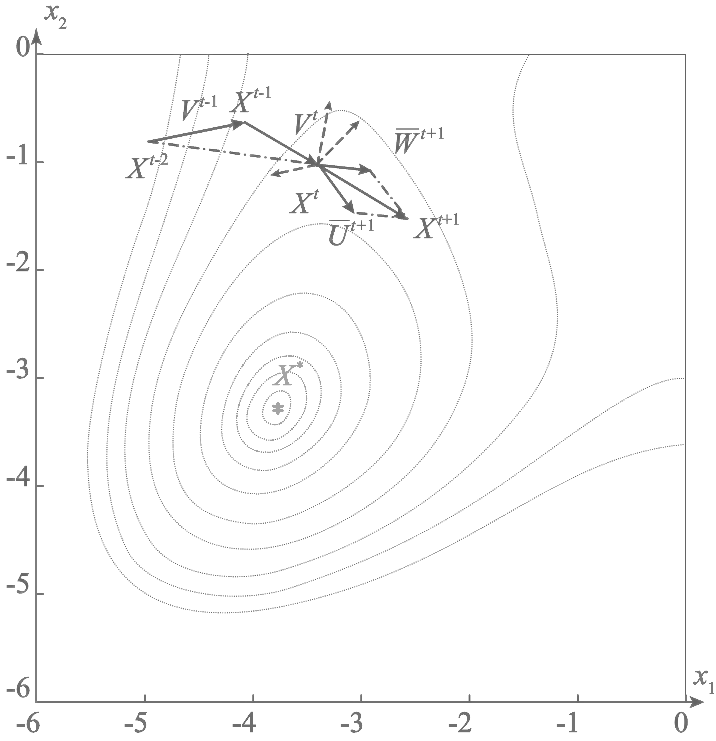


Рис. 1.7. Фрагмент траектории поиска минимума функции Химмельблау алгоритмом повторяющегося случайного поиска: $|X|=2$

3) Генерируем случайный вектор $U_{|X|}(-1;1)$ и по формулам (1.14), (1.15) вычисляем координаты точки X^{t+1} и значение фитнес-функции в этой точке $\varphi(X^{t+1})$.

4) Если $\varphi(X^{t+1})<\varphi(X^t)$ и условие окончания итераций выполнено, то полагаем $\tilde{X}^*=X^{t+1}$ и завершаем итерации. Если $\varphi(X^{t+1})<\varphi(X^t)$ и условие окончания не выполнено, то по некоторому правилу увеличиваем длину шага λ, например, полагая $\lambda=\nu\lambda$, принимаем $t=t+1$ и переходим к шагу 3.

5) Если $\varphi(X^{t+1})\geq\varphi(X^t)$, то, исходя из точки X^t, делаем не более m попыток добиться уменьшения значения функции $\varphi(X)$ путем только изменения вектора $U_{|X|}(-1;1)$ (не меняя величину λ) и переходим к шагу 3. Если это фиксированное число попыток не привело к успеху, то по некоторому правилу уменьшаем длину шага λ, например, полагая $\lambda=\mu\lambda$, и переходим к шагу 3.

В качестве критерия окончания поиска можно использовать одно из стандартных условий окончания итераций.

Известно большое число модификаций рассмотренной простейшей схемы алгоритма повторяющегося случайного поиска. Например, в процессе поиска могут изменяться по некоторым правилам не только длина шага λ, но и значения свободных параметров.

Алгоритм иллюстрирует рис. 1.7, на котором, как и на рис. 1.6, принято $a=0{,}5$, $b=0{,}5$, $\lambda=2$. Пунктиром показаны отвергнутые векторы $\overline{U}$.

1.2.3. Многоточечные алгоритмы

Рассматриваем в качестве примера два многоточечных алгоритма локальной безусловной оптимизации – алгоритм комплексов и алгоритм гиперсферы.

Алгоритм комплексов (конфигураций)

Комплексом в пространстве $R^{|X|}$ называют многогранник с $n > (|X|+1)$ вершинами, рекомендованное число которых равно $n = 2|X|$. Вообще говоря, комплексом в комбинаторной топологии называется геометрическая фигура, которая может быть разбита на более элементарные фигуры. В нашем случае такими элементарными фигурами являются симплексы. Поэтому, говоря более строго, рассматриваем *симплициальные комплексы*. Число используемых точек в алгоритме равно числу вершин комплекса n.

При решении задачи безусловной локальной оптимизации алгоритмом комплексов используют следующие операции:

– генерация комплекса;

– отражение вершины комплекса с растяжением;

– сжатие комплекса.

Генерация комплекса (рис. 1.8). Координаты вершин случайного комплекса с n вершинами определяют по формуле

$$X_i = X^0 + \lambda \frac{U}{\|U\|} = X^0 + \lambda \overline{U}_i,\ U = U_{|X|}(-1;1),\ i \in [1:n]. \qquad (1.16)$$

Отражение вершины комплекса с растяжением. Рассмотрим текущий комплекс $C = \{X_i, i \in [1:n]\}$. Положим, что его вершину X_j необходимо отразить через центр тяжести комплекса с растяжением. В новом комплексе C' все вершины, кроме *j*-й, совпадают с соответствующими вершинами исходного комплекса C, а *j*-я вершина находится на прямой, проходящей через центр тяжести этого комплекса и его вершину X_j (рис. 1.9):

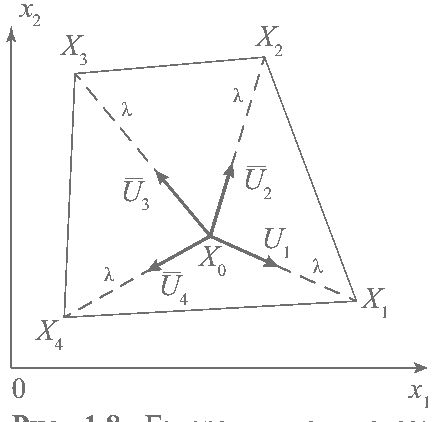


Рис. 1.8. Генерация комплекса: $|X| = 2$; $n = 4$

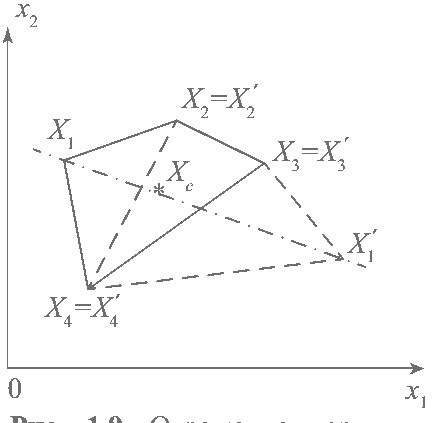


Рис. 1.9. Отражение вершины X_1 комплекса C через центр его тяжести X_c с растяжением: $|X| = 2$; $n = 4$

$$C' = \{X'_i = X_i, i \in [1:n], i \neq j\ ;\ X'_j = X_c + \alpha\,(X_c - X_j)\}\,. \qquad (1.17)$$

Здесь α – коэффициент растяжения (рекомендуемое значение равно 1,3), X_c – вектор координат центра тяжести комплекса C, определяемый формулой

$$X_c = \frac{1}{n}\sum_{i=1}^{n} X_i\,.$$

Сжатие комплекса. Положим, что вершину X_j комплекса $C = \{X_i, i \in [1:n]\}$ необходимо переместить ближе к его центру тяжести – выполнить сжатие комплекса. В новом комплексе C' все вершины, кроме j-й, совпадают с соответствующими вершинами исходного комплекса C, а j-я вершина находится на прямой, проходящей через центр тяжести этого комплекса и его вершину X_j (рис. 1.10):

$$C' = \{X'_i = X_i, i \in [1:n], i \neq j\ ;\ X'_j = X_c + \beta\,(X_j - X_c)\}\,. \qquad (1.18)$$

Здесь β – коэффициент сжатия (рекомендуемое значение равно 0,5), X_c – вектор координат центра тяжести комплекса C.

Параметры α, β являются свободными параметрами алгоритма.

Схема алгоритма комплексов может быть представлена в следующем виде.

1) Задаем начальную точку X^0, начальное значение величины λ и полагаем счетчик числа итераций $t = 0$.

2) Генерируем n случайных векторов $U_{|X|}(-1;1)$ и по формуле (1.16) находим координаты вершин комплекса C^0.

3) Вычисляем величины $\varphi(X_i)$, $i \in [1:n]$.

4) Находим максимальную из этих величин

$$\varphi(X_j) = \max_{i \in [1:n]} \varphi(X_i)\,.$$

5) По формуле (1.17) отражаем вершину X_j комплекса C – получаем вершину X'_j и новый комплекс C'. Вычисляем величину $\varphi(X'_j)$.

6) Если $\varphi(X'_j) < \varphi(X_j)$, то переходим к шагу 7. В противном случае, по формуле (1.18) выполняем сжатие симплекса C' в направлении $(X'_j - X'_c)$, получаем вершину X''_j и новый комплекс C''; полагаем $t = t + 2$ и переходим к шагу 4.

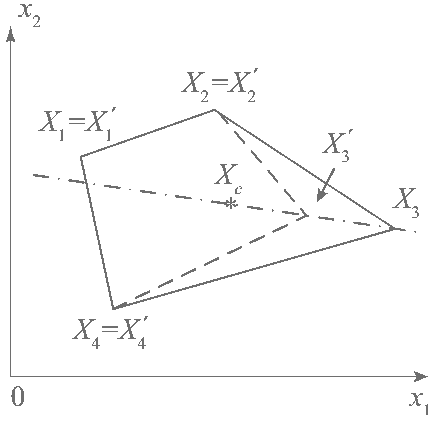


Рис. 1.10. Сжатие вершины X_3 комплекса C: $|X| = 2$; $n = 4$

7) Если условие окончания поиска выполнено, то в качестве точки $\tilde{X}^*$ принимаем вершину текущего комплекса C, в которой функция $\varphi(X)$ имеет наименьшее значение, и завершаем итерации. В противном случае полагаем $t = t + 1$ и переходим к шагу 4.

В качестве критерия окончания поиска могут использоваться следующие условия:

– максимальная длина ребра комплекса C не превышает ε_X – требуемую точность решения по X;

– максимальная разность значений функции $\varphi(X)$ в двух вершинах комплекса C не превышает ε_φ – требуемую точность решения по φ.

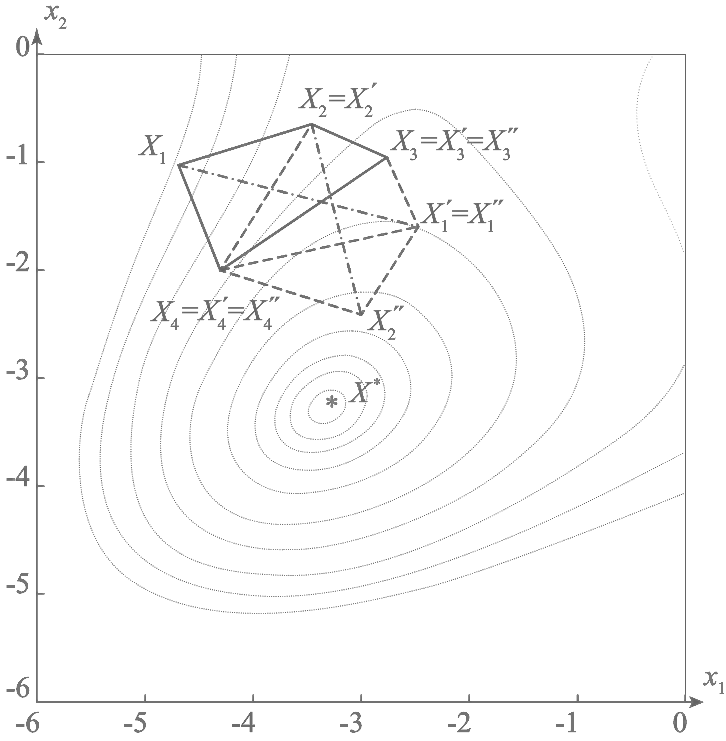


Рис. 1.11. Фрагмент траектории поиска минимума целевой функции Химмельблау алгоритмом комплексов: $|X| = 2;\ n = 4$

Могут использоваться также более сложные условия окончания поиска, например условия

$$\frac{1}{n}\sum_{i=1}^{n}\left\|X_i - X_c\right\|_E \le \varepsilon_X, \tag{1.19}$$

$$\frac{1}{n}\sqrt{\sum_{i=1}^{n}(\varphi(X_i)-\varphi_c)^2} \le \varepsilon_\varphi. \tag{1.20}$$

В формуле (1.19) евклидова векторная норма означает расстояние вершины X_i до центра тяжести комплекса C, а сама эта формула определяет среднее расстояние вершин комплекса C до его центра тяжести. В формуле (1.20) величина $\varphi_c = \varphi(X_c)$ представляет собой значение фитнес-функции в центре тяжести комплекса C, а сама формула определяет среднее отклонение значений функции $\varphi(X)$ в вершинах комплекса C от ее «центрального» значения.

Известно множество модификаций алгоритма комплексов, направленных, в частности, на преодоление *уплощения комплекса* в процессе поиска. С этой целью через фиксированное число итераций находят максимальную и минимальную диагонали комплекса и, если их отношение превышает заданное, то по рассмотренной схеме производят построение нового комплекса.

Алгоритм комплексов иллюстрирует рис. 1.11. Здесь после отражения с растяжением вершины X_1 комплекса C получен комплекс C'; после от-

ражения с растяжением вершины X_2' последнего комплекса получен комплекс C''.

Отметим, что из числа рассмотренных операций над комплексом только операция генерации начального комплекса является стохастической. Однако из-за явления уплощения комплекса эта операция в процессе функционирования алгоритма выполняется, вообще говоря, многократно. Это обстоятельство и позволяет классифицировать алгоритм как стохастический.

Алгоритм гиперсферы

Алгоритм гиперсферы (*алгоритм случайного поиска с постоянным радиусом поиска и случайными направлениями*) использует процедуру генерации случайных точек, равномерно распределенных по поверхности гиперсферы в пространстве $R^{|X|}$. Схему алгоритма гиперсферы определяют следующие его шаги.

1) Задаем начальную точку X^0, начальный радиус гиперсферы ρ^0, число точек на гиперсфере n и полагаем счетчик числа итераций $t = 0$.

2) Генерируем случайные точки X_i^t, $i \in [1:n]$, равномерно распределенные по поверхности гиперсферы радиуса ρ^t с центром в точке X^t.

3) Вычисляем значения функции $\varphi(X_i^t)$ в полученных точках и находим точку, в которой она достигает минимального значения

$$\varphi(X_j^t) = \min_{i \in [1:n]} \varphi(X_i^t).$$

4) Каким-либо из алгоритмов одномерной минимизации находим минимум функции $\varphi(X)$ в направлении $(X_j^t - X^t)$ и найденную точку минимума используем в качестве точки X^{t+1}:

$$\varphi(X^{t+1}) = \min_{\lambda \in R^1} \varphi(X^t + \lambda(X_j^t - X^t)).$$

5) Если условие окончания поиска выполнено, то полагаем $\tilde{X}^* = X^{t+1}$ и завершаем итерации. Иначе принимаем $t = t + 1$ и переходим к шагу 2.

Известно большое число модификаций алгоритма. Например, можно не отыскивать минимум функции $\varphi(X)$ в направлении $(X_j^t - X^t)$, а использовать шаг λ фиксированной длины в этом направлении. В процессе поиска радиус гиперсферы может меняться, увеличиваясь при удачных шагах (вдали от точки X^*) и уменьшаясь при неудачных шагах (вблизи от точки X^*). Поиск может быть ускорен, если точки на гиперсфере выбирать (случайным образом) в некотором секторе по отношению к предыдущему направлению. Угол раскрыва этого сектора может меняться в процессе поиска.

В качестве критерия окончания поиска можно использовать одно из стандартных условий окончания итераций. Могут быть использованы также другие критерии, например текущий радиус гиперсферы не должен превышать некоторой величины ε_X:

$$\rho^t \le \varepsilon_X.$$

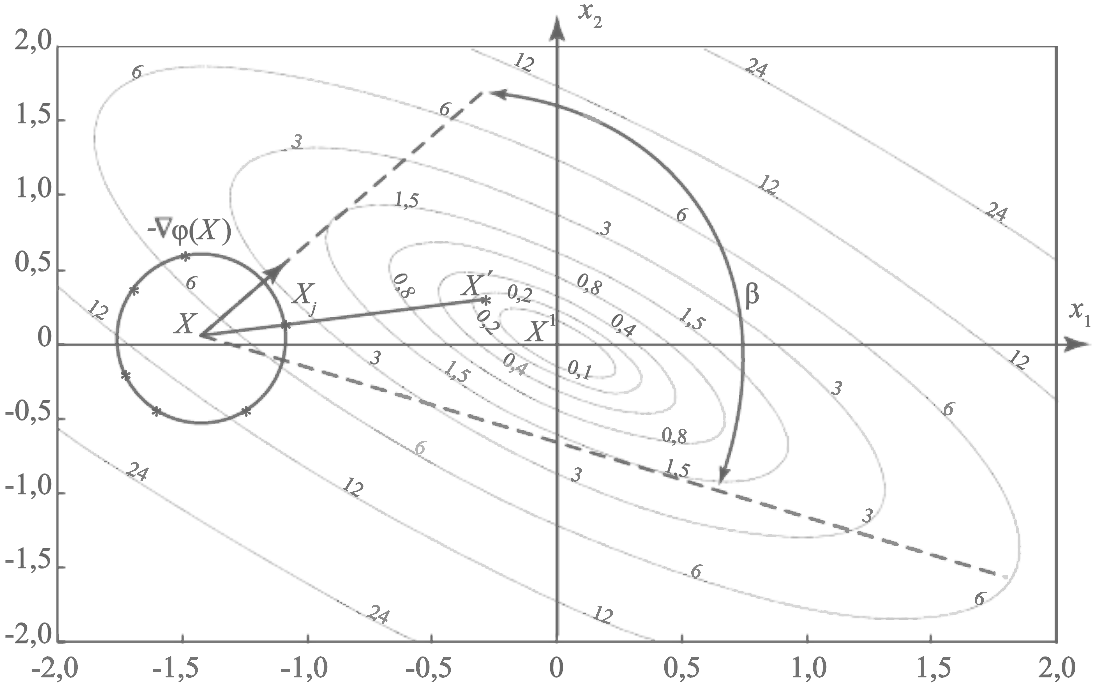


Рис. 1.12. Фрагмент траектории поиска минимума функции алгоритмом гиперсферы: $|X| = 2$

Алгоритм гиперсферы иллюстрирует рис. 1.12, на котором линии уровня соответствуют двумерной квадратичной функции

$$\varphi(x_1, x_2) = x_1^2 + x_2^2 + 3(x_1 + x_2)^2 = 4x_1^2 + 4x_2^2 + 6x_1x_2 .$$

Одна итерация алгоритма гиперсферы может привести к уменьшению минимизируемой функции в большей степени, чем один шаг поиска в направлении антиградиента этой функции. Данное утверждение также иллюстрирует рис. 1.12. Один шаг поиска в направлении антиградиента $\nabla\varphi(X)$ указанной функции приводит на линию уровня 1,5. Поскольку любое направление поиска в секторе β является лучшим, чем направление антиградиента, одна итерация метода гиперсферы может привести на линию уровня с меньшим значением функции $\varphi(X)$ (в примере на рис. 1.12 – на линию уровня ~0,2).

1.3. ЛОКАЛЬНАЯ УСЛОВНАЯ ОПТИМИЗАЦИЯ

Выделяем два класса алгоритмов решения задачи условной оптимизации:

– алгоритмы, основанные на сведении задачи условной оптимизации к задаче безусловной оптимизации;

– алгоритмы, не использующие указанный переход к задаче безусловной оптимизации.

1.3.1. Алгоритмы на основе сведения задачи условной оптимизации к задаче безусловной оптимизации

Представляем два алгоритма сведения задачи условной оптимизации к задаче безусловной оптимизации – алгоритм штрафных функций и алгоритм скользящего допуска.

Алгоритмы штрафных функций

Алгоритмы штрафных функций относят к классу *алгоритмов последовательной безусловной оптимизации*. Рассмотрим прежде общую схему алгоритмов этого класса.

Положим, что множество допустимых значений D вектора варьируемых параметров определяют ограничения как типа равенств, так и типа неравенств. Обычно в задачах условной оптимизации требуют лишь приближенного выполнения указанных ограничений, т. е. используют ограничения вида

$$\left|e_i(X)\right| \le \varepsilon_i^e,\ g_j(X) \ge \varepsilon_j^g,\ i \in [1:|E|],\ j \in [1:|G|], \tag{1.21}$$

где $\varepsilon_i^e > 0$, $\varepsilon_j^g > 0$ – требуемые точности выполнения ограничений $e_i(X) = 0$, $g_j(X) \ge 0$ соответственно.

Суть алгоритмов последовательной безусловной оптимизации состоит в преобразовании задачи условной оптимизации к последовательности задач безусловной оптимизации:

$$\min_{X \in R^{|X|}} \varphi(X,\alpha) = \min_{X \in R^{|X|}} \left(f\left(X\right) + p\left(X,\alpha\right)\right) = \varphi\left(\tilde{X}^*,\alpha\right),\ \alpha = const,\ \alpha = \alpha_1, \alpha_2, ..., \tag{1.22}$$

где $p(X,\alpha)$ – *функция штрафа*, возрастающая вблизи границ области допустимых значений D, и тем быстрее, чем больше значение вещественного *параметра штрафной функции* α. В качестве приближенного решения исходной задачи принимают решение $\tilde{X}^*$ вспомогательной задачи (1.22) при достаточно большом значении величины α. Заметим, что функция $\varphi(X,\alpha)$ здесь имеет смысл фитнес-функции.

Среди алгоритмов последовательной безусловной оптимизации выделяют алгоритм штрафных функций и алгоритм барьерных функций. Чаще всего в вычислительной практике используют первый из этих алгоритмов.

В *алгоритме штрафных функций* функцию $p(X,\alpha)$ подбирают таким образом, чтобы при больших значениях величины α функция $\varphi(X,\alpha)$ мало отличалась от целевой функции $f(X)$, когда $X \in D$, и быстро возрастала при удалении точки $X \notin D$ от границы области допустимых значений D (рис. 1.13). Заметим, что в силу указанного поведения функции $\varphi(X,\alpha)$ в алгоритме штрафных функций точка X в процессе поиска может выходить за границы области D.

Штрафная функция в общем случае имеет вид

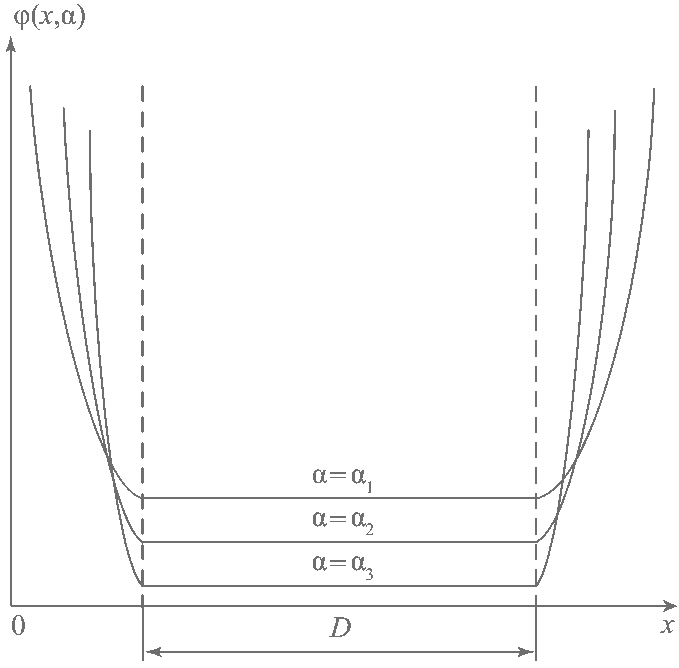


Рис. 1.13. К алгоритму штрафных функций: $|X|=1$; $\alpha_3 > \alpha_2 > \alpha_1$

$$p(X,\alpha)=\alpha_e\sum_{i=1}^{|E|}\mu_i P_e\left(e_i(X)\right)+\alpha_g\sum_{j=1}^{|G|}\lambda_j P_g\left(g_j(X)\right).$$

Здесь приняты следующие обозначения: $\alpha=(\alpha_e,\alpha_g)$ – двумерный вектор параметров штрафной функции; $\mu=(\mu_1,\mu_2,...,\mu_{|E|})$, $\lambda=(\lambda_1,\lambda_2,...,\lambda_{|G|})$ – векторы, компоненты которых позволяют изменять относительное влияние на функцию $p(X,\alpha)$ различных ограничивающих функций; $P_e(e_i(X))$, $P_g(g_j(X))$ – функционалы над ограничивающими функциями $e_i(X)$, $g_j(X)$, удовлетворяющие условиям

$$\begin{cases} P_e(e_i(X))\to 0, & e_i(X)\to 0,\\ P_e(e_i(X))=0, & e_i(X)=0;\end{cases}$$

$$\begin{cases} P_g(g_j(X))\to 0, & g_j(X)\to 0^-,\\ P_g(g_j(X))=0, & g_j(X)\ge 0.\end{cases}$$

В последнем выражении запись $g_j(X)\to 0^-$ означает такое стремление функции $g_j(X)$ к нулю, при котором значения этой функции остаются отрицательными (когда точка X находится вне области D).

В качестве функционалов $P_e(e_i(X))$, $P_g(g_j(X))$ естественно было бы использовать расстояния в какой-либо метрике от точки X до соответствующей границы множества D. Однако вычисление этих расстояний чаще всего затруд-

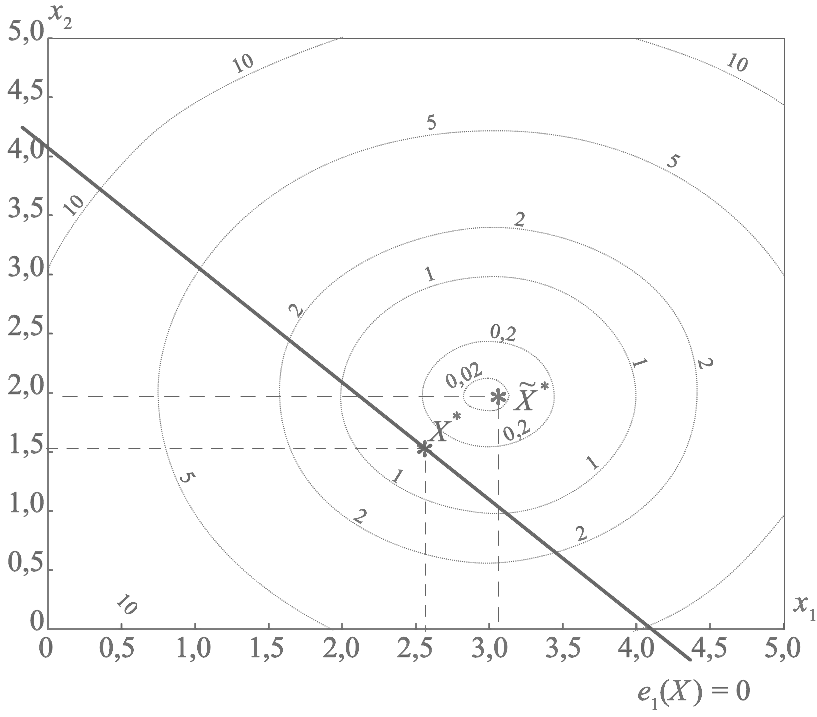


Рис. 1.14. Линии уровня функции $\varphi(X,\alpha)$: $\alpha = 0$

нительно. Поэтому обычно применяют функционалы более удобного в вычислительном плане вида, например, функционалы

$$P_e(e_i(X)) = e_i^2(X)\,,\ i \in [1:|E|]\,,$$

$$P_g(g_j(X)) = (g_j^+(X))^2\,,\ j \in [1:|G|]\,,$$

где

$$g_j^+(X) = \begin{cases} 0, & g_j(X) \ge 0, \\ g_j(X), & g_j(X) < 0. \end{cases}$$

Схема алгоритма штрафных функций имеет представленный ниже вид.

1) Задаем начальную точку X^0, начальные значения компонент векторов α, μ, λ и полагаем счетчик числа итераций $t = 0$.

2) Исходя из точки X^t, одним из алгоритмов локальной безусловной оптимизации решаем задачу (1.22) – находим точку X^{t+1}.

3) Проверяем выполнение ограничений (1.21). Если эти условия выполнены, а также выполнено условие окончания поиска, то полагаем $\tilde{X}^* = X^{t+1}$ и завершаем итерации. Иначе по некоторому правилу увеличиваем значения тех из параметров μ_i, λ_j, $i \in [1:|E|]$, $j \in [1:|G|]$, которые соответствуют недопустимо нарушенным ограничениям, полагаем $t = t+1$ и переходим к шагу 2.

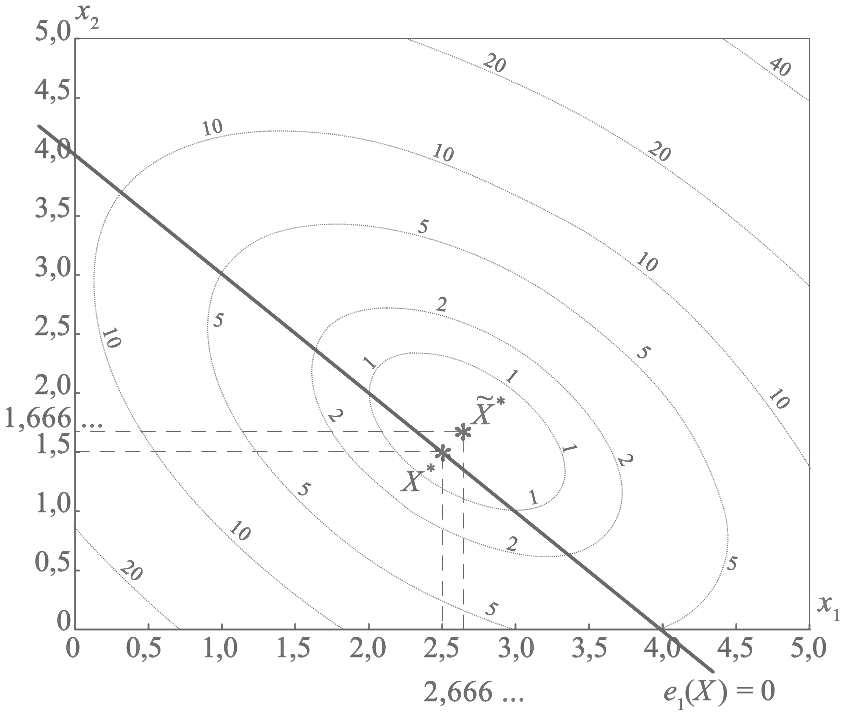


Рис. 1.15. Линии уровня функции $\varphi(X,\alpha)$: $\alpha = 1$

В качестве критерия окончания поиска при решении задачи (1.22) используют критерий соответствующего алгоритма безусловной оптимизации.

В зависимости от алгоритма локальной безусловной оптимизации, который используют для решения задач (1.22), комбинация этого алгоритма с алгоритмом штрафных функций может представлять собой детерминированный или стохастический алгоритм условной оптимизации.

Недостаток алгоритма последовательной безусловной оптимизации заключается в том, что значительно усложняется структура минимизируемой функции, что является платой за исключение ограничений. Поясним примером суть алгоритма, а также последнее утверждение.

Пример 1.1. Рассмотрим целевую функцию $f(X) = (x_1 - 3)^2 + (x_2 - 2)^2$ двух аргументов. Пусть имеется одно ограничение типа равенств с ограничивающей функцией $e_1(X) = x_1 + x_2 - 4$. Таким образом, $|E| = 1$, $|G| = 0$.

Легко видеть, что минимум функции $f(X)$ при указанном ограничении достигается в точке $X^* = (x_1^*; x_2^*) = (2{,}5; 1{,}5)$ – рис. 1.14. Положим, что штрафная функция имеет вид $p(X,\alpha) = \alpha e_1^2(X)$, где $\alpha \geq 0$ – вещественная константа. Линии уровня функции $\varphi(X,\alpha)$ при $\alpha = 0$; 1; 2 приведены на рис. 1.14, 1.15, 1.16 соответственно.

Из рис. 1.14 следует, что при минимизации функции $\varphi(X, 0)$, в лучшем случае, может быть найдено решение $\tilde{X}^* = (3{,}0; 2{,}0)$, так что

$$\left\| X^* - \tilde{X}^* \right\|_E = \sqrt{(3{,}0 - 2{,}5)^2 - (2{,}0 - 1{,}5)^2} = 0{,}71.$$

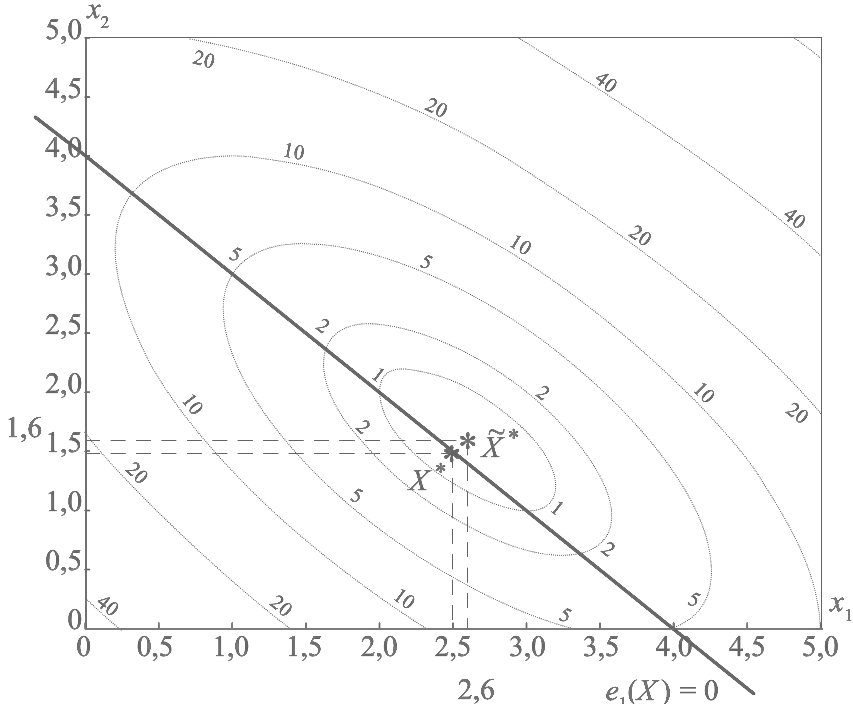


Рис. 1.16. Линии уровня функции $\varphi(X,\alpha)$: $\alpha = 2$

Аналогично, в случае $\varphi(X,1)$ получаем $\tilde{X}^* = (2{,}67;1{,}67)$, $\left\|X^* - \tilde{X}^*\right\|_E = 0{,}24$ (рис. 1.15), а в случае $\varphi(X,2)$ – $\tilde{X}^* = (2{,}60;1{,}60)$, $\left\|X^* - \tilde{X}^*\right\|_E = 0{,}14$ (рис. 1.16).

Пример 1.1 показывает, что асимптотически при $\alpha \to \infty$ решение $\tilde{X}^*$ стремится к точному решению X^*. С ростом величины α функция $\varphi(X,\alpha)$ становится все более овражной •

Алгоритмы на основе скользящего допуска

Положим, что множество допустимых значений D формируют только ограничения типа неравенств.

Метод скользящего допуска существенно использует неотрицательный скаляр $\delta = \delta(t)$ – *критерий скользящего допуска* и $w(X)$ – неотрицательно определенный функционал над множеством всех ограничивающих функций g_j, $j \in [1:|G|]$.

Критерий скользящего допуска δ определяет требуемую точность выполнения ограничений $g_j(X) \geq 0$, $j \in [1:|G|]$ и должен уменьшаться с ростом числа итераций t. Функционал $w(X)$ должен быть сконструирован таким образом, чтобы при $X \in D$ имело место равенство $w(X) = 0$, и значение $w(X)$ возрастало по мере удаления точки X вовне от границы области допустимых значений D.

Точка X называется *допустимой точкой*, если $w(X) = 0$, *почти допустимой точкой* – если $0 < w(X) \leq \delta$ и *недопустимой точкой* – если $w(X) > \delta$. Поскольку величина δ с ростом числа итераций уменьшается, отклонение от границы области D, при котором точка считается допустимой, сужается, так что в пределе рассматриваются только допустимые точки.

Алгоритм скользящего допуска может быть скомбинирован со многими алгоритмами многомерной локальной безусловной оптимизации. Будем называть алгоритм, с которым скомбинирован алгоритм скользящего допуска, *базовым алгоритмом*.

Итерация алгоритма скользящего допуска состоит из одного или двух следующих шагов.

1) С помощью базового алгоритма, исходя из текущей точки X, выполняем итерацию по решению задачи локальной безусловной оптимизации:

$$\min_{Y \in R^{|X|}} f(Y) = f(X').$$

Если $w(X') \le \delta$ (точка X' является допустимой или почти допустимой), то полагаем $X = X'$ и заканчиваем данную итерацию.

2) Если $w(X') > \delta$ (точка X' является недопустимой), то отыскиваем точку $X = X''$, лежащую ближе к границе области D. Для этого (обычно с помощью того же базового алгоритма), исходя из точки X', решаем задачу локальной безусловной оптимизации:

$$\min_{Y \in R^{|X|}} w(Y) = w(X'') \tag{1.23}$$

с условием окончания итераций

$$w(X'') \le \delta \tag{1.24}$$

и заканчиваем данную итерацию.

Достоинством алгоритма скользящего допуска является то, что степень нарушения ограничений по мере приближения к минимуму функции $f(X)$ постепенно уменьшается. Другими словами, на первых итерациях ограничения удовлетворяются лишь приближенно, а высокая точность удовлетворения ограничений обеспечивается только в окрестности этого минимума. Данное обстоятельство позволяет, как правило, сократить полный объем вычислений по сравнению с другими алгоритмами.

Использование в качестве базового алгоритма того или иного стохастического алгоритма безусловной оптимизации порождает набор соответствующих стохастических алгоритмов условной оптимизации.

Рассмотрим комбинацию алгоритма скользящего допуска с многоточечным алгоритмом гиперсферы как базовым алгоритмом.

В качестве критерия скользящего допуска в этом случае естественно использовать радиус гиперсферы $\rho = \rho^t$:

$$\begin{cases} \delta' = \min(\delta, \rho), \\ \delta^0 = \rho^0. \end{cases}$$

Легко видеть, что при всех t так определенная величина δ является положительной. С другой стороны, поскольку радиус гиперсферы при приближении к решению задачи уменьшается, справедливо предельное соотношение

$$\lim_{t \to \infty} \delta(t) = 0.$$

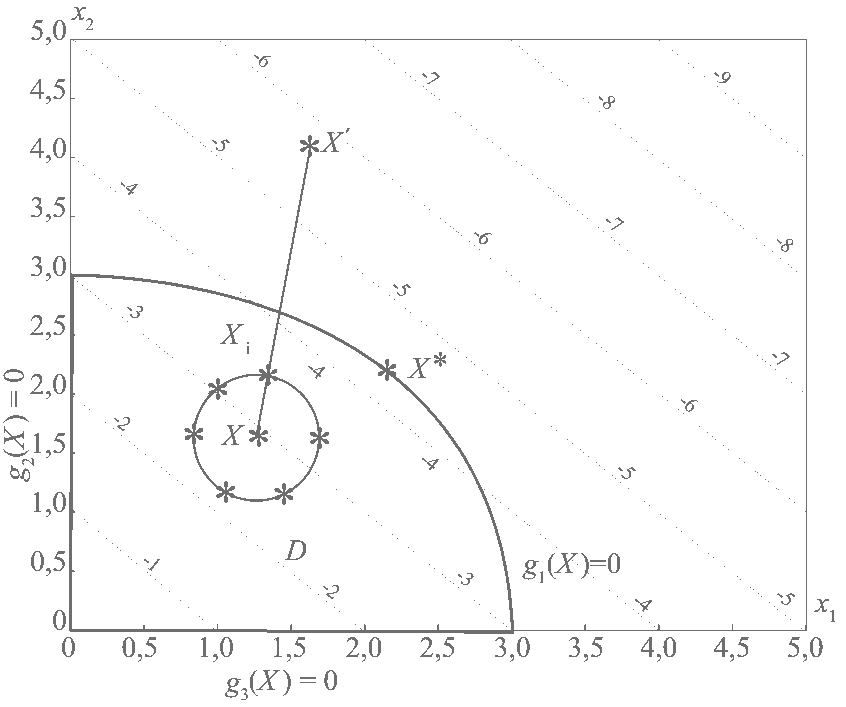


Рис. 1.17. К примеру 1.2: линии уровня функции $f(X) = -x_1 - x_2$

Для рассматриваемых ограничений типа неравенств в качестве $w(X)$ обычно используют функционал

$$w(X) = \sqrt{\sum_{i=1}^{|G|} (g_i^+(X))^2} \ .$$

Легко видеть, что так заданный функционал обладает свойством

$$\begin{cases} w(X) = 0, X \in D, \\ w(X) > 0, X \notin D, \end{cases}$$

а также тем свойством, что если точка X находится вне области D и недалеко от ее границы, то значение функционала мало.

Пример 1.2. Рассмотрим двумерную задачу условной оптимизации, когда целевая функция равна $f(X) = -x_1 - x_2$ и множество допустимых значений D определяют ограничения

$$\begin{cases} g_1(X) = -x_1^2 - x_2^2 + 9 \geq 0, \\ g_2(X) = x_1 \geq 0, \\ g_3(X) = x_2 \geq 0, \end{cases}$$

так что $|E| = 0$, $|G| = 3$ (рис. 1.17).

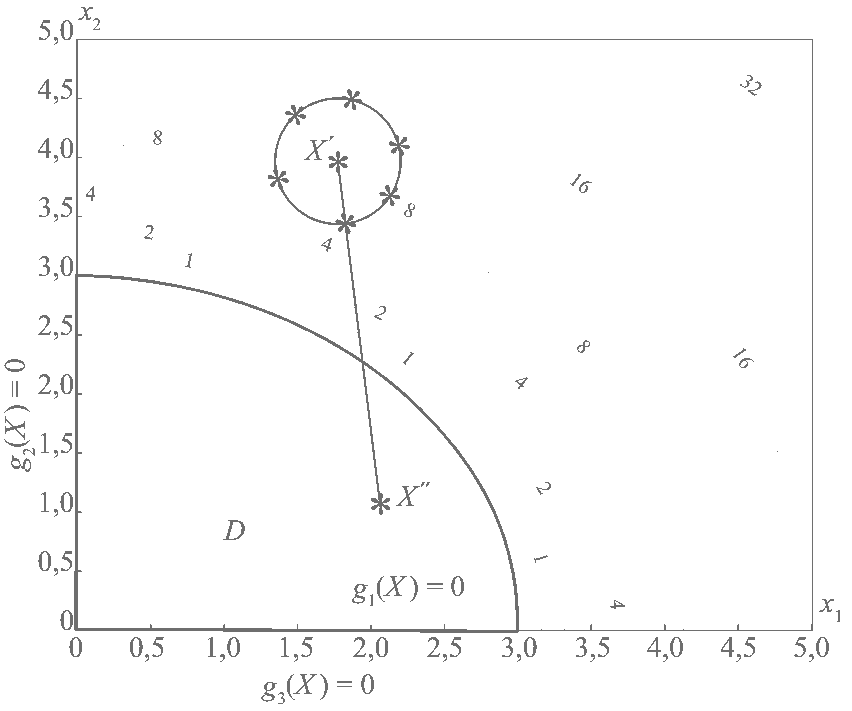


Рис. 1.18. К примеру 1.2: линии уровня функции $w(X)$

Используем модификацию алгоритма гиперсферы, при которой шаг в направлении $(X_i' - X)$ не определяется из условия минимума функции $f(X)$ в этом направлении, а фиксирован и равен $\lambda > 0$, так что координаты точки X' задает формула

$$X' = X + \lambda(X_i - X).$$

Пусть на данной итерации точка X' является недопустимой, т. е. $w(X') > \delta$ (рис. 1.17). Тогда на следующем шаге данной итерации решаем задачу минимизации функции $w(X)$, линии уровня которой приведены на рис. 1.18 •

Пример 1.2 иллюстрирует одну из сложностей применения алгоритма скользящего допуска – возможные *осцилляции решения* относительно границы области D. Суть явления осцилляции заключается в следующем. Если точка $X \in D$ находится близко к границе области D и величина текущего шага λ достаточно велика, то точка X' может оказаться вне области D и далеко от ее границы (рис. 1.17). При минимизации затем функции $w(X)$ точка X'' может опять оказаться глубоко внутри этой области (рис. 1.18). И так далее.

Для ослабления влияния осцилляций в простейшем случае можно вместо точки X' использовать точку A – середину отрезка $[X; X']$. Чаще с этой целью применяют квадратичную интерполяцию функции $w(X)$ на указанном отрезке по трем точкам $(X, w(X))$, $(A, w(A))$, $(X', w(X'))$ – рис. 1.19.

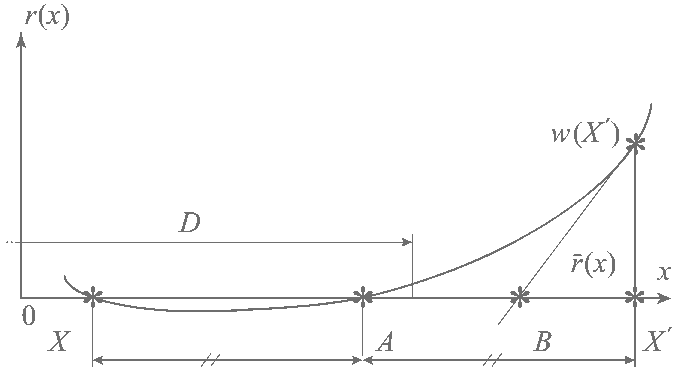


Рис. 1.19. Квадратичная интерполяция функции $w(X)$ для ослабления осцилляций

Обозначим интерполирующий полином $r(x) = ax^2 + bx + c$, где a, b, c – его коэффициенты. Вместо точки X' в этом случае используют приближенное значение B нуля функции $r(x)$, найденное, например, известным методом касательных.

1.3.2. Алгоритмы, не использующие редукцию задачи условной оптимизации к задаче безусловной оптимизации

Идея алгоритмов данного класса состоит в том, что если в процессе поиска агент выходит за границы области допустимых значений D, то он тем или иным образом возвращается в эту область.

Известно значительное число правил возвращения агента в область допустимых значений. Например, в методе *«смертельных» штрафов* (см. 2.6) агента, нарушившего ограничения, удаляют из популяции и вместо него в области поиска случайным образом создают нового агента.

Другое правило используется в методе *«лечения» локальным поиском* (см. 2.6). Пусть X_i – недопустимая точка, а X_j – ближайшая к ней в некоторой метрике допустимая точка. Тогда «лечение» агента s_i может быть произведено путем его перемещения по отрезку $[X_i; X_j]$ до тех пор, пока положение агента не станет допустимым.

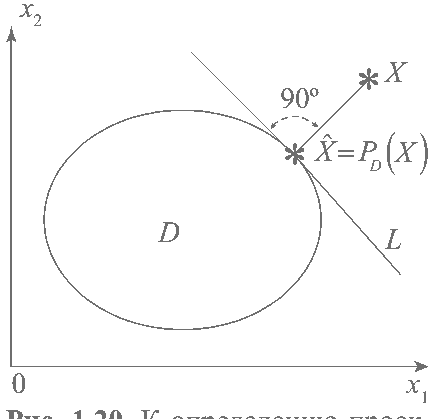


Рис. 1.20. К определению проекции точки на множество: $|X| = 2$

В некоторых случаях «лечение» недопустимого агента может быть выполнено путем проецирования точки X_i на границу области D. Примером алгоритма, использующего это правило «лечения», является *модифицированный алгоритм комплексов*. Рассмотрим данный алгоритм.

Положим, что множество допустимых значений D представляет собой гиперпараллелепипед Π. Нам понадобится далее понятие *проецирования точки на множество*.

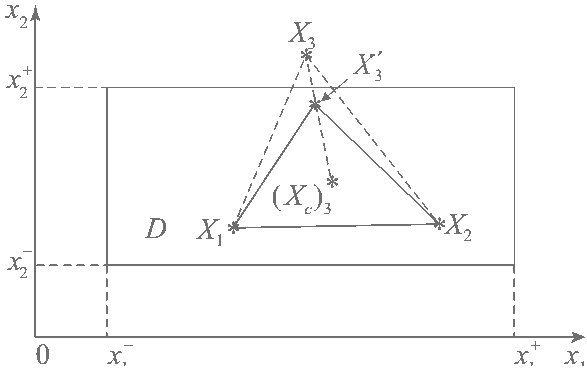


Рис. 1.21. К построению начального комплекса: $|X| = 2$

Определение 1.1. Точка $\hat{X} \in D$ называется проекцией точки $X \in R^{|X|}$ на замкнутое множество $D \subset R^{|X|}$, если

$$\rho(X, \hat{X}) = \min_{Y \in D} \rho(X, Y), \tag{1.25}$$

где $\rho(A, B)$ – расстояние между точками $A, B \in R^n$ в некоторой метрике.

Из определения 1.1 следует, что проекция точки $X \in R^{|X|}$ на замкнутое множество $D \subset R^{|X|}$ представляет собой ближайшую (в используемой метрике) к точке X точку $\hat{X}$ этого множества. Обозначим эту точку $P_D(X)$. Заметим, что если $X \in D$, то $P_D(X) = X$.

Определение 1.1 иллюстрирует рис. 1.20, где прямая L – касательная к границе области D в точке $P_D(X) = \hat{X}$. Можно показать, что если D – замкнутое выпуклое множество пространства $R^{|X|}$, то для любой точки $X \in R^{|X|}$ существует единственная ее проекция на это множество.

Задача (1.25) является задачей многомерной условной оптимизации, и ее решение может вызвать в общем случае значительные затруднения. Эта задача становится задачей квадратичного программирования, если множество D определяют лишь линейные ограничения типа неравенств и если функция $\rho(X, Y)$ является квадратичной функцией Y, например, если $\rho(X, Y) = \|X, Y\|_E$.

Легко видеть, что если множество D представляет собой гиперпараллелепипед $П$, то $P_D(X) = \hat{X} = (\hat{x}_1, \hat{x}_2, ..., \hat{x}_{|X|})$, где

$$\hat{x}_i = \begin{cases} x_i, & x_i^- \le x_i \le x_i^+, \\ x_i^-, & x_i < x_i^-, \\ x_i^+, & x_i > x_i^+, \end{cases} \quad i \in [1 : |X|]. \tag{1.26}$$

Модифицированный алгоритм комплексов использует следующие операции:

– генерация начального комплекса;
– отражение вершины комплекса с растяжением;
– сжатие комплекса;
– возврат вершины в область D путем сжатия комплекса;
– возврат вершины в область D путем ее проекции на множество D.

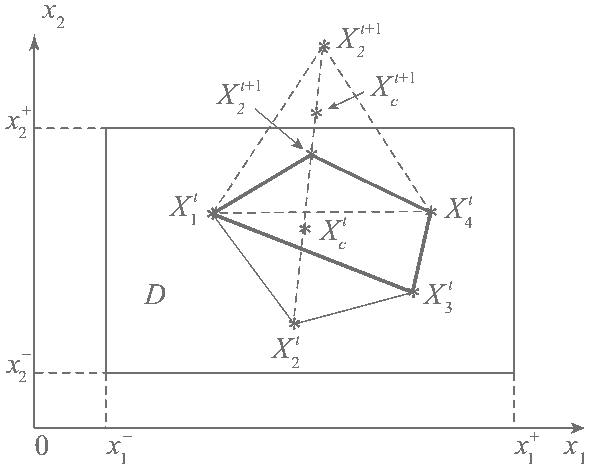


Рис. 1.22. Возврат вершины в область D путем сжатия комплекса: $|X| = 2$

Первые три операции рассмотрены в 1.2.3. Однако в данном случае операция генерации начального комплекса имеет некоторые особенности. Поэтому рассмотрим эту, а также две оставшиеся операции.

Генерация начального комплекса. Координаты вершин случайного комплекса $C = \{X_i, i \in [1:n]\}$ находим поочередно по формуле (1.16). Если при построении вершины X_i она оказывается недопустимой (выходит за границы области D), то по формуле, аналогичной формуле (1.18), выполняем сжатие уже построенного многогранника с i вершинами вдоль направления $(X_i - (X_c)_i)$ до тех пор, пока вершина X_i не окажется внутри области D. Здесь $(X_c)_i$ – центр тяжести построенного многогранника с i вершинами.

Процедуру иллюстрирует рис. 1.21. При построении точки X_3 она оказалась за границей области D. После операции сжатия вершины X_3 получена новая вершина X_3'.

Возврат вершины в область D путем сжатия комплекса. Пусть в результате отражения вершины комплекса с растяжением точка X_j^{t+1} оказалась недопустимой (вышла за границы области D) и имеет место неравенство $\varphi(X_j^{t+1}) > \varphi(X_j^t)$. Тогда по формуле (1.18) производим сжатие комплекса C^{t+1} вдоль направления $\left(X_j^{t+1} - X_c^{t+1}\right)$, где X_c^{t+1} – центр тяжести последнего комплекса. Если однократное сжатие не привело к тому, что точка X_j^{t+1} стала допустимой, производим повторное сжатие полученного комплекса и так далее до тех пор, пока точка X_j^{t+1} не станет допустимой (рис. 1.22).

Отметим, что необходимость проверки выполнения условия $\varphi(X_j^{t+1}) > \varphi(X_j^t)$ означает, что функция $\varphi(X)$ должна быть определена и вне области D.

Рис. 1.22 иллюстрирует следующую ситуацию. После отражения с растяжением комплекса C^t получен комплекс C^{t+1}, вершина X_2^{t+1} которого не принадлежит области D. В результате выполнения операции сжатия этого комплекса получена новая вершина X_2^{t+1}. Результирующий комплекс C^{t+1} на рис. 1.22 выделен.

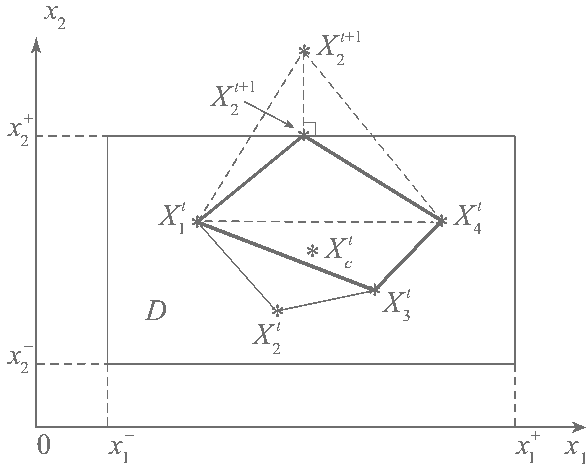


Рис. 1.23. Возврат вершины в область D путем проецирования вершины на это множество: $|X| = 2$

Возврат вершины в область D путем ее проекции на множество D. Пусть в результате отражения вершины комплекса с растяжением получена недопустимая точка X_j^{t+1} и имеет место неравенство $\varphi(X_j^{t+1}) < \varphi(X_j^t)$. Тогда по формуле (1.26) производим проецирование точки X_j^{t+1} на множество D. Отметим, что для выполнения данной операции функция $\varphi(X)$ также должна быть определена и вне области D.

Операцию возврата вершины в область D путем ее проекции на множество D иллюстрирует рис. 1.23. В результате отражения с растяжением комплекса C^t получен комплекс C^{t+1}, вершина которого X_2^{t+1} не принадлежит области D. После операции проецирования этой вершины на область D получена новая вершина X_2^{t+1}. Результирующий комплекс C^{t+1} на рис. 1.23 выделен.

Схема модифицированного алгоритма комплексов представлена ниже.

1) Задаем начальную точку $X^0 \in D$, величину λ, значения свободных параметров a, b, а также величину n. Полагаем счетчик числа итераций $t = 0$.

2) Исходя из точки X^0, строим случайный начальный комплекс $C^0 = \{X_i^0, i \in [1:n]\}$.

3) Вычисляем значения функции $\varphi(X)$ во всех тех вершинах комплекса C^t, в которых ранее эти значения не были вычислены. Находим максимальное из этих значений

$$\varphi(X_j^t) = \max_{i \in [1:n]} \varphi(X_i^t).$$

4) Отражаем с растяжением вершину X_j^t комплекса C^t – получаем вершину X_j^{t+1} и новый комплекс C^{t+1}.

5) Если $X_j^{t+1} \notin D$ и $\varphi(X_j^{t+1}) > \varphi(X_j^t)$, то выполняем возврат вершины X_j^{t+1} в область D путем сжатия комплекса и переходим к шагу 7.

6) Если $X_j^{t+1} \notin D$ и $\varphi(X_j^{t+1}) < \varphi(X_j^t)$, то выполняем возврат вершины X_j^{t+1} в область D путем ее проецирования на множество D и переходим к следующему шагу.

7) Если условие окончания поиска выполнено, то в качестве точки $\tilde{X}^*$ принимаем вершину комплекса C^{t+1}, в которой функция $\varphi(X)$ имеет наименьшее значение, и завершаем итерации, иначе полагаем $t = t+1$ и переходим к шагу 3.

С целью преодоления уплощения комплекса вблизи границы области допустимых значений D, через фиксированное число итераций выполняют процедуру восстановления комплекса (см. 1.2.3).

1.4. ГЛОБАЛЬНАЯ ОПТИМИЗАЦИЯ

В данном параграфе последовательно представляем три класса алгоритмов решения задачи многомерной глобальной условной оптимизации:

– алгоритмы на основе сведения исходной задачи к одной либо совокупности одномерных задач глобальной условной оптимизации;

– алгоритмы случайного поиска;

– поиск с запретами.

1.4.1. Алгоритмы одномерной редукции

К данному классу алгоритмов относим алгоритм одномерных вложений и алгоритм развертки Пеано. Полагаем, что множество допустимых значений D представляет собой гиперпараллелепипед Π.

Алгоритм одномерных вложений

Суть алгоритма одномерных вложений состоит в замене исходной задачи глобальной многомерной оптимизации совокупностью вложенных задач глобальной одномерной минимизации:

$$\min_{x_1 \in [x_1^-, x_1^+]} \min_{x_2 \in D(x_1)} \ldots \min_{x_{|X|} \in D(x_1, \ldots, x_{|X|-1})} \varphi\left(x_1, x_2, \ldots, x_{|X|}\right), \tag{1.27}$$

где множества $D(x_1)$, $D(x_1, x_2), \ldots, D(x_1, x_2, \ldots, x_{|X|-1})$ представляют собой соответствующие сечения множества D (см. ниже).

Рассмотрим для примера двумерную задачу $\left(|X| = 2\right)$. Задачи глобальной минимизации (1.27) в этом случае имеют вид

$$\min_{X \in D} \varphi\left(X\right) = \min_{x_1 \in \left[x_1^-, x_1^+\right]} \tilde{\varphi}\left(x_1\right), \tag{1.28}$$

$$\tilde{\varphi}\left(x_1\right) = \min_{x_2 \in D(x_1)} \varphi(x_1, x_2), \tag{1.29}$$

где $D(x_1)$ – сечение области D прямой, параллельной оси $0x_2$ и проходящей через точку $(x_1, 0)$ (рис. 1.24).

Задача (1.28) представляет собой задачу одномерной глобальной условной минимизации функции $\tilde{\varphi}(x_1)$. Для вычисления значений этой функции при некотором x_1 необходимо решить задачу (1.29) глобальной минимизации функции $\varphi(x_1, x_2)$ по параметру $x_2 \in D(x_1)$.

Положим, что для решения каждой из задач (1.27) используется один и тот же алгоритм. Обозначим через n_i число испытаний, необходимых для локализации этим алгоритмом с заданной точностью глобального минимума функции $\varphi(x_1,...,x_{i-1},x_i,x_{i+1},...,x_{|X|})$ по параметру $x_i \in [x_i^-, x_i^+]$ (когда параметры $x_1,...,x_{i-1},x_{i+1},...,x_{|X|}$ фиксированы). Тогда общее число испытаний для решения задачи (1.27) равно, очевидно,

$$n = \prod_{i=1}^{|X|} n_i .$$

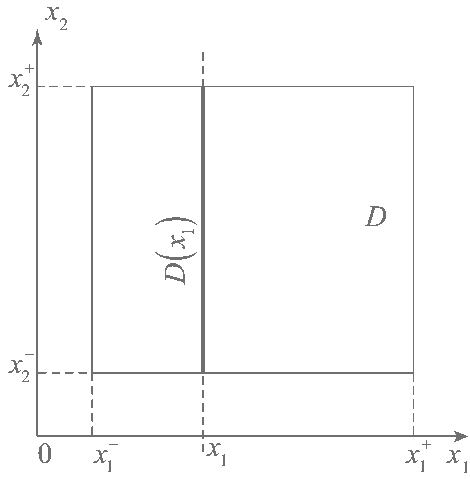


Рис. 1.24. К схеме алгоритма одномерных вложений: $|X| = 2$

Отсюда следует, что величина n быстро растет с ростом размерности пространства поиска $|X|$, так что уже при $|X| > 3$ алгоритм становится неэффективным. При $|X| \le 3$ надежность алгоритма достаточно высока, а затраты значительно меньше затрат на полный перебор на соответствующей сетке.

Алгоритм одномерных вложений (рис. 1.24) может быть скомбинирован как с детерминированными, так и со стохастическими алгоритмами решения задачи глобальной условной одномерной минимизации.

На примере двумерной задачи ($|X| = 2$) рассмотрим следующую схему комбинации этого алгоритма с алгоритмом Монте-Карло (см. 1.4.2).

1) Задаем величины n_1, n_2 – числа испытаний при решении задач (1.28), (1.29) соответственно. Полагаем $t = 1$.

2) Генерируем случайное число $\tilde{x}_1^t = \tilde{x}_1^* = U_1(x_1^-; x_1^+)$. Аналогично генерируем случайное число $\tilde{x}_2^t = \tilde{x}_2^*$. Вычисляем значение фитнес-функции $\tilde{\varphi}^t = \tilde{\varphi}^* = \varphi(\tilde{x}_1^*, \tilde{x}_2^*)$. Полагаем $t = 2$.

3) Аналогично шагу 2 генерируем случайное число x_1^t.

4) Методом Монте-Карло решаем задачу (1.29) при $x_1 = x_1^t$ – находим точку $\tilde{x}_2^t$ и вычисляем значение фитнес-функции $\tilde{\varphi}^t = \varphi(x_1^t, \tilde{x}_2^t)$.

5) Если $\tilde{\varphi}^t < \tilde{\varphi}^*$, то выполняем присваивания $\tilde{x}_1^* = x_1^t$, $\tilde{x}_2^* = \tilde{x}_2^t$, $\tilde{\varphi}^* = \tilde{\varphi}^t$.

6) Если $t < n_1$, то выполняем присваивание $t = t + 1$ и переходим к шагу 4. Иначе принимаем точку $(\tilde{x}_1^*, \tilde{x}_2^*)$ в качестве точки глобального минимума и заканчиваем вычисления.

Найденное решение может быть уточнено путем организации поиска локального минимума функции $\varphi(X)$ в окрестности найденной точки $(\tilde{x}_1^*, \tilde{x}_2^*)$ каким-либо из алгоритмов детерминированной либо стохастической локальной оптимизации.

Пример 1.3. Рассмотрим двумерную задачу поиска минимума функции Химмельблау (Приложение Б)

$$\varphi(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

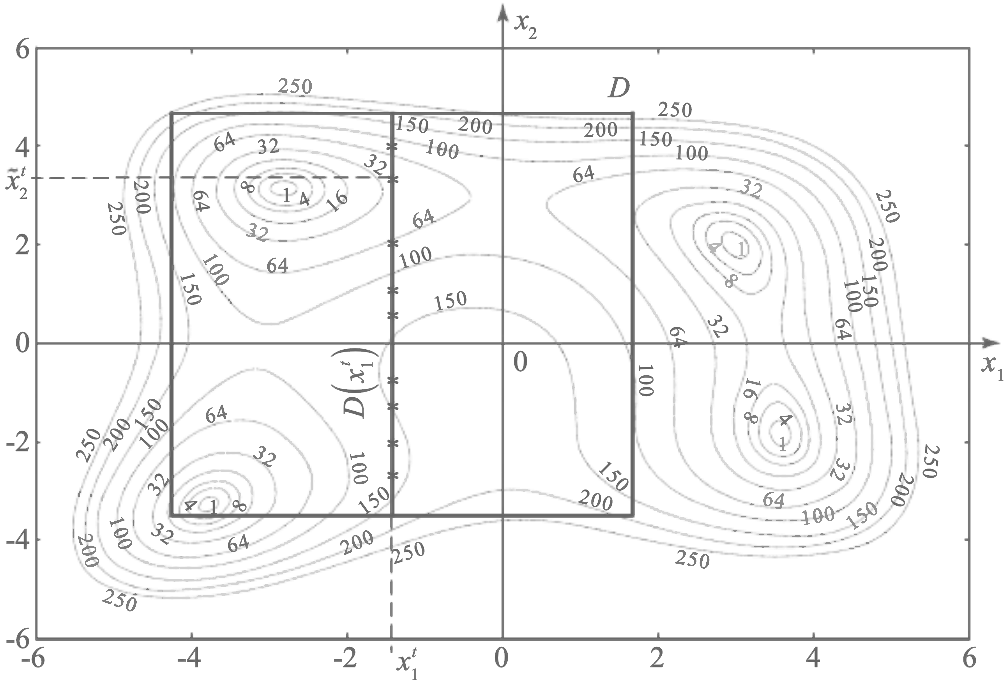


Рис. 1.25. К примеру 1.3. Одна итерация комбинации алгоритма одномерных вложений и одномерного алгоритма Монте-Карло

в области $\Pi = [-4,3; 1,7] \times [-3,3; 4,7]$. Одну итерацию решения этой задачи комбинацией алгоритма одномерных вложений и алгоритма Монте-Карло иллюстрирует рис. 1.25. Принято $n_2 = 9$ •

Алгоритм развертки Пеано

Рассмотрим задачу многомерной глобальной условной оптимизации в случае, когда параллелепипед Π представляет собой гиперкуб, длина ребер которого равна единице, так что $x_i^- = -0,5$, $x_i^+ = 0,5$; $i \in [1:|X|]$. Произвольный гиперпараллелепипед Π с помощью линейного преобразования может быть сведен к указанному гиперкубу.

Алгоритм развертки Пеано основан на непрерывном отображении гиперкуба D на отрезок вещественной оси. При построении этого отображения используют процедуры разбиения гиперкуба и единичного отрезка. Рассмотрим прежде эти процедуры.

Схема *процедуры разбиения единичного гиперкуба* имеет следующий вид.

Шаг 1 $(r = 1)$. Координатными плоскостями гиперкуб D разобьем на $2^{|X|}$ гиперкубов первого разбиения с длиной ребра, равной $\frac{1}{2} = \left(\frac{1}{2}\right)^{r=1}$ (рис. 1.26, *а*). Пронумеруем полученные гиперкубы с помощью переменной $z_1 \in \left[0 : \left(2^{|X|} - 1\right)\right]$. Гиперкуб первого разбиения с номером z_1 обозначим $D(z_1)$. Нумерацию произведем таким образом, чтобы гиперкубы с номерами $D(z_1)$ и $D(z_1 - 1)$, $z_1 \in \left[1 : \left(2^{|X|} - 1\right)\right]$ имели общую грань (в результате будут иметь общую грань

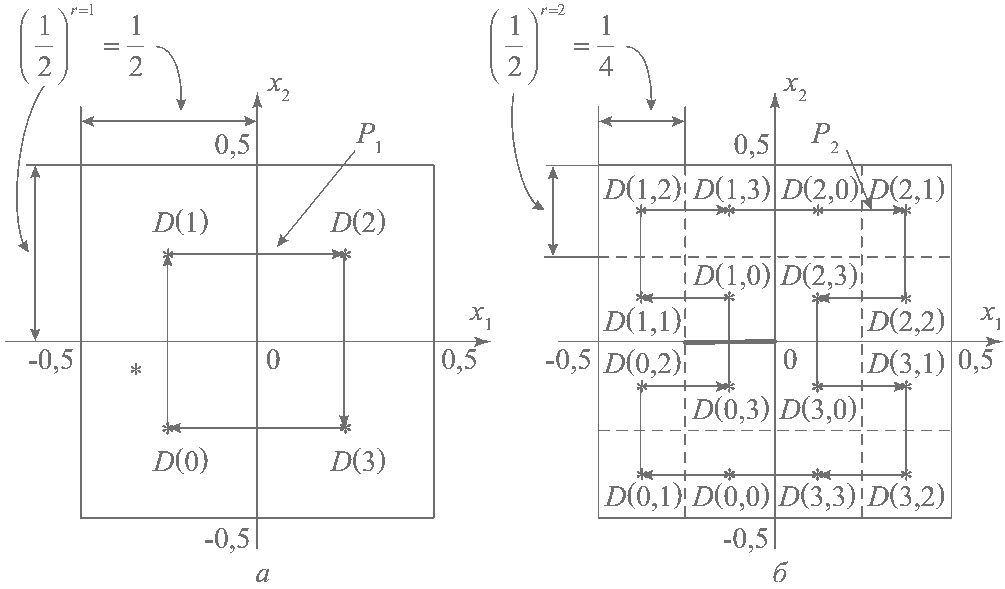


Рис. 1.26. Разбиение гиперкуба $\left(|X| = 2\right)$:
а – первое разбиение; *б* – второе разбиение

и гиперкубы $D(0)$, $D\left(2^{|X|} - 1\right)$. Соединим центры гиперкубов ломаной P_1 в порядке введенной нумерации.

Шаг 2 $(r = 2)$. По рассмотренной схеме каждый гиперкуб первого разбиения разобьем плоскостями, параллельными координатным плоскостям и проходящими через его центр, на $2^{|X|}$ гиперкубов второго разбиения с длиной ребра, равной $\left(\frac{1}{2}\right)^{r=2} = \frac{1}{4}$ (рис. 1.26, *б*). Пронумеруем полученные гиперкубы с помощью переменной $z_2 \in \left[0 : \left(2^{|X|} - 1\right)\right]$. Обозначим гиперкубы второго разбиения $D(z_1, z_2)$. Нумерацию выполним аналогично нумерации гиперкубов первого разбиения с тем отличием, что гиперкуб $D(z_1, 0)$ должен иметь общую грань с гиперкубом $D\left(\left(z_1 - 1\right), \left(2^{|X|} - 1\right)\right)$. Соединим центры гиперкубов ломаной P_2 в порядке введенной нумерации.

...

Шаг r. Аналогично шагу 2 разобьем гиперкубы $(r-1)$-го разбиения на гиперкубы *r*-го разбиения с длиной ребра, равной $\left(\frac{1}{2}\right)^r$, пронумеруем их с помощью переменной $z_r \in \left[0 : \left(2^{|X|} - 1\right)\right]$, соединим центры гиперкубов ломаной P_r в порядке введенной нумерации и обозначим полученные гиперкубы $D(z_1, z_2, ..., z_r)$.

...

Стрелками на рис. 1.26 показаны направления нумерации гиперкубов. В пределе при $r \to \infty$ ломаная P_r называется *кривой Пеано* (*разверткой Пеано*). Кривая Пеано обладает тем свойством, что проходит через все точки гиперкуба и имеет в каждой точке излом.

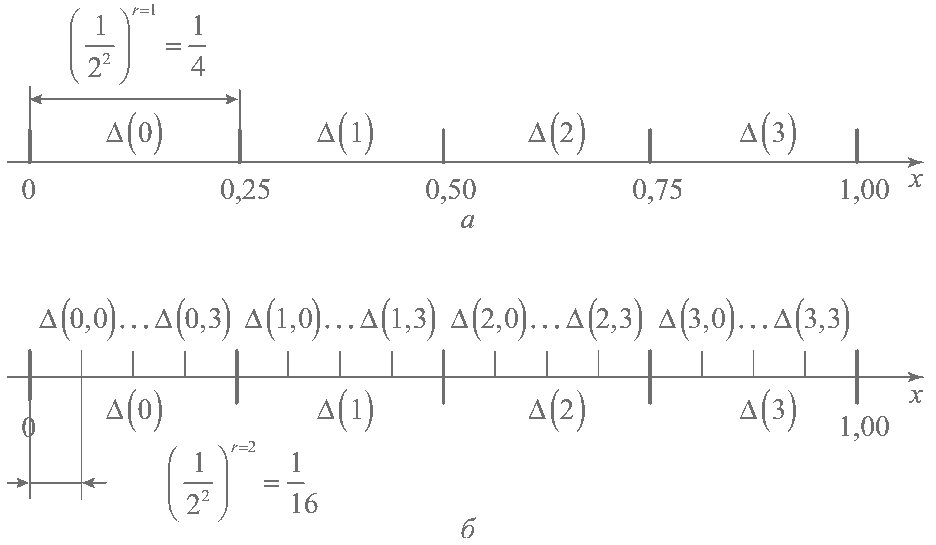


Рис. 1.27. Разбиение единичного отрезка $(|X| = 2)$: *а* – первое разбиение; *б* – второе разбиение

Схема *процедуры разбиения единичного отрезка* представлена ниже.

Шаг 1 $(r = 1)$. Разобьем отрезок $[0;1]$ на $2^{|X|}$ равных частей длиной $\left(\frac{1}{2^{|X|}}\right)^{r=1}$ (рис. 1.27, *а*), пронумеруем их слева направо с помощью переменной $z_1 \in \left[0:\left(2^{|X|-1}\right)\right]$ и обозначим $\Delta(z_1)$.

Шаг 2 $(r = 2)$. Каждый из отрезков $\Delta(z_1)$, $z_1 \in \left[0:\left(2^{|X|-1}\right)\right]$ разобьем на $2^{|X|}$ равных частей длиной $\left(\frac{1}{2^{|X|}}\right)^{r=2}$ (рис. 1.27, *б*), аналогично шагу 1 пронумеруем их с помощью переменной $z_2 \in \left[0:\left(2^{|X|-1}\right)\right]$ и обозначим $\Delta(z_1, z_2)$.

…

Шаг r. Аналогично шагу 2 каждый из отрезков $(r-1)$-го разбиения разобьем на $2^{|X|}$ равных частей длиной $\left(\frac{1}{2^{|X|}}\right)^{r}$, пронумеруем их с помощью переменной $z_r \in \left[0:\left(2^{|X|-1}\right)\right]$ и обозначим $\Delta(z_1, z_2, ..., z_r)$.

…

Перейдем к рассмотрению *процедуры отображения отрезка на гиперкуб*. Определим отображение точки x отрезка $[0;1]$ на гиперкуб D следующим образом: если $x \in \Delta(z_1, z_2, ..., z_r)$, то соответствующая точка $X_{D(z_1, z_2, ..., z_r)}$ является центром гиперкуба $D(z_1, z_2, ..., z_r)$. Обозначим введенное отображение $P(x)$. Таким образом, если $x \in \Delta(z_1, z_2, ..., z_r)$, то $P(x) = X_{D(z_1, z_2, ..., z_r)} \in D(z_1, z_2, ..., z_r)$.

Процедуру отображения отрезка на гиперкуб иллюстрирует рис. 1.28. Любая точка $x_1 \in \Delta(0,2)$ отображается в центр гиперкуба $D(0,2)$, т. е. в точку $X_{D(0,2)}$. Аналогично любой точке $x_2 \in \Delta(1,3)$ соответствует точка $X_{D(1,3)}$ и любой точке $x_3 \in \Delta(2,1)$ – точка $X_{D(2,1)}$.

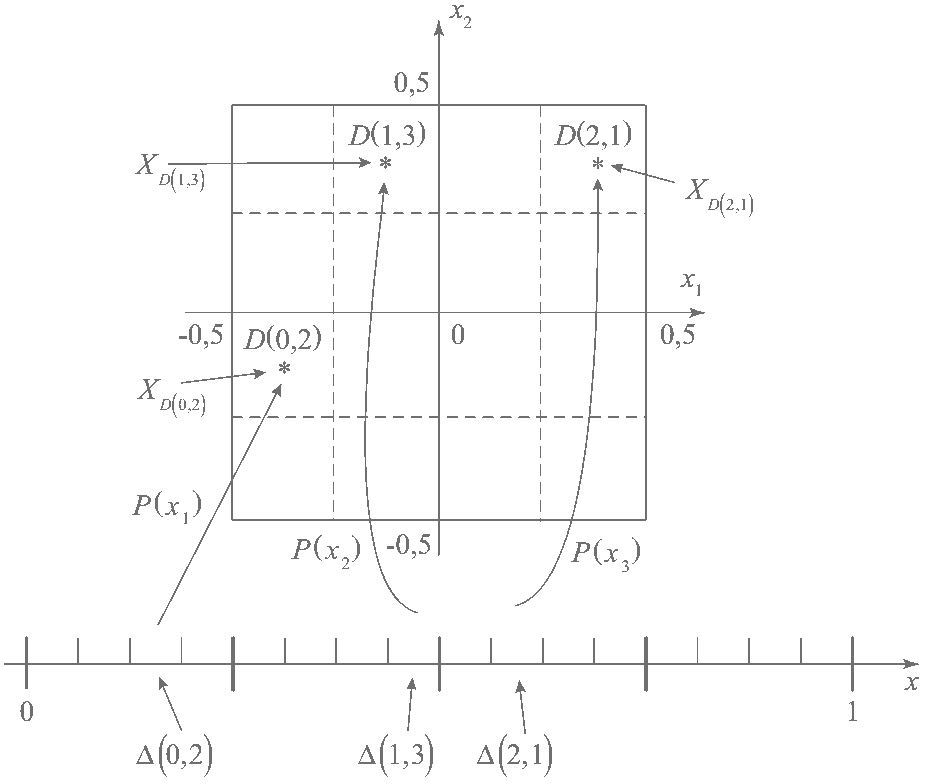


Рис. 1.28. К отображению единичного отрезка на гиперкуб

В пределе при $r \to \infty$ введенное отображение отображает отрезок $[0;1]$ на кривую Пеано и является непрерывным и взаимно-однозначным.

Пусть $0,\alpha_1\alpha_2...\alpha_i...$ – двоичное представление числа x, т. е.

$$x_{(2)} = 0,\alpha_1\alpha_2...\alpha_i...$$

Утверждение 1.1. Если $x \in \Delta(z_1, z_2, ..., z_r)$, то первые rn двоичных цифр числа $x_{(2)} = 0,\alpha_1\alpha_2...\alpha_i...$ определяют разбиение $\Delta(z_1, z_2, ..., z_r)$ отрезка $[0;1]$, где

$$z_1 = \langle \alpha_1\alpha_2...\alpha_n \rangle,$$
$$z_2 = \langle \alpha_{n+1}\alpha_{n+2}...\alpha_{2n} \rangle,$$
$$...$$
$$z_r = \langle \alpha_{(r-1)n+1}\alpha_{(r-1)n+2}...\alpha_{rn} \rangle$$

и $\langle \cdot \rangle$ – операция преобразования двоичного числа в десятичное.

Пример 1.4. Пусть размерность задачи $|X| = 2$. На отрезке $[0;1]$ рассмотрим точки $x_1 = 0,26$; $x_2 = 0,49$; $x_3 = 0,74$ (рис. 1.29).

Легко видеть, что $x_1 = 0,26_{(10)} = 0,0100001...._{(2)}$. Тогда из утверждения 1.1 вытекает, что

$$z_1 = \langle \alpha_1\alpha_2 \rangle = \langle 01 \rangle = 1_{(10)},\ z_2 = \langle \alpha_3\alpha_4 \rangle = \langle 00 \rangle = 0_{(10)}$$

и точка x_1 принадлежит отрезку $\Delta(z_1, z_2) = \Delta(1,0)$.

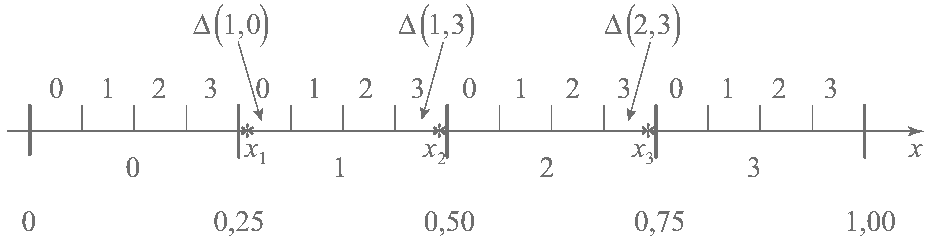


Рис. 1.29. Расположение точек x_1, x_2, x_3 на отрезке [0;1]

Аналогично для $x_2 = 0,49_{(10)} = 0,011111..._{(2)}$

$$z_1 = \langle \alpha_1 \alpha_2 \rangle = \langle 01 \rangle = 1_{(10)},\ z_2 = \langle \alpha_3 \alpha_4 \rangle = \langle 11 \rangle = 3_{(10)}$$

и $x_2 \in \Delta(z_1, z_2) = \Delta(1,3)$.

Наконец, для $x_3 = 0,74_{(10)} = 0,1011101..._{(2)}$

$$z_1 = \langle \alpha_1 \alpha_2 \rangle = \langle 10 \rangle = 2_{(10)},\ z_2 = \langle \alpha_3 \alpha_4 \rangle = \langle 11 \rangle = 3_{(10)}$$

и $x_3 \in \Delta(z_1, z_2) = \Delta(2,3)$ •

Определим на отрезке $[0;1]$ функцию одного переменного $\psi(x) = \varphi(P(x))$. Если функция $\varphi(X)$ является непрерывной, то функция $\psi(x)$ также непрерывна. Однако эта функция является негладкой и многоэкстремальной, даже если функция $\varphi(X)$ гладкая и унимодальная.

С помощью функции $\psi(x) = \varphi(P(x))$ многомерную задачу условной глобальной оптимизации сводим к аналогичной одномерной задаче

$$\min_{x \in [0;1]} \psi(x) = \psi(x^*). \tag{1.30}$$

Алгоритм развертки Пеано может быть скомбинирован как с детерминированными, так и со стохастическими алгоритмами решения задачи (1.30).

Во всех случаях предварительно должно быть определено необходимое число r разбиений гиперкуба D. Эта величина может быть найдена из следующих соображений. Пусть ε_X – заданная точность решения исходной задачи оптимизации. Гиперкуб r-го разбиения $D(z_1, z_2, ..., z_r)$ имеет длину ребра, равную $\left(\frac{1}{2}\right)^r$. Максимальное расстояние точек этого гиперкуба до его центра равно половине диагонали гиперкуба, которая, очевидно, равна

$$\sqrt{|X| \left(\frac{1}{2}\right)^{2r}} = \left(\frac{1}{2}\right)^r \sqrt{|X|}.$$

Таким образом, величина r может быть найдена из условия

$$\left(\frac{1}{2}\right)^{r+1} \sqrt{|X|} \le \varepsilon_X.$$

1.4.2. Алгоритмы случайного поиска

Алгоритмы случайного поиска могут быть разделены:
– на алгоритмы чисто случайного поиска (*pure random search*),
– двухфазные алгоритмы (*two-phase random search*) и
– алгоритмы на основе разбиения (*partitioning random search*).

Суть последних алгоритмов заключается в разделении области поиска на ряд подобластей, выделении из них наиболее перспективных с точки зрения вероятности расположения точки глобального экстремума и сосредоточении поиска преимущественно в этих подобластях. При разбиении области поиска могут быть использованы подходы на основе нечетких (*fuzzy*) функций. Алгоритмы на основе разбиения представляются весьма многообещающими, но в настоящее время развиты не достаточно и по этой причине нами не рассматриваются.

Чисто случайный поиск

Алгоритм Монте-Карло. Наиболее известным алгоритмом чисто случайного поиска является алгоритм Монте-Карло (*Monte Carlo*). Схема алгоритма имеет следующий вид.

1) Задаем общее число итераций $\hat{t}$ и полагаем $t=1$.

2) С помощью какого-либо генератора случайных чисел генерируем $|X|$ компонент вектора $X^t \in D$, вычисляем величину $\varphi(X^t)$ и полагаем $\tilde{X}^* = X^t$, $\tilde{\varphi}^* = \varphi\left(X^t\right)$, $t=2$.

3) Аналогично шагу 2 генерируем случайную точку $X^t \in D$ и вычисляем соответствующее значение фитнес-функции $\varphi^t = \varphi(X^t)$.

4) Выполняем присваивание

$$\tilde{X}^* = \begin{cases} X^t, & \varphi^t < \tilde{\varphi}^*, \\ \tilde{X}^*, & \varphi^t \geq \tilde{\varphi}^*. \end{cases}$$

5) Если $t < \hat{t}$, то полагаем $t = t+1$ и переходим к шагу 3. Иначе принимаем $\tilde{X}^*$ в качестве приближенного решения задачи и заканчиваем вычисления.

В простейшем случае точки $X^t \in D$ генерируют равномерно распределенными в области *D*. Для областей, имеющих сложную топологию, это достаточно трудная задача. Обычно в таком случае используют точки, равномерно распределенные в гиперпараллелепипеде, описанном вокруг области *D*. С целью сокращения вычислительных затрат и при наличии априорной информации о положении точки глобального минимума целесообразно использовать законы распределения, в которых вероятность генерации точки в окрестности предполагаемого глобального минимума выше, чем вне этой окрестности.

Известно значительное число алгоритмов, которые можно считать развитием алгоритма Монте-Карло. Ниже представлены несколько наиболее известных вариантов этого алгоритма – алгоритм имитации отжига, хаотический алгоритм локального поиска, алгоритм локального поиска на основе полетов Леви.

Алгоритм имитации отжига (*Simulated Annealing*, *SA*) предложен в 1953 г. Метрополисом (*N. C. Metropolis*). Алгоритм вдохновлен механизмом исправления дефектов в кристаллической решетке металла и некоторых других веществ.

Алгоритм *SA* можно считать одним из немногих универсальных алгоритмов решения задач глобальной оптимизации. Алгоритм используется при решении широкого класса задач дискретной, комбинаторной и непрерывной оптимизации. Например, известны случаи его высокой эффективности при решении задачи оптимального размещения микросхем на печатных платах, задачи обучения нейронных сетей с числами синаптических весов и смещений порядка нескольких сотен и т. д.

Важно, что удается аналитически исследовать свойства алгоритма *SA* и доказать его асимптотическую сходимость.

Указанные выше дефекты в кристаллической решетке обусловлены тем, что некоторые атомы занимают в решетке «неправильные» места. В силу нехватки кинетической энергии при нормальной температуре, указанные атомы не могут преодолеть потенциальный барьер и занять «правильные» положения в решетке. При этом в целом вся система атомов находится в состоянии локального энергетического минимума. Для вывода системы из этого минимума и перевода в состояние глобального энергетического минимума, соответствующего бездефектной кристаллической решетке, металл нагревают до высокой температуры, а затем медленно охлаждают. При этом «неправильные» атомы могут приобрести энергию, достаточную для преодоления потенциального барьера и занятия «правильных» положений в решетке. Вероятность преодоления потенциального барьера пропорциональна температуре металла, так что по мере его охлаждения вероятность такого перехода стремится к нулю, и кристаллическая решетка стабилизируется в окрестности нового (меньшего) локального или глобального минимума.

Таким образом, положительная роль повышения температуры отжигаемого металла заключается в том, что температурные флюктуации позволяют системе покидать локальные минимумы энергии и смещаться в сторону более глубоких энергетических минимумов.

Применение данной схемы к оптимизации основано на том, что локальное (субоптимальное) решение, найденное в процессе решения задачи оптимизации, также можно рассматривать как дефектное решение. Улучшить это решение (приблизиться к глобальному оптимуму) можно путем его случайных флюктуаций, амплитуда которых уменьшается с ростом номера итераций.

Принципиальным в алгоритме *SA* является то, что, в отличие от большинства других стохастических алгоритмов поисковой оптимизации, он допускает шаги, приводящие к увеличению значений фитнес-функции.

Алгоритм *SA* относится к классу так называемых пороговых стохастических алгоритмов безусловной оптимизации. Рассмотрим суть этих алгоритмов.

На данной итерации *порогового алгоритма* в окрестности $d(X)$ текущего приближения к решению X выбираем случайное решение X'. Если разность $\varphi(X') - \varphi(X) < \varepsilon$, то в качестве нового текущего приближения к решению при-

нимаем X'. В противном случае в окрестности $d(X)$ выбираем новое решение. Здесь ε – заданный положительный порог, величина которого по тому или иному закону убывает с ростом числа итераций t, так что имеет место предельное соотношение $\lim\limits_{t\to\infty}\varepsilon(t)=0$.

Пороговый алгоритм в процессе поиска допускает ухудшение значений фитнес-функции до заданного порога ε, и этот порог в процессе итераций последовательно снижается до нуля.

В алгоритме *SA* величина ε представляет собой случайную величину с математическим ожиданием, равным $\overline{\varepsilon}$, которому придается смысл «температуры» отжигаемого металла. Таким образом, в алгоритме *SA* переход от решения $(X,\ \varphi=\varphi(X))$ к решению $(X',\ \varphi'=\varphi(X'))$ допускается с вероятностью

$$\xi(X\to X')=\begin{cases}1, & \varphi'\le\varphi,\\ \exp\left(-\dfrac{\varphi'-\varphi}{\overline{\varepsilon}}\right), & \varphi'>\varphi.\end{cases}$$

Последняя формула означает, что если переход от точки X к точке X' приводит к уменьшению значения фитнес-функции $\varphi(X)$, то этот переход осуществляется безусловно. В противном случае переход выполняем с вероятностью, которая убывает с ростом разности $(\varphi'-\varphi)$ и уменьшением «температуры» $\overline{\varepsilon}$ в соответствии с известным *распределением Гиббса*.

Скорость сходимости алгоритма *SA* в значительной мере определяет вид функции $\overline{\varepsilon}(t)$. Поэтому известно большое число различных рекомендаций по выбору этой функции. Чаще всего вслед за основоположниками алгоритма используют функцию вида $\overline{\varepsilon}(t)=\alpha\,\overline{\varepsilon}(t-1)$, где α – положительный *коэффициент понижения «температуры»*, достаточно близкий к единице, например, $\alpha\in[0{,}8;0{,}99]$, так что последовательность $\overline{\varepsilon}(t)$, $\overline{\varepsilon}(t+1)$, $\overline{\varepsilon}(t+2),\ldots$ представляет собой убывающую геометрическую прогрессию.

Часто функцию $\overline{\varepsilon}(t)$ определяют формулой

$$\overline{\varepsilon}(t)=\frac{\overline{\varepsilon}_0}{\ln(t+1)},$$

где $\overline{\varepsilon}_0$ – начальная «температура».

Широко известен вариант так называемого *сверхбыстрого отжига* (*very fast annealing*), когда функцию $\overline{\varepsilon}(t)$ задает формула

$$\overline{\varepsilon}(t)=\overline{\varepsilon}_0\exp\left(-\beta\,t^{\frac{1}{|X|}}\right),$$

где β – положительная вещественная константа, имеющая смысл декремента затухания.

Хаотический алгоритм локального поиска (*Chaotic Local Search*, *CLS*) использует формирование хаотичных последовательностей на основе так называемых логистического и палаточного отображений. Рассмотрим прежде эти отображения.

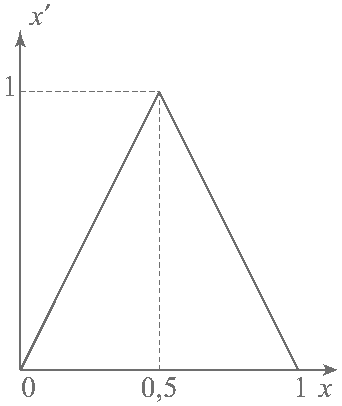


Рис. 1.30. К определению палаточного отображения

Логистическое отображение (*logistic mapping*) является дискретным аналогом известного непрерывного логистического уравнения Ферхюльста (*P. F. Verhulst*), используемого для моделирования динамики популяций, и имеет вид

$$x' = r\,x\,(1-x)\,,$$

где $x = x(t)$ – нормированная численность популяции в дискретный момент времени t, а $x' = x(t+1)$ – та же величина в момент времени $(t+1)$; $x(0) \in (0;1)$ – начальная численность.

Характер изменения численности популяции, описываемой логистическим уравнением, определяется значениями положительного параметра r. Важно, что при $r = 4$ и $x(0) \notin (0;\,0{,}25;\,0{,}5;\,0{,}75;\,1{,}0)$ это уравнение представляет собой модель хаотической динамической системы, так что процесс $x(t)$ является дискретным хаотичным процессом.

Функция плотности вероятности логистического отображения имеет вид

$$\xi(x) = \begin{cases} \dfrac{1}{\pi\sqrt{x(1-x)}}, & x \in (0;1{,}0), \\ 0, & x \notin (0;1{,}0). \end{cases}$$

Палаточное отображение (*tent map*) определяет рекуррентное уравнение

$$x' = \begin{cases} 2x, & x \in [0;\,0{,}5], \\ 2(1-x), & x \in (0{,}5;\,1{,}0). \end{cases} \tag{1.31}$$

Отображение получило свое наименование вследствие вида функции $x'(x)$ (рис. 1.30).

Функция плотности вероятности палаточного отображения является равномерной в интервале $x \in (0;1{,}0)$:

$$\xi(x) = \begin{cases} 1, & x \in (0;1{,}0), \\ 0, & x \notin (0;1{,}0). \end{cases}$$

В алгоритме *CLS* обычно используют модификацию классического палаточного отображения (1.31), имеющую вид

$$x' = \begin{cases} \dfrac{x}{0{,}4}, & x \le 0{,}4, \\ \dfrac{1-x}{0{,}6}, & x > 0{,}4 \end{cases}$$

и обеспечивающую, как утверждают ее авторы, устранение зацикливаний и нестабильностей, возможных при использовании отображения (1.31).

Рассмотрим задачу глобальной минимизации в параллелепипеде *П*. Схема одной итерации алгоритма *CLS*, построенного на основе логистического отображения, имеет следующий вид.

1) Последовательно для $j = 1, 2, ..., |X|$ выполняем следующие действия:

– нормализуем текущее значение *j*-й компоненты вектора X по формуле

$$x_j = \frac{x_j - x_j^-}{x_j^+ - x_j^-};$$

– вычисляем следующее значение величины x_j

$$x'_j = r\, x_j \left(1 - x_j\right);$$

– денормализуем полученную величину x'_j по формуле

$$x'_j = x_j^- + x'_j \left(x_j^+ - x_j^-\right).$$

2) Вычисляем в полученной точке X' значение фитнес-функции $\varphi(X')$.

3) Если $\varphi(X') < \varphi(X)$, то полагаем $\tilde{X}^* = X'$.

Итерации алгоритма продолжаем, например, до достижения максимально допустимого числа итераций.

Из приведенной схемы алгоритма *CLS* следует, что он совпадает, с точностью до способа получения последовательности точек, в которых производятся испытания фитнес-функции, с алгоритмом Монте-Карло.

Алгоритм на основе полетов Леви (*Lévy flights*) использует случайный процесс, предложенный Мандельбротом (*B. Mandelbrot*) – учеником французского математика Леви (*P. P. Lévy*).

Миграцию агента в соответствии с формулой полетов Леви определяет формула

$$X' = X + V \otimes L_{|X|}(\lambda), \tag{1.32}$$

где $V = (v_j,\ j \in [1 : |X|])$ – вектор размера шагов по соответствующим компонентам вектора X; $L_{|X|}(\lambda)$ – $(|X| \times 1)$-случайный вектор независимых вещественных случайных чисел, распределенных по закону Леви.

Полеты Леви обеспечивают высокую вероятность малых и достаточную вероятность больших шагов. Таким образом, траектория агента, движущегося в соответствии с формулой (1.32), представляет собой совокупность небольшого числа больших «исследовательских» шагов и значительного числа малых «локализующих» шагов.

Уравнение (1.32) определяет $(|X| \times 1)$-мерный стохастический процесс, представляющий собой *цепь Маркова* (*Markov chain*), когда следующее положение агента зависит только от его текущего положения (первое слагаемое в уравнении) и вероятности перехода (второе слагаемое). Известны исследования, показывающие, что случайные блуждания (1.32) являются более эффективными по сравнению с другими алгоритмами с точки зрения исследования пространства поиска. Установлено, что характер полета многих птиц и насекомых при исследовании ландшафта близок к полетам Леви. Исследования по-

ведения человека, например, собирателя грибов, также показывают характерные черты полетов Леви.

Схема алгоритма глобального поиска на основе полетов Леви с точностью до способа получения последовательности точек, в которых производятся испытания фитнес-функции, совпадает с алгоритмом Монте-Карло.

Двухфазные алгоритмы случайного поиска

Для локализации с помощью чисто случайных алгоритмов глобального минимума с высокой вероятностью и точностью требуется очень большое число испытаний. Двухфазные алгоритмы случайного поиска (*алгоритмы случайного рестарта*) имеют целью преодоление этого недостатка чисто случайных алгоритмов. Алгоритмы включают в себя *фазу глобального поиска* (*global phase*) и *фазу локального поиска* (*local phase*). Первая из указанных фаз состоит в генерации в области поиска некоторого числа случайных точек. Вторая фаза предполагает локальный поиск, исходя из каждой из этих точек.

Известно большое число алгоритмов данного класса. Рассмотрим наиболее известные из них.

Алгоритм мультистарта (*multistart* или *random restarts*). Схема алгоритма имеет следующий вид.

1) Задаем общее число исходных начальных точек равным $\hat{t}$ и полагаем $t = 1$.

2) С помощью какого-либо генератора случайных чисел генерируем координаты точки $X^t \in D$.

3) Исходя из точки X^t, одним из детерминированных либо стохастических алгоритмов многомерной локальной условной оптимизации находим локальный минимум $\tilde{X}^t = \tilde{X}^*$, $\varphi(\tilde{X}^t) = \tilde{\varphi}^t = \tilde{\varphi}^*$ функции $\varphi(X)$ в окрестности точки X^t. Полагаем $t = 2$.

4) По рассмотренной схеме генерируем координаты следующей точки $X^t \in D$.

5) Выполняем действия шага 3 – находим величины $\tilde{X}^t$, $\tilde{\varphi}^t$ и принимаем

$$\tilde{X}^* = \begin{cases} \tilde{X}^t, & \tilde{\varphi}^t < \tilde{\varphi}^*, \\ \tilde{X}^*, & \tilde{\varphi}^t \geq \tilde{\varphi}^*. \end{cases}$$

6) Если $t < \hat{t}$, то полагаем $t = t + 1$ и переходим к шагу 4, иначе заканчиваем вычисления.

Алгоритм мультистарта имеет тот очевидный недостаток, что при его использовании некоторые локальные решения могут быть найдены многократно, а другие такие же решения не будут достигнуты вообще. Рассматриваемые ниже другие двухфазные алгоритмы можно считать модификациями алгоритма мультистарта, направленными на преодоление этого его недостатка.

Жадный адаптивный алгоритм случайного поиска (*Greedy Randomized Adaptive Search Procedure*, *GRASP*) предложили Фео (*T. A. Feo*) и Ресенде (*M. G. C. Resende*) в 1989, 1995 гг. Непрерывную версию алгоритма (*C-GRASP*) предложил Хирш (*M. J. Hirsch*) с соавторами в 2007 г.

Рассмотрим задачу глобальной минимизации в параллелепипеде Π. Общая схема алгоритма *C-GRASP* в этом случае имеет следующий вид.

1) Выбираем во множестве Π случайную точку X и задаем начальное значение шага локального поиска λ.

2) Исходя из точки X, с помощью *процедуры случайной жадной эвристики* (см. ниже) находим начальное решение X'.

3) С помощью той или иной *процедуры локального поиска*, исходя из точки X', решаем задачу локальной оптимизации – получаем точку $\tilde{X}$.

4) Если $\varphi(\tilde{X}) < \varphi(X)$, то полагаем $X = \tilde{X}$.

5) Проверяем выполнение условия адаптации шага λ. Если это условие выполнено, уменьшаем по некоторому правилу эту величину, например, полагаем $\lambda = \nu\,\lambda$, $\nu \in (0;1)$.

6) Если условие окончания поиска не выполнено, возвращаемся к шагу 2.

Из приведенной схемы следует, что основными процедурами алгоритма *C-GRASP* являются процедура, реализующая случайную жадную эвристику выбора, и процедура локальной оптимизации. Рассмотрим эти процедуры.

Случайная жадная эвристика. Обозначим список номеров компонент вектора варьируемых параметров $N = \{1, 2, ..., |X|\}$. Схему эвристики можно представить в следующем виде.

1) Для каждого $j \in N$ выполняем одномерный поиск вдоль координатного направления x_j, полагая остальные компоненты текущего вектора X неизменными.

2) Вычисляем значение фитнес-функции $\varphi(Y_j)$ в точках $Y_j = (y_{j,1}, y_{j,2}, ..., y_{j,|X|})$, $j \in N$, найденных в результате указанного одномерного поиска, а также величины

$$\varphi_{\min} = \min_{j \in N} \varphi(Y_j), \quad \varphi_{\max} = \max_{j \in N} \varphi(Y_j).$$

3) Формируем *ограниченный список кандидатов* (*Restricted Candidate List*, *RCL*), содержащий те из номеров j текущего набора N, для которых выполняется неравенство

$$\varphi(Y_j) \le (1-\alpha)\,\varphi_{\min} + \alpha\,\varphi_{\max},$$

где $\alpha \in [0;1]$ – свободный параметр алгоритма.

4) Из списка *RCL* случайным образом выбираем индекс k, исключаем этот индекс из набора N и полагаем $x_k = y_{j,k}$.

5) Если набор N не пуст, то переходим к шагу 1. В противном случае возвращаем текущий вектор X.

Легко видеть, что на самом деле рассмотренная эвристика является жадной только при $\alpha = 0$, а при всех иных значениях этой величины представляет собой *полужадную* (*semi-greedy*) *эвристику*.

Локальный поиск. Принципиально процедура локального поиска может быть построена на основе любого детерминированного или стохастического локального алгоритма поисковой оптимизации. Оригинальная версия алгоритма *C-GRASP* предполагает использование одного из вариантов алгоритма, основанного на стохастической аппроксимации градиента фитнес-функции (см. 1.2.1).

Введем в рассмотрение множество номеров $N = \{n_j,\ j \in [1:|N|]\}$ возможных направлений перемещения вдоль координатных осей, где $|N| = 3^{|X|} - 1$ – общее число направлений вида $V = \left(v_1, v_2, ..., v_{|X|}\right)$, $v_k \in \{-1; 0; 1\}$, $k \in [1:|X|]$, исключая направление $(0, 0, ..., 0)$.

Перед началом работы процедуры выполняем присваивания $\tilde{X} = X'$, $\tilde{\varphi} = \varphi(X')$ и полагаем $\tilde{N} = Null$. До тех пор, пока имеет место уменьшение значений фитнес-функции, выполняем следующие действия.

1) Находим натуральное число $n = U_1\left(1:\left(3^{|X|} - 1\right)\right) \notin \tilde{N}$ и включаем его во множество $\tilde{N}$.

2) Определяем соответствующее направление $V = Ternary(n)$ указанного выше вида, где $Ternary(n)$ – функция, выполняющая преобразование числа n в вектор V.

3) Полагаем $X = \tilde{X} + \lambda V$ и вычисляем $\varphi(X) = \varphi$, где $\lambda \in [0; 1]$ – свободный параметр алгоритма.

4) Если $X \in [X^-, X^+]$ и $\varphi < \tilde{\varphi}$, то выполняем присваивания

$$\tilde{X} = X, \tilde{\varphi} = \varphi,\ \tilde{N} = Null.$$

5) Если $|\tilde{N}| < \hat{n}$ и в результате указанных шагов улучшить значение фитнес-функции не удалось, то возвращаемся к шагу 1. Здесь $\hat{n} \le |N|$ – максимально допустимое число шагов перемещения (свободный параметр алгоритма).

Известно значительное число модификаций алгоритма *C-GRASP*. В соответствии с одной из модификаций на каждой итерации алгоритма используют случайные равномерно распределенные на интервале $[0; 1]$ значения параметра λ. В другой модификации эти значения адаптивно выбирают из предварительно построенного множества допустимых величин $(\lambda_1, \lambda_2, ..., \lambda_m)$ таким образом, что обеспечивается более высокая вероятность выбора той из этих величин, при использовании которой были получены лучшие решения.

Итерационный локальный поиск (*Iterated Local Search*, *ILS*). Идея алгоритма *ILS* проста и в той или иной форме использовалась достаточно давно. Однако название *Iterated Local Search* впервые появилось в одной из публикаций Лоуренсо (*H. R. Lourenco*) с соавторами только в 2003 г.

Вместо выбора нового начального решения случайным образом, алгоритм *ILS* создает его путем некоторого возмущения (*perturbation*) предыдущего найденного решения. Основная проблема при этом состоит в том, что данное возмущение, с одной стороны, не должно быть слабым, чтобы избежать зоны притяжения локального решения, а с другой стороны не должно быть слишком сильным, поскольку в таком случае стратегия алгоритма будет мало отличаться от стратегии алгоритма мультистарта.

Схема алгоритма *ILS* имеет следующий вид.

1) В области допустимых значений D генерируем начальное случайное решение X^0. Полагаем счетчик числа итераций $t = 1$.

2) Исходя из точки X^0, с помощью того или иного алгоритма локального поиска отыскиваем решение $\tilde{X}^*$.

3) Путем возмущения решения $\tilde{X}^*$ получаем точку $X^0(t)$.

4) Исходя из точки $X^0(t)$, с помощью используемого алгоритма локального поиска отыскиваем решение $\tilde{X}^*(t+1)$.

5) Если $\varphi(\tilde{X}^*(t+1)) < \varphi(\tilde{X}^*)$, то полагаем $\tilde{X}^* = \tilde{X}^*(t+1)$.

6) Если условие окончания поиска не выполнено, то полагаем $t = t+1$ и возвращаемся к шагу 3. В противном случае в качестве решения задачи принимаем текущие значения величин $\tilde{X}^*$, $\varphi(\tilde{X}^*)$ и завершаем вычисления.

Процедура возмущения алгоритмом не фиксирована, требуется лишь, чтобы она удовлетворяла следующим требованиям:

– возмущение должно быть достаточно сильным, чтобы новое начальное решение $X^0(t)$ располагалось в области притяжения другого локального решения;

– возмущение не должно быть полностью случайным;

– процедура может использовать историю найденных решений, а не только текущее лучшее решение.

Процедура возмущений может быть построена на основе любого из популяционных алгоритмов. Часто с этой целью используют генетический алгоритм. Локальный поиск в алгоритме *ILS* может строиться на базе любых классических и неклассических алгоритмов локальной оптимизации. Таким образом, алгоритм *ILS* в действительности представляет собой метаэвристику, позволяющую построить алгоритм глобальной оптимизации, учитывающий особенности решаемой задачи.

Известно большое число вариантов и модификаций алгоритма *ILS*. Рассмотрим в качестве примера одну из таких модификаций – *алгоритм поиска с чередующимися окрестностями* (*Variable Neighborhood Search algorithm*, *VNS algorithm*). Впервые данный алгоритм был представлен в работе Младеновича (*N. Mladenovic*) и Хансена (*P. Hansen*) в 2001 г.

Схема алгоритма *VNS* представлена ниже.

1) В области допустимых значений D формируем множество окрестностей $\boldsymbol{d} = \{d^t, t \in [1:\hat{t}]\}$. Генерируем начальное случайное решение $\tilde{X}^* \in D$.

2) Принимаем $t = 1$.

3) В окрестности d^t генерируем начальное случайное решение $X^0(t)$.

4) Исходя из точки $X^0(t)$, с помощью некоторого алгоритма локального поиска отыскиваем решение $\tilde{X}^*(t)$.

5) Если $\varphi(\tilde{X}^*(t)) < \varphi(\tilde{X}^*)$, то полагаем $\tilde{X}^* = \tilde{X}^*(t)$, $t = 1$ и возвращаемся к шагу 3. Если $\varphi(\tilde{X}^*(t)) \geq \varphi(\tilde{X}^*)$ и $t < \hat{t}$, то полагаем $t = t+1$ и переходим к шагу 3.

6) Если $\varphi(\tilde{X}^*(t)) \geq \varphi(\tilde{X}^*)$ и $t = \hat{t}$, то проверяем выполнение условия окончания итераций. Если это условие не выполнено, то переходим к шагу 2, в противном случае в качестве решения задачи принимаем текущие значения величин $\tilde{X}^*$, $\varphi(\tilde{X}^*)$ и завершаем вычисления.

Аналогично алгоритму *ILS*, алгоритм *VNS* представляет собой метаэвристику, поскольку не фиксирует правила формирования окрестностей $\boldsymbol{d}$, правила генерации начальных случайных решений $X^0(t)$, принадлежащих этим окрестностям, а также правила выбора алгоритмов локального поиска.

При выборе стратегии формирования окрестностей d^t приходится учитывать два противоречивых соображения. С одной стороны, желательно иметь множества d^t небольшого размера, чтобы сократить трудоемкость локального поиска в них. С другой стороны, более широкие множества d^t могут обеспечить отыскание лучшего локального оптимума. Одним из путей разрешения этого противоречия является использование множеств, размер которых может варьироваться в ходе локального поиска.

Обычно окрестность $d(X)$ решения $X \in D$ определяют в виде шара

$$d(X) = \{Y \in D \mid \rho(X,Y) \le \rho^{\max}\},$$

где $\rho(\cdot,\cdot)$ – некоторая метрика пространства $R^{|X|}$, $\rho^{\max}$ – радиус окрестности. Например, может быть использована метрика

$$\rho(X,Y) = \left(\sum_{j=1}^{|X|} \left|x_j - y_j\right|^p\right)^{\frac{1}{p}}, \ 1 \le p < \infty$$

или метрика

$$\rho(X,Y) = \max_{j \in [1:|X|]} \left|x_j - y_j\right|.$$

Одним из вариантов является формирование вложенных окрестностей.

В приведенной выше схеме алгоритма *VNS* использован фиксированный порядок смены окрестностей и поиск локального минимума относительно каждой из них. Наряду с этим возможен случайный выбор очередной окрестности (*Random VNS*, *RVNS*), который порождает *алгоритм поиска со случайными чередующимися окрестностями* или смешанный выбор.

Известно значительное число других двухфазных алгоритмов случайного поиска, например, кластеризующий алгоритм (*density clustering*), алгоритм управляемого случайного поиска (*controlled random search*), топографический алгоритм (*topographical optimization*).

1.4.3. Поиск с запретами

Алгоритм поиска с запретами (*Tabu Search algorithm*, *TS algorithm*) относится к классу метаэвристических алгоритмов интеллектуального поиска. Основоположником алгоритма *TS* является Гловер (*F. Glover*). Первоначально алгоритм *TS* нашел свое применение в задачах теории графов и целочисленного программирования. Для решения непрерывных задач оптимизации алгоритм начали использовать в конце 80-х годов прошлого столетия. В настоящее время алгоритм широко применяют также в задачах планирования и кластерного анализа.

Основным механизмом, который позволяет алгоритму *TS* преодолевать локальные экстремумы, является *список запретов* $T(t) = T^t$ (*tabu list*). Список строится на основе некоторой части предыстории поиска, то есть на основе решений

$$X^t, X^{t-1}, ..., X^{t-k}, \quad (t-k) \geq 0, \tag{1.33}$$

и запрещает часть окрестности d^t текущего решения X^t, так что новая точка X^{t+1} является решением вспомогательной задачи локальной условной оптимизации

$$f(X^{t+1}) = \min_X f(X), \; X \in d^t \setminus T^t. \tag{1.34}$$

Очевидно, что алгоритм *TS* трансформируется в обычный алгоритм локального поиска в случае, когда список T^t пуст, то есть когда $k = 0$.

Элементы списка T^t называют *табу-перемещениями* (*tabu moves*). Вообще говоря, список запретов может иметь переменную длину, когда $k = k(t) = k^t$. Величина k^t имеет смысл числа итераций, в течение которых данное решение находится в списке T^t, и называется *табу-продолжительностью* (*tabu tenure*). Известна рекомендация, по которой $k^t = O(|X|)$.

Общая схема алгоритма *TS* имеет следующий вид.

1) Тем или иным образом выбираем начальную точку X^0 и полагаем $\tilde{X}^* = X^0$, $\tilde{\varphi}^* = \varphi(X^0)$, $T^0 = Null$, $t = 0$.

2) Если $d^t \setminus T^t = Null$, то переходим к шагу 4. В противном случае, по формуле (1.34) находим решение X^{t+1} и полагаем $t = t + 1$.

3) Если $\varphi(X^t) < \tilde{\varphi}^*$, то выполняем присваивания $\tilde{X}^* = X^t$, $\tilde{\varphi}^* = \varphi(X^t)$.

4) Если условия окончания поиска выполнены, завершаем вычисления. В противном случае обновляем список запретов T^t и возвращаемся к шагу 2.

Особенностью условий окончания поиска в алгоритме *TS* является то, что эти условия включают в себя пустоту множества $d^t \setminus T^t$.

Слишком строгие запреты могут привести к преждевременной стагнации вычислительного процесса. Поэтому в алгоритме *TS* эпизодически разрешается нарушать запреты. С этой целью используют механизм так называемых *многообещающих решений* (*aspirant decisions*) или эквивалентный механизм *функции аспирации* (*aspirant-function*) $a(X)$, которая отменяет табу-статус перемещения *X*. Говорят, что решение $X \in T^t$ достигло уровня аспирации и может быть объявлено допустимым, если $\varphi(X) \leq a(X)$.

Вопросы для самопроверки

1. Приведите постановку детерминированной задачи оптимизации. Определите основные классы целевых функций.

2. Дайте классификацию задач оптимизации по виду целевой и ограничивающих функций, наличию и отсутствию ограничений, а также по их характеру, размерности вектора варьируемых параметров, числу точек минимума, характеру искомого решения.

3. Дайте классификацию поисковых алгоритмов оптимизации по характеру искомого решения, характеру ограничений, виду итерационной функции, числу предыдущих учитываемых шагов, порядку используемых производных.

4. Приведите итерационные формулы основных одношаговых алгоритмов случайного поиска, а также правила выбора величины шага в этих алгоритмах.

5. Представьте общую итерационную формулу многошаговых алгоритмов случайного поиска.

6. Изобразите траекторию поиска минимума некоторой двумерной овражной функции с помощью алгоритма комплексов (конфигураций).

7. Изобразите траекторию поиска минимума двумерной квадратичной функции с помощью алгоритма гиперсферы (алгоритма случайного поиска с постоянным радиусом поиска и случайными направлениями).

8. Изложите общую схему алгоритмов последовательной безусловной оптимизации. Приведите схему алгоритма штрафных функций.

9. Назовите отличительные особенности метода скользящего допуска. Объясните суть явления осцилляции решений в этом методе.

10. Перечислите основные алгоритмы условной оптимизации, не использующие редукцию этой задачи к задаче безусловной оптимизации.

11. Представьте схему модифицированного алгоритма комплексов для решения задачи условной оптимизации.

12. Изложите схемы алгоритмов одномерных вложений и развертки Пеано.

13. Перечислите основные алгоритмы чисто случайного поиска. Изложите суть алгоритмов Монте-Карло, имитации отжига, хаотического алгоритма локального поиска, а также алгоритма на основе полетов Леви.

14. Дайте общее определение двухфазных алгоритмов случайного поиска. Укажите основные особенности алгоритмов мультистарта, жадного адаптивного алгоритма случайного поиска, алгоритма итерационного локального поиска.

15. Изложите суть алгоритма поиска с запретами.

Глава 2
ЭВОЛЮЦИОННЫЕ АЛГОРИТМЫ

Эволюционные алгоритмы включают в себя генетические алгоритмы (*genetic algorithm*), *эволюционную стратегию* (*evolution strategy*), *эволюционное программирование* (*evolutionary programming*), алгоритмы *дифференциальной эволюции* (*differential evolution*), а также *генетическое программирование* (*genetic programming*). Последний класс алгоритмов не используют для решения задач непрерывной оптимизации, поэтому он далее не рассматривается.

Суть парадигмы эволюционных алгоритмов состоит в использовании базовых принципов теории биологической эволюции – отбора, мутации и воспроизводства особей. Эволюционные алгоритмы являются частью более широкой технологии *мягких вычислений* (*soft computing*), включающих в себя еще *нечёткую логику* Заде (*L. A. Zadeh*), *нейронные сети*, *вероятностные рассуждения* и *сети доверия*. Данные технологии дополняют друг друга и используются в различных комбинациях или самостоятельно для создания интеллектуальных систем.

Наиболее развитый класс эволюционных алгоритмов – генетические алгоритмы. Также большое развитие получили эволюционная стратегия и эволюционное программирование. В сравнении с указанными алгоритмами, алгоритм дифференциальной эволюции разработан в существенно меньшей степени.

Алгоритмы эволюционной стратегии и эволюционного программирования имеют ряд важных общих свойств, так что существует даже точка зрения, в соответствии с которой оба эти алгоритма, в сущности, одинаковы и представляют собой модификации генетического алгоритма. При разработке алгоритмов эволюционной стратегии и эволюционного программирования в качестве базовой принята следующая концепция: эволюция популяции заключается прежде всего в приспособлении особей популяции к окружающей среде на поведенческом уровне, а не на генетическом уровне, как в генетическом алгоритме. Следствием этой концепции является отсутствие в указанных методах рекомбинации особей, а значит, и оператора кроссовера. Единственным оператором в этих методах, обеспечивающим поиск альтернативных решений, является оператор мутации.

Глава посвящена последовательным неадаптивным эволюционным алгоритмам для решения одноцелевой задачи оптимизации. В первом параграфе главы изложены биологические предпосылки и общая схема эволюционных алгоритмов. Второй параграф посвящен методам кодирования особей в эволюционных алгоритмах. В трех последующих параграфах рассмотрены основные операторы эволюционных алгоритмов – операторы мутации, скрещивания и отбора особей. Шестой параграф содержит описание некоторых других операторов и процедур, используемых в эволюционных алгоритмах.

Седьмой параграф включает в себя перечень типовых генетических алгоритмов. Восьмой параграф посвящен теории шим, являющейся методологической основой эволюционных алгоритмов. В последующих четырех параграфах последовательно рассмотрена эволюционная стратегия, эволюционное программирование, дифференциальная эволюция и генетический коэволюционный алгоритм. В заключительном параграфе приведен пример использования генетического алгоритма для решения одной задачи оптимального управления.

2.1. БИОЛОГИЧЕСКИЕ ПРЕДПОСЫЛКИ И ОБЩАЯ СХЕМА ЭВОЛЮЦИОННЫХ АЛГОРИТМОВ

Первые эволюционные алгоритмы – генетические алгоритмы – предложил профессор Мичиганского университета Холланд (*J. Holland*) в 60-е гг. прошлого века. Элементы этих алгоритмов исследованы ранее в работах других авторов. Генетические алгоритмы получили всеобщее признание после выхода в 1975 г. в свет книги Холланда, ставшей классикой в этой области, «Адаптация в естественных и искусственных системах» (*Adaptation in Natural and Artificial Systems*). Холланд отмечал, что разработку им теории генетических алгоритмов стимулировало чтение биологической литературы по дарвиновской теории естественного отбора и селекции сельскохозяйственных культур.

Наряду с моделью эволюции Дарвина (*Ch. R. Darwin*), положенной в основу канонического и большинства современных эволюционных алгоритмов, известно значительное число других моделей эволюции – модель Ламарка, модель Фриза, модель Гаулда и Элдриджа и другие. На основе принципов, заложенных в этих моделях, построены соответствующие варианты эволюционных алгоритмов. Некоторые идеи указанных моделей могут быть использованы и используются также для модификации классических генетических алгоритмов.

Доступно чрезвычайно большое число источников, посвященных различным аспектам эволюционных алгоритмов, и число это продолжает быстро увеличиваться. Так, в настоящее время запрос «*genetic algorithm*» в поисковой системе *Google* дает более четырех миллионов ссылок. Поэтому сколь-нибудь исчерпывающее изложение теории и практики эволюционных алгоритмов в рамках одной главы невозможно.

Чарльз Дарвин в его знаменитой работе «Происхождение видов» (1859) показал, что эволюционное развитие земной флоры и фауны происходит под влиянием окружающей среды на основе следующих принципов: наследственность (потомки сохраняют свойства родителей), изменчивость (потомки почти всегда неидентичны), естественный отбор (выживают наиболее приспособленные потомки).

В 1944 г. Эвери (*O. Avery*), Маклауд (*C. MacLeod*) и Маккарти (*M. McCarty*) доказали, что дезоксирибонуклеиновая кислота (ДНК) является веществом, определяющим наследственные процессы. В 1953 г. Крик (*F. Crick*) и Уотсон (*J. D. Watson*) расшифровали структуру ДНК в виде двойной спирали. Таким

образом, в начале 60-х гг. XX века стали известны молекулярно-биологические основы наследственности и изменчивости видов, которые и легли в основу теории генетических алгоритмов, а позднее – в основу других эволюционных алгоритмов.

В каждой клетке любого живого существа содержится вся генетическая информация этого существа, записанная в молекулах ДНК, представляющих собой цепочку из молекул нуклеотидов четырех типов, обозначаемых *A*, *T*, *C*, *G*. Генетическую информацию несет порядок следования нуклеотидов в ДНК. Можно сказать, что генетический код индивидуума – это очень длинное слово, алфавит которого содержит всего четыре буквы.

Структуру, содержащую ДНК, называют *хромосомой* (*chromosome*). Каждое врожденное качество особи (цвет волос, цвет глаз, наследственные болезни и т. д.) кодирует определенная часть хромосомы – *ген* (*gene*) этого свойства. Место гена в хромосоме называют *локусом* (*locus*), а функциональное значение гена – его *аллелем* (*allele*). Совокупность генов данной особи называют *генотипом* (*genotype*). На основе генотипа формируется *фенотип* (*phenotype*) – совокупность характеристик, присущих особи на определенной стадии ее развития.

При размножении живых существ происходит слияние двух родительских половых клеток, и их ДНК взаимодействуют, образуя ДНК потомка. Основным способом взаимодействия ДНК является их *скрещивание* (*mating*, *crossing*), при котором ДНК родителей делятся на две части, а затем обмениваются этими своими частями. Как синонимы используют также термины *кроссовер* (*crossover*) и *рекомбинация* (*recombination*).

Из-за радиоактивности и/или других факторов некоторые гены в половых клетках родителей могут изменяться – могут происходить *мутации* (*mutation*) генов. В результате скрещивания измененные гены могут передаваться потомку, придавая ему свойства, отсутствующие у родителей. Если эти новые свойства полезны, то они, скорее всего, сохранятся в данном виде и приведут к повышению его приспособленности.

Схема канонического генетического алгоритма имеет следующий вид.

1) Агентов (особей, индивидуумов) представляем в виде хромосом.

2) Случайным образом создаем некоторое число исходных особей – *начальную популяцию*.

3) Особи оцениваем с помощью фитнес-функции – каждой особи ставим в соответствие определенное значение *приспособленности* (*fitness*), которое определяет вероятность ее выживания.

4) На основе приспособленностей выбираем особи для скрещивания – этап *селекции* (*selection*). К хромосомам этих особей применяем *генетические* (*genetic*) *операторы* скрещивания и мутации, создавая таким образом следующее *поколение* (*generation*) особей.

5) Особи созданного поколения также оцениваем, производим селекцию, применяем генетические операторы и так далее до тех пор, пока не будет выполнен критерий останова алгоритма.

Суть эволюционных алгоритмов, как и популяционных алгоритмов в целом, состоит в обеспечении более высокой средней приспособленности нового поколения по сравнению с такой же приспособленностью предыдущего поколения.

Известно очень большое число вариантов рассмотренной схемы генетического алгоритма, отличающихся способами кодирования хромосом, набором генетических операторов, структурой и параметрами алгоритма и т. д.

Рассматриваем задачу глобальной условной оптимизации

$$\max_{X \in D \subset R^{|X|}} f(X) = f(X^*) = f^* \tag{2.1}$$

с ограничениями общего вида (см. 1.1). Полагаем, что большим значениям целевой функции $f(X)$ соответствуют большие значения фитнес-функции $\varphi(X)$.

Закодированный вектор варьируемых параметров X, то есть хромосому соответствующей особи s, обозначаем $H = (H_i^g,\ i \in [1:|X|])$. Здесь ген H_i^g представляет собой код компоненты x_i вектора X; i – номер локуса, в котором находится этот ген. Аллель гена H_i^g есть не что иное, как совокупность всех возможных значений величины x_i.

С другой стороны, хромосома H представляет собой совокупность $h_1, h_2, ..., h_{|H|}$ символов некоторого алфавита, с помощью которого выполнено кодирование вектора X, так что $H = (h_1, h_2, ..., h_{|H|})$, а наборы $(h_1, h_2, ..., h_{|H_1^g|})$, $(h_{|H_1^g|+1}, h_{|H_1^g|+2}, ..., h_{|H_1^g|+|H_2^g|})$,... представляют коды генов H_1^g, H_2^g,... соответственно. Здесь $|H|$ – длина хромосомы, $|H_i^g|$ – размерность i-го гена; $i \in [1:|X|]$. Легко видеть, что имеет место соотношение

$$\sum_{i=1}^{|X|} |H_i^g| = |H|.$$

Особь s, таким образом, определяет тройка

$$s = \langle X, H, \varphi(X) \rangle,$$

где X – фенотип особи, H – ее генотип, а $\varphi(X)$ – приспособленность этой особи. Наряду с $\varphi(X)$ в том же смысле будем писать $\varphi(H)$ и $\varphi(s)$.

Для простоты записи в ситуациях, в которых не может возникнуть неверное чтение, будем далее писать «ген H_i^g особи s», имея в виду ген H_i^g хромосомы H, с помощью которой выполнено кодирование фенотипа этой особи. Аналогично будем говорить «разряд h_i особи s», имея в виду разряд h_i хромосомы H.

Основными являются следующие генетические операторы:

– оператор мутации, реализующий случайное изменение одного или нескольких генов хромосомы;

– оператор скрещивания, предназначенный для создания особей-потомков путем рекомбинации хромосом двух или более родительских особей;

– оператор управления популяцией, формирующий на основе популяции $S = S(t)$ текущего поколения t *промежуточную* (*intermediate*) *популяцию* $S' = (s_{i_j},\ i_j \in [1:|S|],\ j \in [1:|S'|])$, где $|S'|$ – число особей в этой популяции;

– оператор селекции, осуществляющий выбор из промежуточной популяции S' особей для скрещивания и формирование на этой основе популяции $S(t+1)$ нового поколения $(t+1)$.

В простейшем случае промежуточная популяция может совпадать с текущей популяцией.

Указанные операторы используют, как правило, некоторое число свободных параметров. Эти параметры могут изменять свои значения в процессе вычислений либо в функции числа поколений алгоритма, либо адаптивно – на основе использования обратной связи их значений с эффективностью процесса поиска. Операторы с такими параметрами называют *неравномерными* и *адаптивными операторами* соответственно.

Для эволюционных алгоритмов вообще и для генетических алгоритмов в частности остро стоит вопрос сходимости. В некоторой степени ответ на этот вопрос дает следующая теорема.

Теорема 2.1. Положим, что последовательность популяций $S(t)$, $t = 0, 1, \ldots$, генерируемая генетическим алгоритмом, монотонна, то есть для всех t удовлетворяет условию

$$\max\,(\varphi(s_i) \mid s_i \in S(t+1)) \geq \max\,(\varphi(s_j) \mid s_j \in S(t)).$$

Пусть также для любых двух допустимых особей s, s' возможен переход от первой из них ко второй посредством однократного или многократного применения операторов мутации и кроссовера. Тогда генетический алгоритм отыскивает глобально оптимальную особь s^*, доставляющую максимум функции $\varphi(s)$ с вероятностью ξ, равной единице, т. е.

$$\lim_{t \to \infty} (\xi(s^* \in S(t)) = 1 \bullet$$

Первое из указанных в теореме условий является весьма сильным и в реальных генетических алгоритмах без дополнительных усилий обычно не выполняется. Второе условие, можно считать, выполняется всегда. В целом доказательство сходимости различных комбинаций генетических операторов представляет собой современную актуальную научную задачу.

2.2. КОДИРОВАНИЕ ОСОБЕЙ

Чаще всего используют *бинарное* либо *вещественное кодирование* генов в хромосоме, а также большое число способов кодирования, приводящих к следующим основным типам особей:

– монохромосомные;
– мультихромосомные;
– адаптивные;
– символьные;
– вещественные.

При бинарном кодировании используют обычные двоичные коды или двоичные коды Грея. Последние в большинстве случаев обеспечивают более высокую эффективность эволюционных алгоритмов. Это связано с тем, что расстояние Хемминга между обычными бинарными хромосомами, как мера их близости, может не отражать близость соответствующих фенотипов (например, числа семь и восемь различаются в этом случае в четырех разрядах). В случае использования кода Грея такая ситуация не имеет места.

Монохромосомы

Классическим способом кодирования является кодирование значений варьируемых параметров задачи в виде монохромосомы, то есть в виде бинарной строки фиксированной длины вида $H = (h_i, i \in [1:|H|])$. Битовую подстроку строки H, соответствующую гену H_j^g, $j \in [1:|X|]$, обозначим $(h_{j,k}, k \in [1:|H_j^g|])$.

Рассмотрим для примера двумерную $(|X| = 2)$ задачу (2.1) с фитнес-функцией $\varphi(X) = f(X) = f(x_1, x_2) = x_1^2 + x_2^2$, где величины x_1, x_2 могут принимать только целочисленные значения. Положим, $|H_1^g| = |H_2^g| = 4$ и $x_1 = 10$, $x_2 = 12$. Тогда, легко видеть, что $|H| = 8$, и имеет место картина, представленная в табл. 2.1.

Таблица 2.1 **Пример кодирования в виде монохромосомы**

Гены	H_1^g				H_2^g			
Хромосома	1	0	1	0	1	1	0	0
Локусы	1				2			
Параметры	x_1				x_2			
Значения параметров	10				12			
$\varphi(x_1, x_2)$	244							

Кодирование в хромосоме дробных чисел может быть осуществлено с помощью *модифицированных монохромосом*, в которых для бинарного представления дробной части отводят некоторое фиксированное число разрядов, определяемое требуемой точностью вычислений. Положим, что в рассмотренном примере для представления как целой, так и дробной частей компонент вектора X достаточно четырех двоичных разрядов, так что $|H_1^g| = |H_2^g| = 8$, и монохромосома имеет вид, представленный в табл. 2.2.

Таблица 2.2 **Пример кодирования в виде модифицированной монохромосомы**

Гены	H_1^g								H_2^g							
	Целая часть				Дробная часть				Целая часть				Дробная часть			
Хромосома	1	0	1	0	0	0	1	0	1	1	0	0	1	0	0	0
Локусы	1								2							
Параметры	x_1								x_2							
Значения параметров	10,125								12,5							
$\varphi(x_1, x_2)$	258,765625															

Для уменьшения длины монохромосомы может быть использовано *логарифмическое* (*logarithmic*) *кодирование*. В этом случае ген H_i^g кодируют в виде двоичного числа $(h_{i,1}, h_{i,2}, ..., h_{i,|H_i^g|})$ такого, что

$$\left[h_{i,j},\ j \in [1:|H_i^g|]\right]_{(10)} = (-1)^{h_{i,2}} \exp\left((-1)^{h_{i,1}} \left[h_{i,3}, ..., h_{i,|H_i^g|}\right]_{(10)}\right),$$

где запись вида $[h_1, h_2, ...]_{(10)}$ означает десятичное значение числа, закодированного в указанной в скобках двоичной последовательности. Таким образом, второй бит $h_{i,2}$ гена H_i^g кодирует знак числа, первый бит $h_{i,1}$ – знак степени экспоненциальной функции, а остальные биты $h_{i,3}, h_{i,4}, ...$ – саму эту степень.

Если, например, $H_i^g = 10110$, то значение соответствующего параметра равно $x_i = (-1)^0 \exp\left((-1)^1 [110]_{(10)}\right) = e^{-6} \approx 0{,}002478752$.

Мультихромосомы

В случае кодирования особей в виде мультихромосомы используют структурирование хромосомы так, что каждая ее часть отвечает за определенный аспект решения. При этом генетические операторы применяют к каждой из частей хромосомы в отдельности, не скрещивая эти части между собой, а значения фитнес-функции вычисляют для всей хромосомы в целом. Различные гены мультихромосомы могут кодировать как целые, так и дробные числа.

Рассмотрим решение задачи структурно-параметрического синтеза динамической системы путем сведения ее к задаче оптимизации. Пусть параметры x_i, $i \in [1:n]$ определяют структуру системы (к примеру, типы возможных нелинейностей), а параметры x_j, $j \in [(n+1):|X|]$ – значения ее коэффициентов (например, коэффициентов передаточных функций линейных частей системы). Тогда хромосому целесообразно разделить на две части, первая из которых определяет структуру системы, а вторая – значения ее коэффициентов (табл. 2.3).

Таблица 2.3 **Пример мультихромосомы**

Гены и параметры	Структура системы				Коэффициенты системы			
Гены	H_1^g	H_2^g	$\cdots$	H_n^g	H_{n+1}^g	H_{n+2}^g	$\cdots$	$H_{\|X\|}^g$
Параметры	x_1	x_2	$\cdots$	x_n	x_{n+1}	x_{n+2}	$\cdots$	$x_{\|X\|}$

Адаптивное кодирование

Адаптивное кодирование особи в общем случае предполагает, что в процессе итерации эволюционного алгоритма хромосома меняет свою структуру и длину, так что $|H_i^g| \neq const$ и $|H| \neq const$; $i \in [1:|X|]$.

Положим, что рассмотренная выше задача структурно-параметрического синтеза динамической системы решается в классе линейных систем с двумя

неизвестными передаточными функциями. Тогда хромосому целесообразно сконструировать так, что структуру системы определяют гены H_1^g, H_2^g, кодирующие варианты первого и второго линейных звеньев системы соответственно.

Если значения генов в этих наборах определяют два апериодических звена (каждое из которых имеет два свободных коэффициента), то структуру мультихромосомы представляет табл. 2.4.

Таблица 2.4 **Мультихромосома для двух апериодических звеньев**

Гены и параметры	Структура системы		Коэффициенты системы			
			Первое звено		Второе звено	
Гены	H_1^g	H_2^g	H_3^g	H_4^g	H_5^g	$H_{\|X\|}^g = H_6^g$
Параметры	x_1	x_2	x_3	x_4	x_5	$x_{\|X\|} = x_6$

Структура мультихромосомы для случая, когда определены два колебательных звена (каждое из которых имеет три свободных коэффициента), представлена в табл. 2.5.

Таблица 2.5 **Мультихромосома для двух колебательных звеньев**

Гены и параметры	Структура системы		Коэффициенты системы					
			Первое звено			Второе звено		
Гены	H_1^g	H_2^g	H_3^g	H_4^g	H_5^g	H_6^g	H_7^g	$H_{\|X\|}^g = H_8^g$
Параметры	x_1	x_2	x_3	x_4	x_5	x_6	x_7	$x_{\|X\|} = x_8$

Заметим, что в данном примере структуры системы можно было бы кодировать не двумя, а всего одним геном.

Символьное кодирование

Кодирование, используемое в монохромосоме, имеет следующий серьезный недостаток. При высокой размерности вектора варьируемых параметров X и высокой требуемой точности представления компонент этого вектора число битов в особи оказывается чрезвычайно большим, что может значительно замедлить работу эволюционного алгоритма.

При *символьном кодировании* интервал $[x_i^-; x_i^+]$ изменения параметра x_i, $i \in [1:|X|]$ разбивают на фиксированное число подынтервалов n_i, каждый из которых кодируют уникальным набором битов. Таким путем суммарная длина каждого из генов H_i^g, а значит, и длина всей хромосомы H могут быть существенно сокращены, а скорость выполнения генетических операций увеличена.

Пусть, для примера, параметр x_i может принимать только целочисленные значения и только в интервале $[0; 80]$. Разобьем этот интервал на $n_i = 8$ подынтервалов $\Delta x_{i,j}$, $j \in [1:n_i]$ и поставим в соответствие подынтервалу

$\Delta x_{i,j}$ битовую подстроку $H_{i,j}^{g}$, представляющую собой его бинарный код. Соответствующую символьную кодировку параметра x_i иллюстрирует табл. 2.6.

Легко видеть, что для кодирования рассматриваемого параметра x_i в виде монохромосомы понадобилось бы семь битов. Символьное кодирование позволило уменьшить это число до трех.

Таблица 2.6 **Пример символьного кодирования**

Ген	H_i^g							
Подстрока	$H_{i,1}^g$	$H_{i,2}^g$	$H_{i,3}^g$	$H_{i,4}^g$	$H_{i,5}^g$	$H_{i,6}^g$	$H_{i,7}^g$	$H_{i,8}^g$
Бинарный код	000	001	010	011	100	101	110	111
$\Delta x_{i,j}$	$[0;10)$	$[10;20)$	$[20;30)$	$[30;40)$	$[40;50)$	$[50;60)$	$[60;70)$	$[70;80]$

Известны исследования, показывающие, что помимо отмеченного уменьшения длины хромосомы у символьного кодирования есть еще одно достоинство – эволюционный алгоритм, использующий такое кодирование, обеспечивает более полный обзор пространства решений и повышает вероятность локализации глобального экстремума фитнес-функции.

В то же время символьные коды не свободны от недостатков. Прежде всего, символьное кодирование ставит в соответствие одному генотипу множество фенотипов. В результате снижается точность решения задачи, поскольку вместо конкретного вектора X^* получаем лишь гиперпараллелепипед вида $\Delta x_{1,j_1} \times \Delta x_{2,j_2} \times \ldots \times \Delta x_{|X|,j_{|X|}}$, $j_i \in [1:n_i]$, которому принадлежит решение. Данный недостаток в значительной мере может быть устранен путем перехода от точного значения фенотипа к его вероятностному представлению, когда фенотип, соответствующий каждому из генотипов, полагаем случайной величиной, равномерно распределенной в соответствующем гиперпараллелепипеде.

Кроме того, недостатком символьных кодов является быстрый рост числа интервалов n_i с ростом требуемой точности представления величин x_i, $i \in [1:|X|]$. Преодолеть этот недостаток удается только путем использования вещественного кодирования.

Вещественное кодирование

При решении задач оптимизации в непрерывных пространствах бинарное и символьное кодирование являются весьма ограничительными. Поэтому разработано вещественное кодирование, когда гены в хромосоме напрямую представляют в виде вещественных чисел, точность которых ограничена только используемой в ЭВМ разрядной сеткой. Естественно, что такое кодирование требует использования специальных генетических операторов. Алгоритм получил название *непрерывного генетического алгоритма* (*Real-coded Genetic Algorithm*, *RGA*). По аналогии с этим назовем генетический алгоритм, использующий бинарное кодирование, *бинарным*.

Развитие вещественного кодирования долгое время сдерживало отсутствие для непрерывного генетического алгоритма теоремы, аналогичной теореме шим для бинарного кодирования (см. 2.8), которая, по сути, является математической основой бинарного генетического алгоритма. В настоящее время такие теоремы имеются, и вещественное кодирование широко используют в вычислительной практике.

В качестве еще одного резона использования вещественного кодирования укажем на следующий недостаток бинарного эволюционного алгоритма. Если не предпринимать специальных усилий, то на первых итерациях этот алгоритм, по сути, бесполезно тратит значительные вычислительные ресурсы на обработку младших разрядов чисел, закодированных в каждом гене хромосомы. Дело в том, что на этих итерациях (вдали от экстремума фитнес-функции) варьирование значений этих разрядов практически не меняет значений данной функции. Аналогично на завершающих итерациях бинарный алгоритм выполняет бесполезные операции над старшими разрядами генов. Для преодоления указанного недостатка бинарного эволюционного алгоритма приходится использовать ряд специальных приемов.

Наконец заметим, что для обеспечения достаточной эффективности бинарного эволюционного алгоритма с увеличением длины хромосом приходится пропорционально увеличивать численность популяции, что приводит к росту вычислительных затрат на реализацию алгоритма.

При вещественном кодировании каждый ген $H_i^g, i \in [1:|X|]$ хромосомы, соответствующей вектору X, представляет собой значение параметра x_i этого вектора, так что $H_i^g = h_i = x_i$ и $|H| = |X|$. Другими словами, при вещественном кодировании генотип особи идентичен ее фенотипу. Вещественное кодирование оказывается весьма удобным для задачи непрерывной оптимизации в силу близости к ее постановке. В качестве преимущества вещественного кодирования отметим также исключение операций кодирования/декодирования, что снижает вычислительные затраты на реализацию самого эволюционного алгоритма.

Из числа генетических операторов только оператор отбора пригоден как для бинарного, так и для непрерывного эволюционного алгоритма. Операторы скрещивания и мутации, используемые в классических реализациях эволюционных алгоритмов, не позволяют работать с вещественными хромосомами, и требуется разработка новых таких операторов.

Прочие варианты кодирования

Более широко распространен взгляд на кодирование хромосом, отличный от приведенного в начале данного параграфа. Выделяют только две группы хромосом – монохромосомы и мультихромосомы. При этом среди мультихромосом выделяют семейства *гомоморфных* и *негомоморфных мультихромосом*.

Монохромосомы содержат по одной хромосоме, а мультихромосомы включают в себя две и более хромосом. Гомоморфными называют мультихромосомы, содержащие морфологически (структурно) и генетически близкие хромосомы, то есть хромосомы, имеющие одинаковое число и раз-

рядность генов во всех особях популяции. Негомоморфными являются мультихромосомы, содержащие морфологически и/или генетически разные хромосомы.

В соответствии с указанной точкой зрения выделяют три способа кодирования генов в хромосомах – двоичное, числовое и векторное.

Двоичное кодирование использует алфавит {0;1} и, иногда, алфавит {0;1;*}, где символ «*» имеет смысл «неопределенное значение».

Числовое кодирование предполагает, что гены могут принимать целочисленные либо вещественные значения в заданных интервалах.

Векторное кодирование применяется в том случае, когда каждый ген соответствует вектору чисел, так что, по сути, каждый локус такого гена содержит хромосому, использующую двоичное или числовое кодирование. Таким образом, векторное кодирование позволяет реализовать мультихромосомное представление особей, используя произвольную комбинацию двоичных, числовых и векторных хромосом. Следует помнить, что для таких мультихромосом приходится использовать специальные генетические операторы.

При решении задач непрерывной поисковой оптимизации в подавляющем большинстве случаев используют бинарные либо вещественные монохромосомы. Именно они и будут рассмотрены далее.

2.3. ОПЕРАТОРЫ МУТАЦИИ

Сутью мутации является случайное изменение хромосомы. Известна точка зрения, в соответствии с которой оператор мутации является основным поисковым оператором эволюционных алгоритмов. Соответственно, существуют алгоритмы, использующие только генетический оператор мутации как оператор «бесполого» размножения и оператор селекции.

Сначала обсуждаем операторы мутации, ориентированные на бинарную кодировку особей,– бинарные (*binary*) мутаторы. Затем рассматриваем аналогичные операторы для мутации вещественных хромосом – вещественные (*real valued*) мутаторы.

2.3.1. Бинарные мутаторы

Выделяем два класса бинарных операторов мутации:

– операторы, использующие инверсию одного либо нескольких битов родительской особи;

– операторы, основанные на перестановке битов этой особи.

Операторы с инверсией битов

Оператор равномерной хромосомной мутации. Суть оператора заключается в замене одного из разрядов бинарной особи на противоположный. Схему равномерной хромосомной мутации определяет следующая последовательность шагов.

1) Генерируем случайное число $u_1 = U_1(0;1)$. Если $u_1 \le \xi_m$, то полагаем, что мутация родительской особи не нужна и завершаем вычисления. Здесь $\xi_m \in (0;1]$ – свободный параметр оператора.

2) Генерируем натуральное случайное число $u_2 = U_1(1:|H|)$. Если $u_2 = i$, то бит h_i родительской хромосомы H инвертируем, так что хромосома-потомок приобретает вид

$$H' = (h_1, ..., h_{i-1}, \overline{h}_i, h_{i+1}, ..., h_{|H|}).$$

Если не оговорено противное, здесь и далее надчеркивание символа разряда бинарной хромосомы означает инверсию соответствующего бита этой хромосомы.

Величина ξ_m имеет, очевидно, смысл вероятности мутации особи и обычно принимается равной ~0,01. Известны рекомендации по выбору этой вероятности $\xi_m = \frac{1}{|X|}$ или $\xi_m = \frac{1}{|H|}$.

Легко видеть, что при использовании равномерной хромосомной мутации вероятность мутации одного бита хромосомы

$$\xi = \frac{\xi_m}{|H|}.$$

Оператор равномерной (*density*) *генной мутации*. В этом случае с некоторой вероятностью может быть инвертирован каждый бит родительской особи. Схема равномерной генной мутации представлена ниже.

1) Генерируем случайное число $u = U_1(0;1)$. Если $u \le \xi'$, то полагаем, что мутация родительской особи S не нужна и завершаем вычисления.

2) Генерируем вектор случайных чисел $U = U_{|H|}(0;1)$.

3) Если $u_i \ge \xi$, то бит h_i особи s подлежит мутации, так что имеет место равенство

$$H' = (h_1, ..., h_{i_1-1}, \overline{h}_{i_1}, h_{i_1+1}, ..., h_{i_2-1}, \overline{h}_{i_2}, h_{i_2+1}, ..., h_{|H|}),$$

где принято, что неравенства $u_i \ge \xi$ справедливы для $i = i_1, i_2$.

Здесь ξ', ξ – свободные параметры оператора со значениями в интервале $(0;1]$. Последний параметр имеет смысл вероятности мутации бита в хромосоме. Величины ξ', ξ рекомендуют выбирать такими, чтобы в среднем мутировало от 1 до 10% битов данной хромосомы.

Очевидная модификация данной схемы заключается в том, что пороговое значение $\xi_i \in (0;1]$ задаем для каждой из величин u_i и бит h_i инвертируем при выполнении условия $u_i \ge \xi_i$; $i \in [1:|H|]$. Свободные параметры оператора в этом случае образуют вектор $(\xi_1; \xi_2; ...; \xi_{|H|})$. Такой модифицированный мутатор называют *неравномерным*. Вероятность ξ_i инверсии разряда h_i в общем случае может быть динамической величиной, зависящей как от генотипа, так и от фенотипа особи. Например, эта вероятность может зависеть от длины хромосомы $|H|$, номера локуса i, числа разрядов $|H_i^g|$ гена H_i и т. д. В бинарном эволюционном алгоритме неравномерная генная мутация может быть использована для того, чтобы повысить вероятность изменения младших разрядов гена относительно вероятности изменения его старших разрядов и наоборот.

Операторы равномерной и неравномерной генной мутации относят к классу *операторов макромутации*, которые предполагают инверсию с некоторой

вероятностью значений группы разрядов особи и дают возможность за одну итерацию эволюционного алгоритма выполнить мутацию ее нескольких генов. Тем самым расширяется пространство поиска и может быть повышена эффективность соответствующего алгоритма.

Операторы с перестановкой битов

Оператор обменной (*swap*) *мутации* использует следующий принцип мутации: в хромосоме $H = (h_1, h_2, ..., h_{|H|})$ случайным образом выбирают два разряда (не обязательно соседних), и эти разряды обмениваются своими битами. Если, например, в хромосоме *H* указанным образом выбраны разряды h_{i_1}, h_{i_2}, то после мутации эта хромосома приобретает вид

$$H' = (h_1, ..., h_{i_1-1}, h_{i_2}, h_{i_1+1}, ..., h_{i_2-1}, h_{i_1}, h_{i_2+1}, ..., h_{|H|}),\ 1 < i_1 < i_2 < |H|.$$

Оператор зеркальной мутации формирует хромосому H' путем зеркального отражения всех или части битов хромосомы *H*. Схема выполнения оператора имеет следующий вид.

1) Генерируем натуральные случайные числа $i_1 = U_1(1:|H|)$, $i_2 = U_1(1:|H|)$, такие, что $1 \le i_1 < i_2 \le |H|$.

2) Хромосому H' получаем путем зеркального отражения в хромосоме H фрагмента, включающего в себя биты h_i, $i \in [i_1 : i_2]$, относительно середины этого фрагмента:

$$H' = (h_1, ..., h_{i_1-1}, h_{i_2}, ..., h_{i_1}, h_{i_2+1}, ..., h_{|H|}).$$

Оператор транспозиции можно считать модификацией зеркального мутатора. Схему оператора определяет следующая последовательность шагов.

1) По схеме предыдущего оператора генерируем натуральные случайные числа i_1, i_2, i_3, удовлетворяющие неравенству $1 \le i_1 < i_2 < i_3 \le |H|$.

2) Хромосому H' получаем путем зеркального отражения фрагмента хромосомы *H*, находящегося в позициях с номерами от i_2 до i_3, и перемещения полученного набора битов в позиции хромосомы H', начиная с позиции i_1 (смещая, конечно, оставшиеся биты):

$$H' = (h_1, ..., h_{i_1}, h_{i_3}, ..., h_{i_2}, h_{i_1+1}, ..., h_{|H|}).$$

2.3.2. Вещественные мутаторы

Операторы вещественной мутации заменяют подлежащий мутации ген x_i на ген x'_i, $i \in [1:|X|]$. Заметим, что, вообще говоря, вещественные мутаторы могут быть сконструированы на основе рассмотренных выше бинарных мутаторов по следующей схеме:

– сначала по правилам функционирования бинарного мутатора реализуем выбор подлежащих мутации генов;

– затем по схеме того или иного сугубо вещественного мутатора осуществляем вещественную мутацию этих генов.

Оператор случайной (*random*) *мутации* присваивает гену x_i случайное число из интервала $[x_i^-; x_i^+]$. Схема простейшего оператора случайной мутации имеет следующий вид.

1) Генерируем несовпадающие натуральные случайные числа $i_j = U_1(1:|X|)$, $j \in [1:\nu]$, где ν – число подлежащих мутации генов (свободный параметр).

2) Генерируем случайное число $u_1 = U_1(x_1^-; x_1^+)$ и полагаем, что ген-потомок имеет значение, равное $x'_{i_1} = u_1$.

3) По той же схеме выполняем мутацию гена x_{i_2} и так далее до гена x_{i_ν}.

Частным случаем оператора случайной мутации можно считать *оператор граничной мутации*, когда с равной вероятностью гену x_i присваивают значение x_1^- или значение x_1^+.

Оператор гауссовой (*Gauss*) *мутации* заменяет родительский ген x_i случайным значением x'_i по правилу

$$x'_i = x_i + N_1(m, \sigma); \; i \in [1:|X|].$$

Здесь m, σ – свободные параметры оператора. Обычно полагают $m = 0$.

Арифметический оператор вещественного сдвига (*arithmetic real number creep operator*) определяет новое значение гена x_i в соответствии с формулой

$$x'_i = x_i - b(2u - 1), \; i \in [1:|X|],$$

где случайное число $u = U_1(0;1)$, b – свободный параметр (коэффициент сдвига). Легко видеть, что величина x'_i, вычисленная по этой формуле, оказывается равномерно распределенной в интервале $[x_i - b; x_i + b]$.

Геометрический оператор вещественного сдвига (*geometrical real number creep operator*) значение гена-потомка x'_i определяет по аналогичной формуле

$$x'_i = x_i - x_i b(2u - 1), \; i \in [1:|X|],$$

так что величина x'_i оказывается равномерно распределенной в интервале $x_i[(1-b); (1+b)]$. Отметим, что величина x'_i в этом случае пропорциональна величине x_i и имеет тот же знак, что и эта величина.

BGA-оператор мутации (*Breeder Genetic Algorithm operator*). В соответствии с алгоритмом, используемым данным оператором, особь s' находят случайным образом в окрестности особи s по формуле

$$x'_i = x_i \pm b\left(x_i^+ - x_i^-\right)\delta_i, \; i \in [1:|X|].$$

Здесь знаки плюс, минус выбирают с равной вероятностью; b – свободный параметр оператора (рекомендуемое значение равно 0,5); δ_i – диаметр указанной окрестности (величина шага мутации). Известно несколько близких правил определения последней величины, например правило

$$\delta_i = 2^{-k\,u} \in [2^{-k}; 1], \; u = U_1(0;1),$$

где $k \in [4:20]$ – целый свободный параметр, определяющий диапазон возможных значений δ_i.

Степенной (*power*) *оператор мутации* вычисляет значение гена x_i' на основе значения гена x_i по формуле

$$x_i' = \begin{cases} x_i - u_1(x_i - x_i^-), & \delta_i < u_2, \\ x_i + u_1(x_i^+ - x_i), & \delta_i \ge u_2, \end{cases} \quad i \in [1:|X|],$$

где u_1 – случайная величина, плотность распределения вероятности которой $\xi(v) = bv^{b-1}$, $v \in [0;1]$; b – свободный параметр оператора, определяющий интенсивность мутаций; $u_2 = U_1[0;1]$;

$$\delta_i = \frac{x_i - x_i^-}{x_i^+ - x_i}.$$

Оператор неравномерной мутации Михалевича (*Michalewicz's non-uniform operator*) относится к классу нестационарных мутаторов. Новое значение гена x_i, подлежащего изменению, в этом случае вычисляют по формуле

$$x_i' = \begin{cases} x_i + \delta(t, x_i^+ - x_i), & u_1 = 0, \\ x_i - \delta(t, x_i - x_i^-), & u_1 = 1, \end{cases} \quad i \in [1:|X|].$$

Здесь $u_1 = u_{sign}^{\pm}$ – целое случайное число, с равной вероятностью принимающее значение 0 или 1; функцию $\delta(t, y)$ определяет выражение

$$\delta(t, y) = y\left(1 - u_2^{\left(1-\frac{t}{\hat{t}}\right)^b}\right),$$

где случайное число $u_2 = U_1(0;1)$; b – свободный параметр оператора, определяющий степень зависимости значений функции от числа поколений (рекомендуемое значение $b = 5$).

Легко видеть, что функция $\delta(t, y)$ принимает значения в интервале $[0; y]$ и с ростом номера поколения t этот интервал сужается. В результате на начальных итерациях эволюционного алгоритма мутатор Михалевича оказывается близким к оператору случайной мутации, а на завершающих итерациях производит мутации, обеспечивающие близость значений величин x_i и x_i'.

Динамический (*dynamic*) *оператор мутации* базируется на оценке близости между родительскими особями s_1, s_2. В качестве меры близости оператор использует величину

$$\rho(s_1, s_2) = 1 - \frac{1}{|X|}\sum_{i=1}^{|X|}\left|\frac{x_{1,i} - x_{2,i}}{x_i^+ - x_i^-}\right|^b,$$

так что $\rho(s, s) = 0$. Здесь b – свободный параметр оператора, рекомендуемое значение которого равно 0,2.

Оператор определяет только вероятность мутации особей s_1, s_2, но не регламентирует собственно процедуру мутации. Указанную вероятность полагают равной

$$\xi_m(s_1, s_2) = c\,\rho(s_1, s_2),$$

где c – свободный параметр с рекомендуемым значением 0,9.

Данный оператор обеспечивает более высокую вероятность мутации генетически близких особей, чем далеких. Такое поведение оператора призвано уменьшить вероятность преждевременной сходимости эволюционного алгоритма.

2.4. ОПЕРАТОРЫ СКРЕЩИВАНИЯ (КРОССОВЕРЫ)

Суть оператора скрещивания заключается в создании особей-потомков (*children*) из нескольких особей-родителей (*parents*) путем рекомбинации (*recombination*) их хромосом. В результате потомки наследуют часть свойств от каждого из родителей. Иногда оператор скрещивания (кроссовера) называют кроссинговером. Подобно оператору мутации, последовательно рассмотрим бинарные и вещественные кроссоверы.

2.4.1. Бинарные кроссоверы

Бинарные кроссоверы можно разделить на двухродительские и многородительские.

Двухродительские кроссоверы

Одноточечный кроссовер (*single-point* или *one-point crossover*) представляет собой классический бинарный кроссовер, использованный еще в работах Холланда. Схему одноточечного кроссовера можно представить в следующем виде.

1) Генерируем натуральное случайное число $i = U_1(1:|H|)$ и выбираем i-й разряд хромосомы в качестве точки скрещивания.

2) Копируем хромосомы H_1, H_2 в хромосомы H_1', H_2' соответственно.

3) Все биты хромосомы H_2, расположенные правее точки скрещивания, передаем хромосоме H_1', так что получаем

$$H_1' = (h_{1,1}, \ldots, h_{1,i-1}, h_{2,i}, \ldots, h_{2,|H|}).$$

4) Аналогично, все биты хромосомы H_1, расположенные правее точки скрещивания, передаем хромосоме H_2:

$$H_2' = (h_{2,1}, \ldots, h_{2,i-1}, h_{1,i}, \ldots, h_{1,|H|}).$$

Развитием одноточечного кроссовера можно считать так называемый *упорядоченный* (*order*) *кроссовер*. В этом случае после копирования левого фрагмента хромосомы H_1 в хромосому H_1' переупорядочивают тем или иным образом биты хромосомы H_2 и копируют в хромосому H_1' правый фрагмент

полученной хромосомы. При формировании хромосомы H_2' поступают аналогичным образом.

Многоточечный (multi-point) кроссовер является развитием одноточечного кроссовера. Частный случай многоточечного кроссовера – широко используемый *двухточечный кроссовер*.

Основные этапы функционирования *n*-точечного кроссовера выглядят следующим образом.

1) Генерируем несовпадающие натуральные случайные числа $i_j = U_1(1:|H|)$, $j \in [1:n]$ такие, что $1 < i_1 < i_2 < \ldots < i_n < |H|$; $n \in [1:|H|]$. Выбираем разряды хромосомы i_1, $i_2, \ldots, i_n$.

2) Копируем хромосомы H_1, H_2 в хромосомы H_1', H_2' соответственно.

3) Передаем хромосоме H_1' биты хромосомы H_2, разряды которых удовлетворяют условиям $i_1 \le i < i_2$, $i_3 \le i < i_4$ и т. д.:

$$H_1' = (h_{1,1}, \ldots h_{1,i_1-1}, h_{2,i_1}, \ldots, h_{2,i_2-1}, h_{1,i_2}, \ldots, h_{1,i_3-1}, h_{2,i_3}, \ldots, h_{2,i_4-1}, \ldots).$$

4) Аналогично передаем хромосоме H_2' биты хромосомы H_1 с номерами, удовлетворяющими тем же условиям:

$$H_2' = (h_{2,1}, \ldots, h_{2,i_1-1}, h_{1,i_1}, \ldots, h_{1,i_2-1}, h_{2,i_2}, \ldots, h_{2,i_3-1}, h_{1,i_3}, \ldots, h_{1,i_4-1}, \ldots).$$

Вариант многоточечного кроссовера можно получить последовательным повторением одноточечного кроссовера необходимое число раз.

Однородный (*uniform*) *кроссовер* представляет собой особый случай многоточечного кроссовера, широко используемого в вычислительной практике. В этом кроссовере каждый бит особи-родителя с равной вероятностью может быть помещен в соответствующий разряд первой или второй особи-потомка. Схему однородного кроссовера задает следующая последовательность шагов.

1) Генерируем вектор случайных чисел $U = U_{|H|}(0;1)$.

2) Поочередно для $i = 1, 2, \ldots, |H|$ анализируем компоненты вектора U. Если $u_i < 0,5$, то в *i*-й разряд хромосомы H_1' копируем бит, находящийся в том же разряде хромосомы H_1. В противном случае в *i*-й разряд хромосомы H_1' копируем бит, находящийся в том же разряде хромосомы H_2.

3) По рассмотренной схеме генерируем новый вектор U.

4) Если $u_i < 0,5$, $i \in [1:|H|]$, то в *i*-й разряд хромосомы H_2' копируем бит, находящийся в том же разряде хромосомы H_2, иначе в *i*-й разряд хромосомы H_2' копируем бит, находящийся в том же разряде хромосомы H_1.

Вариантом однородного кроссовера является так называемый *полуоднородный* (*half uniform*) *кроссовер*, в котором каждой особи-потомку переходит ровно половина битов каждой из особей-родителей.

Перетасовочный (*shuffler*) *кроссовер* реализует следующие операции. У хромосом H_1, H_2 случайным образом выбираем по одному разряду и обмениваем биты, находящиеся в этих разрядах (выполняем операцию тасовки). Обозначим полученные хромосомы $\tilde{H}_1$, $\tilde{H}_2$. Далее реализуем одноточечный кроссовер этих хромосом – получаем хромосомы H_1', H_2'. Хромосомы H_1', H_2' вновь подвергаем тасовке и объявляем полученные хромосомы H_1'', H_2'' результирующими.

Комбинированный кроссовер построен на основе классического одно- или двухточечного и равномерного кроссоверов. Суть оператора состоит в том, что один из этих кроссоверов применяют к каждому из выбранных генов H_i^g, $i \in [1:|X|]$ в отдельности. При этом необходимость скрещивания данного гена определяют заданной вероятностью ξ_a (свободный параметр алгоритма).

Операторы вставки и удаления используют вместе, как, по сути, один оператор. Рассмотрим схему простейшего варианта использования этих операторов.

1) Генерируем натуральные случайные числа $i_j = U_1(1:|H|)$, $j \in [1:4]$, подчиненные ограничениям $1 < i_1 < i_2 < i_3 < i_4 \le |H|$, $i_2 - i_1 = i_4 - i_3$ и имеющие смысл точек вставки и удаления.

2) Копируем хромосому H_2 в хромосому H'.

3) Все биты хромосомы H_1, расположенные правее разряда i_1 и левее разряда i_2, передаем хромосоме H', так что длина этой хромосомы становится равной $(|H| + (i_2 - i_1))$:

$$H' = (h_{2,1},...,h_{2,i_1},h_{1,i_1+1},...,h_{1,i_2},h_{2,i_1+1},...,h_{2,|H|}) = (h'_1,...,h'_{|H|+(i_2-i_1)}).$$

4) Из хромосомы H' удаляем биты, расположенные правее разряда i_3 и левее разряда i_4, так что ее длина становится равной $|H|$:

$$H' = (h'_1,...,h'_{i_3},h'_{i_4+1},...,h'_{|H|}).$$

Операторы вставки и удаления могут использовать и более двух родительских особей.

Многородительский кроссовер

Триадный (triadic) кроссовер использует три родительские хромосомы, одну из которых объявляют маской. Примерно 10 % битов маски инвертируют. Биты хромосомы H_1 сравнивают с битами маски. Если они совпадают, то их передают в соответствующие позиции хромосомы H'_1. В оставшиеся позиции этой хромосомы переходят соответствующие биты хромосомы H_2. Хромосома H'_2 формируется по аналогичной схеме: при совпадении битов хромосомы H_2 с битами маски первые из них переходят в соответствующие позиции хромосомы H'_2, а в оставшиеся позиции этой хромосомы помещают соответствующие биты хромосомы H_1.

В модификации данного кроссовера хромосому H'_1 получают сложением по модулю 2 соответствующих битов хромосомы H_1 и указанной маски, а хромосому H'_2 получают по аналогичной схеме на основе хромосомы H_2.

Маска может быть не выбрана из текущей популяции, а получена с помощью генератора случайных чисел, равномерно распределенных в интервале [0;1], по следующему правилу: если величина этого числа превышает заданную вероятность, то в соответствующий разряд хромосомы помещают едини-

цу, в противном случае – ноль. Могут быть использованы и более сложные правила формирования маски.

Оператор сегрегации. Схему оператора можно представить в следующем виде.

1) С помощью генератора натуральных случайных чисел, равномерно распределенных в интервале $[1:|S|]$, выбираем из текущей популяции n особей, которые для простоты записи обозначим s_1, s_2 ,..., s_n, где $n \in [2:|H|]$ – свободный параметр.

2) Генерируем натуральные случайные числа $i_j = U_1(1:|H|)$, $j \in [1:(n-1)]$, подчиненные ограничению $1 < i_1 < i_2 <,\ldots, < i_{n-1} \le |H|$ и имеющие смысл номеров точек скрещивания.

3) Все биты хромосомы H_1, расположенные левее разряда i_1, передаем хромосоме H':

$$H' = (h_{1,1},\ldots,h_{1,i_1-1},*,\ldots,*).$$

4) Аналогично, биты хромосомы H_2, расположенные правее разряда (i_1-1) и левее разряда i_2, передаем хромосоме H':

$$H' = (h_{1,1},\ldots,h_{1,i_1-1},h_{2,i_1},\ldots,h_{2,i_2-1},*,\ldots,*).$$

5) И так далее до последней хромосомы H_n:

$$H' = (h_{1,1},\ldots,h_{1,i_1-1},h_{2,i_1},\ldots,h_{2,i_2-1},h_{n,i_n-1},\ldots,h_{n,|H|}).$$

Рассмотренные кроссоверы (кроме однородного кроссовера) реализуют обмен битами между хромосомами-родителями на уровне хромосом в целом, то есть не учитывают фенотип особей. В результате при использовании этих кроссоверов происходит кардинальное изменение значений некоторого числа закодированных в хромосомах параметров (их разрушение). Указанное явление приводит к диверсификации поиска и на ранних его стадиях играет положительную роль. С другой стороны, это явление уменьшает интенсивность поиска и на поздних его стадиях может существенно замедлить сходимость алгоритма.

Для задач оптимизации высокой размерности данное явление вызывает еще следующий эффект. Поскольку в этом случае число параметров $|X|$ значительно превосходит число точек скрещивания, эволюционный алгоритм, по сути, переходит от поиска новых значений этих параметров к поиску комбинаций уже найденных их значений. В результате диверсификацию поиска приходится обеспечивать, в основном, с помощью оператора мутации.

На первый взгляд, возможным решением указанных проблем является использование однородного скрещивания. Однако в этом случае при переходе от текущей популяции к следующей могут быть разрушены абсолютно все параметры, и баланс между интенсификацией и диверсификацией поиска будет кардинально нарушен в пользу последней.

Таким образом, в ряде случае, особенно в задачах высокой размерности, целесообразно использовать кроссоверы, построенные на основе фенотипа особей.

2.4.2. Вещественные кроссоверы

Вещественные операторы скрещивания обеспечивают получение на основе двух родительских векторов вещественных чисел одного или большего числа векторов-потомков. Большинство вещественных кроссоверов порождают особей-потомков в некоторой окрестности родительских особей.

Выделим три класса вещественных кроссоверов:

– кроссоверы, которые на основе двух родительских особей продуцируют одну особь-потомка;

– аналогичные кроссоверы, производящие двух особей-потомков;

– кроссоверы, порождающие более двух особей-потомков.

Кроссоверы с одной особью-потомком

Плоский (*flat*) *кроссовер* ставит в соответствие хромосомам X_1, X_2 хромосому X', в которой ген x'_i, $i \in [1:|X|]$ представляет собой случайное число, равномерно распределенное в интервале $\left[x_i^{\min}; x_i^{\max}\right]$. Здесь и далее в этом пункте принято, что $x_i^{\min} = \min\left(x_{1,i}, x_{2,i}\right)$, $x_i^{\max} = \max\left(x_{1,i}, x_{2,i}\right)$.

Данное правило построения хромосомы-потомка иллюстрирует рис. 2.1.

Эвристический (*heuristic*) *кроссовер* представляет собой вариант арифметического кроссовера (см. ниже). В данном кроссовере на основе хромосом X_1, X_2 создают хромосому X'_1 такую, что

$$x'_i = u(x_{1,i} - x_{2,i}) + u\, x_{1,i}, \; i \in \left[1:|X|\right],$$

где принято, что приспособленность хромосомы X_1 выше приспособленности хромосомы X_2; $u = U_1(0;1)$.

Смешанный (*blend*) *кроссовер* генерирует хромосому X', в которой ген x'_i представляет собой случайное число, равномерно распределенное в интервале

$$\left[(x_i^{\min} - b\delta_i); (x_i^{\max} + b\delta_i)\right], \; i \in \left[1:|X|\right],$$

где $\delta_i = x_i^{\max} - x_i^{\min}$; b – свободный параметр кроссовера (рис. 2.2).

В случае, когда величина параметра b одна и та же для всех $|X|$ генов, смешанный кроссовер превращается в *равномерный смешанный кроссовер*, в котором векторы X'_1, X'_2 представляют собой линейные комбинации векторов X_1, X_2. Легко видеть, что при $b = 0$ смешанный кроссовер трансформируется в плоский кроссовер.

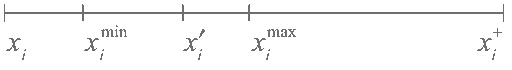


Рис. 2.1. К схеме плоского кроссовера

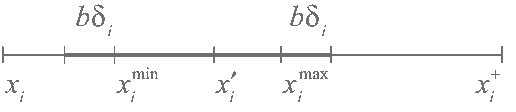


Рис. 2.2. К схеме смешанного кроссовера

Расширенный линейный (*extended line*) *кроссовер* на основе хромосом X_1, X_2 создает хромосому X' такую, что

$$x'_i = x_i^{\min} + u\delta_i,\ i \in \left[1:|X|\right],$$

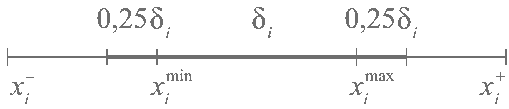


Рис. 2.3. К схеме расширенного линейного кроссовера

где случайное число $u = U_1(-0,25; 1,25)$.
Величина x'_i представляет собой при этом случайное число, равномерно распределенное в интервале $[(x_i^{\min} - 0,25\delta_i); (x_i^{\max} + 0,25\delta_i)]$ (рис. 2.3).

Дискретный (*discrete*) *кроссовер*. Схема кроссовера близка к схеме однородного бинарного кроссовера. В качестве значения гена x'_i хромосомы X' дискретный кроссовер с равной вероятностью использует значения гена $x_{1,i}$ либо гена $x_{2,i}$ родительских хромосом X_1, X_2 соответственно.

Кроссоверы с двумя особями-потомками

Простейший (*simple*) *кроссовер* генерирует две хромосомы-потомка по следующему правилу. Находим натуральное случайное число $i_1 = U_1\left(1:\left(|X|-1\right)\right)$. В качестве хромосом X'_1, X'_2 используем хромосомы

$$X'_1 = \left(x_{1,1}, \ldots, x_{1,i_1}, x_{2,i_1+1}, \ldots, x_{2,|X|}\right),$$

$$X'_2 = \left(x_{2,1}, \ldots, x_{2,i_1}, x_{1,i_1+1}, \ldots, x_{1,|X|}\right).$$

Арифметический (*arithmetical*) *кроссовер* создает хромосомы X'_1, X'_2 такие, что

$$x'_{1,i} = ux_{1,i} + (1-u)x_{2,i},$$

$$x'_{2,i} = ux_{2,i} + (1-u)x_{1,i}.$$

Здесь случайное число $u = U_1(0;1)$; $i \in \left[1:|X|\right]$. Схему функционирования арифметического кроссовера иллюстрирует рис. 2.4, на котором точки 1, 2, 3 соответствуют величине $x'_{1,i}$ при $u = 1/3$, $1/2$, $2/3$ соответственно.

В ситуации, когда величина параметра u одна и та же для всех $|X|$ генов хромосомы, арифметический кроссовер превращается в *равномерный арифметический*.

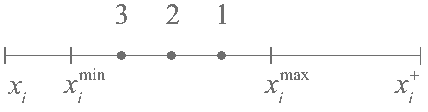


Рис. 2.4. К схеме арифметического кроссовера

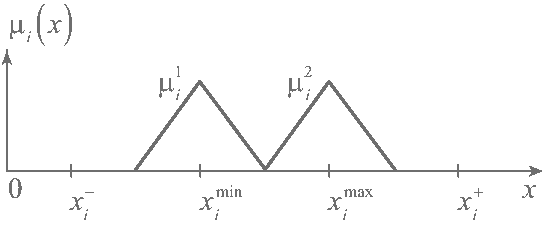


Рис. 2.5. К схеме нечеткого кроссовера: $b = 0,5$

Геометрический (*geometrical*) *кроссовер* продуцирует хромосомы X_1', X_2' по правилу

$$x_{1,i}' = x_{1,i}^{b}\, x_{2,i}^{1-b},$$

$$x_{2,i}' = x_{2,i}^{b}\, x_{1,i}^{1-b}.$$

Здесь b – свободный параметр; $i \in \left[1:\left|X\right|\right]$. В случае, когда $b = const$, имеем *равномерный геометрический кроссовер*.

Нечеткий (*fuzzy*) *кроссовер* при вычислении значений генов $x_{1,i}'$, $x_{2,i}'$, $i \in [1:|X|]$ хромосом X_1', X_2' использует функцию принадлежности $\mu_i(x) = (\mu_i^1(x), \mu_i^2(x))$ (рис. 2.5).

Формально функции $\mu_i^1(x)$, $\mu_i^2(x)$ определяем следующим образом:

– функция $\mu_i^1(x)$ на интервалах $[(x_i^{\min} - b\delta_i);\ x_i^{\min}]$, $[x_i^{\min};(x_i^{\min} + b\delta_i)]$ является линейно возрастающей и убывающей соответственно, а вне этих интервалов равна нулю;

– функция $\mu_i^2(x)$ аналогичным образом на интервалах $[(x_i^{\max} - b\delta_i);\ x_i^{\max}]$, $[x_i^{\max};(x_i^{\max} + b\delta_i)]$ является линейной и вне указанных интервалов равной нулю.

Свободный параметр кроссовера b задает степень перекрытия приведенных на рис. 2.5 треугольников. Рекомендуемое значение параметра равно 0,5 (именно такой случай иллюстрирует рис. 2.5).

Вероятность того, что ген $x_{1,i}'$, $i \in [1:|X|]$ хромосомы X_1' примет некоторое значение x, определяет функция $\mu_i^1(x)$, интерпретируемая как функция плотности вероятности. По аналогичному правилу определяем значение гена $x_{2,i}'$.

Кроссовер, имитирующий бинарный кроссовер (*Simulated Binary Crossover*, *SBX*), формирует хромосомы X_1', X_2' по правилу

$$x_{1,i}' = 0{,}5\left((1-u_1)x_{1,i} + (1+u_1)x_{2,i}\right),$$

$$x_{2,i}' = 0{,}5\left((1-u_1)x_{2,i} + (1+u_1)x_{1,i}\right),$$

где $i \in [1:|X|]$; u_1, u_2 – случайные величины, плотность вероятности которых подчинена закону

$$\xi(u) = \begin{cases} (2u)^{\frac{1}{b+1}}, & u \ge 0{,}5, \\ \left(\dfrac{1}{2(1-u)}\right)^{\frac{1}{b+1}}, & u < 0{,}5. \end{cases}$$

Здесь $b \in [2:5]$ – натуральное число (свободный параметр кроссовера), увеличение которого влечет за собой увеличение вероятности появления особи-потомка s_i' в окрестности особи-родителя s_i и наоборот; $i = 1,2$ (рис. 2.6).

SBX-кроссовер обладает следующими важными свойствами. Во-первых, этот кроссовер, аналогично бинарному одноточечному кроссоверу, обеспечи-

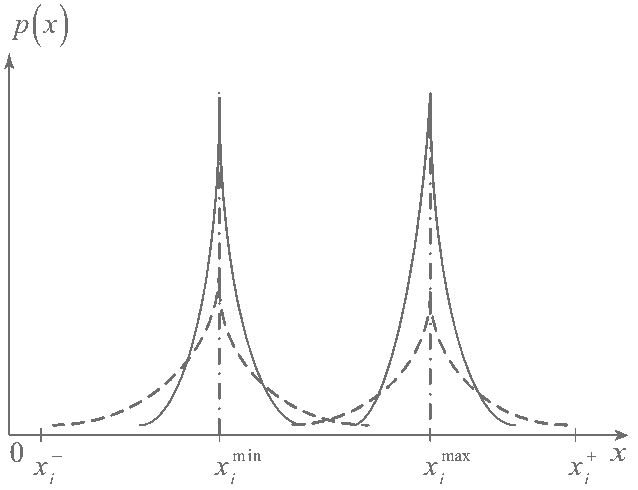


Рис. 2.6. К схеме *SBX*-кроссовера: $b_1 > b_2$;
—— b_1, - - - b_2

вает постоянство средней приспособленности особей-родителей и особей-потомков. Во-вторых, сила поиска (*search power*) *SBX*-кроссовера равна силе поиска указанного бинарного кроссовера. Под силой поиска понимают величину, характеризующую распределение вероятностей появления любой особи-потомка от двух произвольных особей-родителей.

Кроссоверы с тремя особями-потомками

Линейный (*linear*) *кроссовер* производит три хромосомы X_1', X_2', X_3' по правилу (рис. 2.7)

$$x_{1,i}' = 0,5\, x_{1,i} + 0,5\, x_{2,i},$$

$$x_{2,i}' = 1,5\, x_{1,i} - 0,5\, x_{2,i},$$

$$x_{3,i}' = -0,5\, x_{1,i} + 1,5\, x_{2,i},$$

где $i \in [1 : |X|]$.

Рассмотренные вещественные кроссоверы, исключая *SBX*-кроссовер, были предложены достаточно давно. Как показали исследования, во многих случаях они обеспечивают невысокую эффективность. Исключением является лишь смешанный кроссовер с параметром $b = 0,5$. *SBX*-кроссовер относится к числу наиболее известных улучшенных вещественных кроссоверов и широко используется в вычислительной практике.

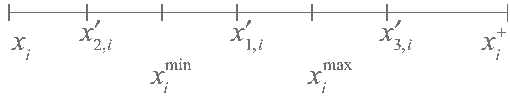


Рис. 2.7. К схеме линейного кроссовера

Исследования показали, что для уменьшения вероятности преждевременной сходимости эволюционного алгоритма эффективным может быть комбинирование нескольких различных операторов кроссовера. При этом используют специальные стратегии комбинирования, изменяющие вероятность применения того или иного кроссовера в зависимости от его эффективности. Кроме того, есть большое число гибридных кроссоверов, в которых входящие в них «элементарные» кроссоверы интегрированы настолько тесно, что образуют, по сути, новые кроссоверы.

2.5. ОПЕРАТОРЫ ОТБОРА

Выделяем два близких, но функционально несколько отличающихся класса операторов отбора:

– операторы управления популяцией;

– операторы селекции.

Поскольку, как отмечалось, операторы отбора (в отличие от операторов мутации и кроссовера) инвариантны к схеме кодирования особей, не различаем бинарное и вещественное кодирование.

2.5.1. Операторы управления популяцией

Разделим операторы управления популяцией на две следующие группы:

– операторы, учитывающие только приспособленность особей;

– операторы, учитывающие также близость генотипов особей.

Операторы на основе близости приспособленностей

Метод рулетки (*roulette*) или *метод пропорционального отбора* (*proportional sampling*) является простейшим методом отбора. В этом случае вероятность ξ_i отбора особи s_i текущей популяции определяет формула

$$\xi_i = \frac{\varphi(s_i)}{\sum_{j=1}^{|S|} \varphi(s_j)}, \; i \in [1:|S|]. \tag{2.2}$$

Схему отбора методом рулетки представляет следующая последовательность шагов.

1) Для каждой из особей s_i, $i \in [1:|S|]$ текущей популяции вычисляем ее приспособленность и по формуле (2.2) определяем вероятности отбора.

2) Генерируем случайное число $u = U_1(0; 2\pi)$.

3) Помещаем особь s_i, соответствующую углу u, в промежуточную популяцию S' (рис. 2.8). Если требуемое число $|S'|$ особей в этой популяции сформировано, то завершаем вычисления, иначе переходим к шагу 2.

На рис. 2.8 принято, что угол раскрыва сектора, соответствующего вероятности ξ_i, пропорционален этой вероятности.

Из схемы метода рулетки следует, что чем больше величина ξ_i, тем шире соответствующий сектор виртуальной рулетки, и тем с большей вероятно-

стью будет отобрана особь s_i. В программной реализации метода, очевидно, отнюдь не обязательно моделировать рулетку. Достаточно интервал $[0;1]$ разделить на $|S|$ подынтервалов, длины которых пропорциональны вероятностям ξ_i, $i \in [1:|S|]$, и случайное число u генерировать в том же интервале $[0;1]$.

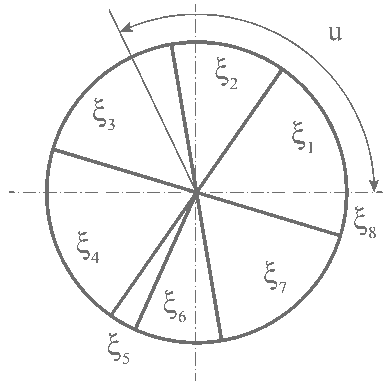


Рис. 2.8. К схеме метода рулетки: $|S| = 8$; отобрана особь S_3

Недостаток метода рулетки заключается в том, что особи с малой приспособленностью слишком быстро исключаются из популяции, что может привести к преждевременной стагнации эволюционного алгоритма. Кроме того, в методе рулетки самые приспособленные особи не всегда переходят в следующее поколение. В результате новая популяция может «потерять» особей предыдущей популяции с наибольшим значением фитнес-функции.

Развитием метода рулетки является *метод пропорционального отбора с остатком* (*remainder proportional*). Метод основан на вычислении отношения $\frac{\varphi(s_i)}{\overline{\varphi}(S)}$, $i \in [1:|S|]$ приспособленности особи s_i текущей популяции S к средней приспособленности особей этой популяции

$$\overline{\varphi}(S) = \frac{\sum_{j=1}^{|S|} \varphi(s_j)}{|S|}. \qquad (2.3)$$

Целая часть отношения $\frac{\varphi(s_i)}{\overline{\varphi}(S)}$ указывает, сколько раз нужно записать особь s_i в промежуточную популяцию, а дробная показывает ее вероятность попасть туда еще раз.

Еще одним вариантом пропорционального отбора является *метод стохастического универсального* (*stochastic universal*) *отбора*. В этом варианте выбор особи также производится пропорционально ее приспособленности, но таким образом, что особи с самой высокой приспособленностью будут гарантированно выбраны хотя бы один раз (а могут быть выбраны и более одного раза). Алгоритм стохастического универсального отбора интенсифицирует поиск, но за счет своей низкой вариабельности (*variance sampling*) уменьшает его широту.

Метод турнирного (*tournament*) *отбора* широко используют в вычислительной практике как более эффективный, чем метод рулетки. Метод предполагает случайное формирование на основе текущей популяции некоторого числа групп из n особей в каждой. В каждой из групп выбираем особь с наилучшей приспособленностью (турнир) и включаем ее в промежуточную популяцию. Обычно группы содержат по две-три особи ($n = 2$ либо $n = 3$). Размер групп может меняться в процессе функционирования эволюционного алгоритма.

Величина n называется размером турнира (*tournament size*) и определяет так называемое *давление селекции* (*selection pressure*). В простейшем случае, когда $n = 1$, метод турнирного отбора обеспечивает случайный отбор. В другом крайнем случае, когда n значительно превышает размер популяции $|S|$, вероятность победы в турнире наиболее приспособленной особи приближается к единице.

Для простоты записи примем, что число особей $|S|$ в текущей популяции кратно величине n, и положим, что число особей в промежуточной популяции $|S'|$ равно $|S|$. Схема турнирного отбора в этом случае имеет следующий вид.

1) Первый проход.

– Тем или иным способом всех особей s_i, $i \in [1:|S|]$ текущей популяции разбиваем на $\frac{|S|}{n}$ групп по n особей в каждой.

– В каждой из групп для каждой особи вычисляем ее приспособленность и выбираем наиболее приспособленную особь. Включаем выбранные $\frac{|S|}{n}$ особей в промежуточную популяцию.

2) Второй проход. Случайным образом перемешиваем исходную популяцию и тем же образом формируем вторые $\frac{|S|}{n}$ особей промежуточной популяции.

…

n) n-й проход. По рассмотренной схеме формируем последние $\frac{|S|}{n}$ особей промежуточной популяции.

Метод рангового (*rank-based*) *отбора*. Суть рангового отбора заключается в том, что число копий данной особи, включенных в промежуточную популяцию, является некоторой априори заданной функцией ранга этой особи. Рангом особи называют номер этой особи в списке, отсортированном по возрастанию приспособленности особей. Схему рангового отбора можно представить в следующем виде.

1) Вычисляем приспособленности всех особей текущей популяции:

$$\varphi(s_i) = \varphi_i, \; i \in [1:|S|].$$

2) Сортируем величины φ_i в порядке их убывания:

$$\varphi_{i_1}, \; \varphi_{i_2}, \ldots, \; \varphi_{i_{|S|}}, \; i_j \in [1:|S|].$$

3) Присваиваем особи с номером i_j ранг, равный j; $j \in [1:|S|]$.

4) Ставим в соответствие рангу j вероятность $\xi_j = \xi(j)$, имеющую смысл вероятности включения особи s_{i_j} в промежуточную популяцию; $\xi(j)$ – некоторая убывающая функция своего аргумента.

5) На основе вероятностей ξ_j с помощью метода рулетки выбираем из исходной популяции требуемое число особей $|S'|$ и включаем их в промежуточную популяцию.

В качестве функции $\xi(j)$ используют простые функции, например линейную функцию вида

$$\xi(j) = aj + b\,,\ a < 0$$

либо убывающую экспоненциальную функцию вида

$$\xi(j) = a \cdot \exp\left(bj + c\right),\ b < 0.$$

Здесь a, b, c – свободные параметры.

Достоинством рангового отбора является исключение ранней стагнации эволюционного алгоритма, поскольку метод гарантирует разнообразие популяции на всех итерациях алгоритма.

Метод отбора на основе элитизма (*elitism-based*) основан на предварительном выборе лучших, то есть наиболее приспособленных особей. Схема простейшего метода отбора на основе элитизма представлена следующей последовательностью шагов.

1) Вычисляем приспособленности всех особей текущей популяции φ_i, $i \in [1:|S|]$ и сортируем величины φ_i в порядке их убывания:

$$\varphi_{i_1}, \varphi_{i_2}, \ldots, \varphi_{i_{|S|}},\ \ i_j \in [1:|S|].$$

2) Отбираем в промежуточную популяцию особи s_{i_j}, соответствующие первым $|S'| < |S|$ членам этой последовательности.

Важным достоинством метода элитизма является то, что он гарантирует сохранение в популяции одной или нескольких наиболее приспособленных особей. Чаще всего используют вариант метода, предполагающий обязательное сохранение только одной лучшей особи.

Известно множество модификаций рассмотренного простейшего метода. Так, в промежуточную популяцию по рассмотренной схеме может быть включено, к примеру, лишь $0{,}1|S'|$ особей. Остальные $0{,}9|S'|$ особей выбирают с помощью одного или нескольких других методов отбора. В том числе эти особи могут быть получены случайной генерацией новых особей (как при инициализации эволюционного алгоритма).

Модификацией метода отбора на основе элитизма можно считать *метод отбора отсечением* (*truncation*). Метод требует задания величины $\hat{\varphi}$ – порога приспособленности. Особи текущей популяции упорядочиваем по убыванию приспособленности, как это сделано выше. Среди особей, приспособленность которых больше $\hat{\varphi}$, случайным образом выбираем особь и помещаем в промежуточную популяцию. Процесс повторяем требуемое число раз. Заметим, что при этом некоторые особи могут попасть в промежуточную популяцию несколько раз, а некоторые особи, приспособленность которых больше величины $\hat{\varphi}$, – не попасть ни разу.

Операторы на основе близости приспособленностей и генотипов

Метод отбора вытеснением (*crowding*) обеспечивает разнообразие популяции путем удаления из нее в некотором смысле близких особей.

Обозначим $\rho(s_1, s_2)$ меру близости особей s_1, s_2 рассматриваемой популяции. В случае бинарного кодирования особей в качестве этой меры может быть использована *метрика Хемминга* $\|H_1, H_2\|_H$ – число различающихся битов в одноименных разрядах соответствующих хромосом. При вещественном кодировании особей можно использовать различные метрики пространства $R^{|X|}$, например евклидову норму $\rho_E(s_1, s_2) = \|X_1, X_2\|_E$.

Схема метода отбора вытеснением имеет следующий вид.

1) Для популяции $S = \{s_i, i \in [1:|S|]\}$ формируем треугольную матрицу $R = \left(\rho_{i,j}, i \in [1:|S|-1], j \in [i+1:|S|]\right)$ расстояний между особями популяции, где $\rho_{i,j} = \rho(s_i, s_j)$.

2) Выбираем элемент ρ_{i_1,j_1} матрицы R, имеющий минимальное значение, и исключаем из дальнейшего рассмотрения с равной вероятностью особь s_{i_1} или особь s_{j_1}.

3) Аналогично выбираем из числа оставшихся в матрице элементов элемент ρ_{i_2,j_2} и оставляем особь s_{i_2} либо особь s_{j_2}.

4) И так далее до тех пор, пока не останется требуемое число особей.

К числу операторов отбора рассматриваемого класса можно отнести также оператор турнирного отбора, когда группы особей формируют не случайным образом, а в зависимости от их близости в пространстве генотипов.

Кроме рассмотренного двухэтапного метода формирования популяции нового поколения по схеме

$$S(t) \rightarrow S'(t) \rightarrow S(t+1)$$

возможен также трехэтапный метод

$$S(t) \rightarrow S'(t) \rightarrow S''(t) \rightarrow S(t+1),$$

где $S''(t)$ – вспомогательная популяция (популяция кандидатов), включающая в себя всю популяцию $S'(t)$, а также особи этой популяции, подвергшиеся мутации. На каждом из указанных этапов могут быть использованы рассмотренные методы отбора. Таким образом, легко сконструировать большое число комбинированных методов отбора, позволяющих достичь требуемого соотношения между интенсивностью и широтой поиска эволюционного алгоритма.

2.5.2. Операторы селекции

Операторы селекции реализуют выбор из промежуточной популяции особей, подлежащих скрещиванию. Как правило, селекцию выполняют, оценивая приспособленность особей. Основным является следующий принцип селективного отбора: вероятность отбора наименее приспособленных особей с ростом номера поколения должна убывать или, что то же самое, вероятность отбора наиболее приспособленных особей должна увеличиваться.

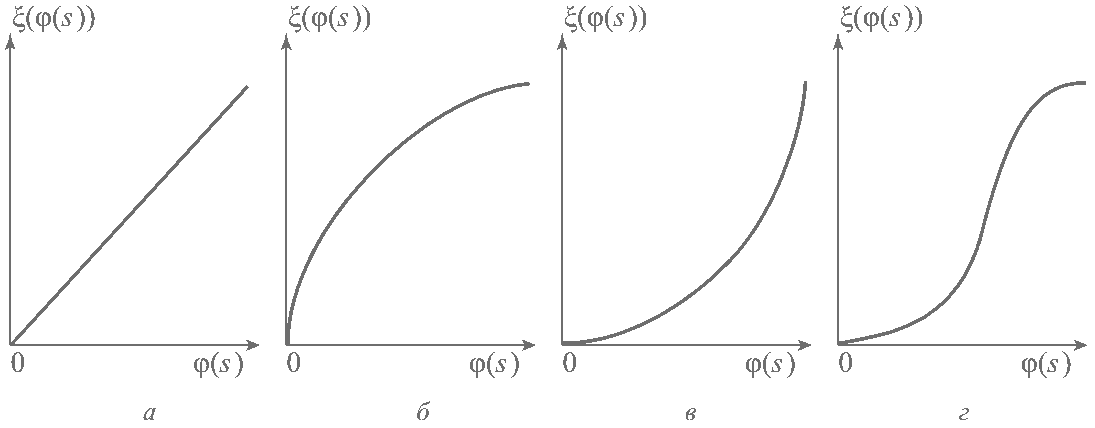


Рис. 2.9. Классы функций отбора: *а* – $\xi''(\varphi)=0$; *б* – $\xi''(\varphi)<0$; *в* – $\xi''(\varphi)>0$; *г* – $\xi''(\varphi)>0, \xi''(\varphi)<0$

Селекцию на основе приспособленности особей можно интерпретировать как отображение

$$\varphi(s) \to \xi(\varphi(s)),$$

где $\xi(\varphi(s))$ – *функция отбора*, имеющая смысл вероятности выбора особи s, приспособленность которой равна $\varphi(s)$. Основные классы функции отбора иллюстрирует рис. 2.9.

Функция отбора, представленная на рис. 2.9, *а*, соответствует пропорциональному отбору.

В случае, представленном на рис. 2.9, *б*, особи со средней приспособленностью имеют практически равную вероятность быть отобранными по сравнению с наиболее приспособленными особями. Операторы селекции, использующие функцию отбора такого рода, замедляют улучшение средней приспособленности популяции в пользу более полного исследования пространства поиска, то есть баланс между интенсивностью и широтой поиска сдвигают в сторону последней.

В ситуации, представленной на рис. 2.9, *в*, особи со средней приспособленностью имеют значительно меньшую вероятность отбора по сравнению с наиболее приспособленными особями. Селекция на основе такой функции сдвигает баланс между интенсивностью и широтой поиска в сторону интенсивности, то есть ускоряет улучшение средней приспособленности популяции за счет менее полного исследования пространства поиска.

Сигмоидальный характер функции отбора (рис. 2.9, *г*) обеспечивает в соответствующих операторах селекции сочетание преимуществ двух предыдущих вариантов построения этих операторов.

Обычно для скрещивания используют пару особей. Для выбора из промежуточной популяции рекомбинируемых особей (родительской пары) могут быть, очевидно, использованы рассмотренные выше методы отбора, если ограничиться отбором только двух особей. Существует также значительное число специальных методов селекции.

Разделим методы селекции на три следующих класса:
– случайные методы, не учитывающие приспособленность и генотип особей;
– методы, использующие только приспособленность особей;
– методы, основанные на генотипе особей.

Случайные методы

Метод панмиксии предполагает случайный равновероятный выбор родительской пары из промежуточной популяции. Особенностью метода является то, что в этом случае любая особь популяции может стать членом нескольких пар. Метод очень прост в реализации и в то же время достаточно эффективен для широкого класса задач оптимизации. Известным недостатком метода является его критичность к размеру популяции – с ростом численности популяции эффективность соответствующего эволюционного алгоритма снижается.

Для простоты записи будем далее опускать указание на то, что селекция производится из промежуточной популяции (тем более что, как отмечалось выше, промежуточная популяция может совпадать с текущей). Схема метода панмиксии представлена ниже.

1) Ставим в соответствие каждой особи s_i популяции вероятность $\xi_i = \frac{1}{|S|}$; $i \in [1:|S|]$.

2) Методом рулетки выбираем из популяции S первую особь s_{i_1} родительской пары.

3) Аналогичным образом выбираем вторую особь s_{i_2} родительской пары.

Методы на основе приспособленности особей

Метод селективного отбора особей в родительскую пару предполагает, во-первых, что родителями становятся только те особи, приспособленность которых не меньше среднего значения приспособленности по популяции. Во-вторых, полагают, что все особи, удовлетворяющие указанному условию, могут образовать пару с равной вероятностью. Схема метода селективного выбора имеет следующий вид.

1) Для каждой особи s_i популяции S вычисляем значение ее приспособленности $\varphi(s_i) = \varphi_i$; $i \in [1:|S|]$.

2) Определяем величину $\overline{\varphi}(S) = \overline{\varphi}$ – среднюю приспособленность особей популяции (см. формулу (2.3)).

3) Выбираем из популяции S особей s_{i_j}, $i_j \in [1:|S|]$, $j \in [1:n]$, приспособленность которых удовлетворяет условию $\varphi(s_{i_j}) = \varphi_{i_j} \geq \overline{\varphi}$. Здесь $n < |S|$ – число таких особей.

4) Ставим в соответствие каждой особи s_{i_j} вероятность $\xi_{i_j} = \frac{1}{n}$.

5) Методом рулетки выбираем из числа особей s_{i_j} первую особь родительской пары. Аналогичным образом выбираем вторую особь родительской пары.

Известно, что метод селективного отбора обеспечивает высокую скорость сходимости эволюционного алгоритма (высокую интенсивность поиска). Од-

нако в случае многоэкстремальной целевой функции это может привести к сходимости алгоритма к некоторому локальному минимуму этой функции (эффект уменьшения широты поиска).

Методы на основе генотипа

Инбридинг (*inbreeding*) представляет собой метод селекции, в котором первую особь родительской пары выбирают из популяции $S = (s_i\ ,\ i \in [1:|S|]\ )$ случайно, а в качестве второй особи выбирают с большей вероятностью особь той же популяции, генотип которой в некотором смысле наиболее близок к генотипу первой особи. В случае бинарного кодирования особей в качестве меры близости особей чаще всего используют метрику Хемминга $\|\cdot\|_H$. Для вещественных особей в качестве этой меры могут быть использованы различные нормы пространства $R^{|X|}$, например, евклидова норма.

Метод инбридинга определяет следующая последовательность шагов.

1) По схеме метода панмиксии выбираем из популяции S первую особь s_{i_1} родительской пары; $i_1 \in [1:|S|]$.

2) Вычисляем расстояние $\rho\left(s_{i_1}, s_i\right)$ от особи s_{i_1} до всех особей популяции s_i; $i \in [1:|S|]$, $i \neq i_1$.

3) Ставим в соответствие каждой из особей s_i вероятность ξ_i, пропорциональную величине $\rho\left(s_{i_1}, s_i\right)$.

4) По схеме метода рулетки на основе вероятностей ξ_i выбираем из числа особей s_i, $i \in [1:|S|]$, $i \neq i_1$ особь s_{i_2} как вторую особь искомой родительской пары.

Метод инбридинга часто приводит к разбиению популяции на отдельные группы особей, концентрирующихся вблизи точек локального экстремума целевой функции $f(X)$ (субоптимальных решений). Метод обычно позволяет быстро найти, по крайней мере, квазиоптимальное решение, т. е. обеспечивает высокую интенсивность поиска.

Аутбридинг (*outbreeding*) предполагает выбор первой особи родительской пары из популяции S по схеме панмиксии. В качестве второй особи с большей вероятностью выбирают особь той же популяции, которая в смысле используемой меры близости генотипов наиболее далека от первой особи. Схема метода аутбридинга совпадает со схемой метода инбридинга с тем отличием, что каждой из особей s_i ставим в соответствие вероятность ξ_i, обратно пропорциональную величине $\rho\left(s_{i_1}, s_i\right)$; $i \in [1:|S|]$, $i \neq i_1$.

Аутбридинг предупреждает раннюю сходимость генетического алгоритма, обеспечивая исследование новых областей пространства поиска, или, другими словами,– высокую диверсификацию поиска.

Методы инбридинга и аутбридинга можно модифицировать, используя в качестве меры близости особей какую-либо меру близости их приспособленностей, например, величину $|\varphi(s_1) - \varphi(s_2)|$.

Легко предложить различные варианты комбинации рассмотренных и других методов селекции, создавая тем самым еще одну степень свободы при конструировании эволюционных алгоритмов.

2.6. ДРУГИЕ ОПЕРАТОРЫ И ПРОЦЕДУРЫ

Рассматриваются комбинированные генетические операторы, а также процедуры масштабирования фитнес-функции, нишевания и учета ограничений на варьируемые параметры.

Комбинированные операторы

Известны генетические операторы, включающие в себя одновременно свойства и операторов мутации, и операторов скрещивания. Примером такого оператора является оператор транслокации.

Бинарный оператор транслокации комбинирует одноточечный оператор кроссовера и зеркалирующий мутатор по следующей схеме.

1) Генерируем натуральное случайное число $i_1 = U_1(1:|H|)$, имеющее смысл точки скрещивания.

2) Все биты хромосомы H_1, расположенные левее разряда i_1, передаем хромосоме H_1', а оставшийся фрагмент хромосомы зеркально отражаем и передаем хромосоме H_2':

$$H_1' = (h_{1,1},...,h_{1,i_1-1},*,...,*);$$

$$H_2' = (*,...,*,h_{1,|H|},...,h_{1,i_1}).$$

3) Аналогично биты хромосомы H_2, расположенные левее точки скрещивания, передаем хромосоме H_2', а оставшийся фрагмент H_2 зеркально отражаем и передаем хромосоме H_1':

$$H_1' = (h_{1,1},...,h_{1,i_1-1},h_{2,|H|},...,h_{2,i_1});$$

$$H_2' = (h_{2,1},...,h_{2,i_1-1},h_{1,|H|},...,h_{1,i_1}).$$

Масштабирование фитнес-функции

Масштабирование фитнес-функции может быть необходимым, во-первых, для предотвращения преждевременной сходимости эволюционного алгоритма, которая проявляется в том, что в популяции начинают доминировать наилучшие, но еще не оптимальные особи. Во-вторых, на конечных итерациях эволюционного алгоритма возможна ситуация, когда в популяции еще имеется значительная неоднородность, но среднее значение приспособленности особей популяции лишь немного отличается от минимального значения приспособленности. При этом средние и наилучшие особи (с точки зрения их приспособленности) могут производить практически одинаковое число потомков, что уменьшает скорость сходимости алгоритма.

Различают три основных метода масштабирования:

– линейное масштабирование;
– метод сигма-отсечения;
– степенное масштабирование.

Линейное масштабирование (*linear scaling*) заключается в переходе от исходной фитнес-функции $\varphi(X)$ к масштабированной фитнес-функции $\varphi_s(X)$ с помощью линейного преобразования вида

$$\varphi_s(X) = a\,\varphi(X) + b,$$

где a, b – заданные константы (свободные параметры преобразования). Значения параметров a, b выбирают такими, чтобы средние значения функций $\varphi(X)$, $\varphi_s(X)$ были равны и минимальное значение функции $\varphi_s(X)$ было в 1,2–2,0 раза меньше ее среднего значения по популяции.

Метод сигма-отсечения (*sigma truncation*) использует масштабированную фитнес-функцию

$$\varphi_s(X) = \varphi(X) + \left(\overline{\varphi} - b\sigma\right),$$

где $\overline{\varphi}$ – среднее значение приспособленности по популяции, b – натуральное число (свободный параметр метода), σ – оценка стандартного отклонения приспособленности по популяции. Рекомендуемые значения параметра b принадлежат интервалу $[1,0:5,0]$.

Степенное (*power law*) *масштабирование* предполагает, что итоговая фитнес-функция

$$\varphi_s(X) = (\varphi(X))^b,$$

где b – подбираемое эмпирически вещественное число, близкое к единице (свободный параметр преобразования).

Нишевание

В некоторых задачах многоэкстремальной оптимизации требуется найти как глобальный, так и все или некоторые локальные экстремумы. В то же время рассмотренные генетические операторы ориентированы на поиск только глобального экстремума, и если таковой не достигается, находят один из локальных экстремумов. Для построения эволюционного алгоритма, способного решать указанную задачу, используют различные процедуры диверсификации поиска путем образования ниш (*niches*). Наиболее известны процедуры, реализующие:

– метод разделяющей (сепарирующей) функции (*sharing function*),
– метод детерминированного сжатия (*deterministic crowding*),
– метод последовательного нишевания (*sequential niching*).

Рассмотрим в качестве примера первую из указанных процедур.

Метод разделяющей функции использует так называемую *разделяющую функцию* $\phi(\rho)$, где $\rho = \rho(s_1, s_2)$ – некоторая мера расстояния между особями s_1, s_2. В случае бинарного кодирования в качестве этой меры может быть использовано хеммингово расстояние $\|s_1, s_2\|_H$, а в случае вещественного кодирования – например, евклидова норма $\|s_1, s_2\|_E$.

Разделяющая функция $\phi(\rho)$ должна удовлетворять следующим требованиям:

$$\phi(\rho(s_1, s_2)) \in [0;1] \text{ для любых } s_1, s_2 \in S;$$

$$\phi(0) = 1, \text{ так что } \phi(\rho(s, s)) = 1;$$

$$\lim_{\rho(s_1, s_2) \to \infty} \phi(\rho(s_1, s_2)) = 0 \text{ для любых } s_1, s_2 \in S.$$

В качестве одной из функций, удовлетворяющих этим требованиям, используют функцию вида

$$\phi(\rho) = \begin{cases} \left(1 - \rho/d\right)^b, & \rho < d, \\ 0, & \rho \geq d, \end{cases} \tag{2.4}$$

где b, d – заданные положительные константы (свободные параметры функции). Константу d интерпретируют как размер ниши. Известна рекомендация, в соответствии с которой в качестве этой константы следует использовать величину

$$d = 0{,}5\, m^{-\frac{1}{|X|}}.$$

Здесь m – натуральное число, имеющее смысл примерного числа максимумов фитнес-функции (если речь идет о ее максимизации).

Модифицированную функцию полезности строят на основе разделяющей функции (2.4) в виде

$$\varphi_s(s_i) = \frac{\varphi(s_i)}{\sum_{j=1}^{|S|} \phi(\rho(s_i, s_j))}, \; i \in [1 : |S|]. \tag{2.5}$$

Из формул (2.4), (2.5) вытекает, что если каждая из особей популяции находится в своей нише (когда $\phi(\rho(s_i, s_j)) = 0$ для всех особей s_j, исключая s_i, и $\phi(\rho(s_i, s_i)) = 1$), то $\varphi_s(s_i) = \varphi(s_i)$. В противном случае значение фитнес-функции уменьшается пропорционально числу и степени близости соседствующих особей.

Использование эволюционного алгоритма с фитнес-функцией $\varphi_s(s_i)$ приводит к тому, что оказывается не выгодным увеличение числа особей, принадлежащих одной нише, поскольку это приводит к увеличению функции $\varphi_s(s_i)$, и особи вынуждены распределяться по разным нишам.

Отметим, что методы образования ниш могут быть использованы не только с целью отыскания локальных экстремумов в многоэкстремальной задаче оптимизации, но и для диверсификации поиска, то есть повышения вероятности локализации глобального экстремума в указанной задаче.

Альтернативные нишеванию методы обеспечения разнообразия популяции рассматриваются в 8.5.

Учет ограничений

Операторы мутации и кроссовера могут порождать недопустимых особей-потомков, то есть особей, не принадлежащих области допустимых значе-

ний D. Для решения этой проблемы используют варианты указанных операторов, которые продуцируют только допустимых особей; специальные методы, разработанные для эволюционных алгоритмов; рассмотренные в 1.3 методы решения задач условной оптимизации, такие как метод штрафных функций и метод скользящего допуска.

Для области D в виде гиперпараллелепипеда Π примерами операторов, которые порождают только допустимых особей, служат оператор случайной мутации, а также плоский и арифметический кроссоверы.

Из числа специальных методов, разработанных для решения задач условной оптимизации с помощью эволюционных алгоритмов, выделяют:

– методы специальных штрафных функций;

– методы, основанные на редукции недопустимых особей к допустимым;

– гибридные методы.

Методы специальных штрафных функций. К методам данной группы относим методы «смертельных» штрафов, статических штрафов и динамических штрафов. Этой же группе принадлежит метод, используемый в так называемом сегрегированном генетическом алгоритме.

Метод «смертельных» штрафов основан на простом отбрасывании недопустимых особей. Схема метода включает в себя следующие шаги.

1) Выполнение данного генетического оператора (мутации либо кроссовера) – получение особи-потомка s'.

2) Проверка особи s' на допустимость, т. е. проверка удовлетворения с заданной точностью ограничений

$$e_i(X') = 0\,,\ i \in [1:|E|]\,,$$
$$g_j(X') \geq 0\,,\ j \in [1:|G|]\,.$$

3) Завершение выполнения оператора, если особь s' успешно прошла проверку на допустимость. Отбрасывание этой особи и переход к шагу 1 в противном случае.

В случае, когда множество допустимых значений вектора варьируемых параметров представляет собой гиперпараллелепипед Π, указанная проверка сводится, очевидно, к проверке на допустимость каждого из генов особи s' в отдельности по правилу $x_j^- \leq x'_j \leq x_j^+,\ j \in [1:|X|]$. Если при этом речь идет об операторе мутации, то достаточно отбросить и повторно сгенерировать только недопустимый ген или гены.

Таким образом, метод «смертельных» штрафов, вообще говоря, приводит к необходимости многократного выполнения рассматриваемого генетического оператора применительно к родительским особям в целом или, в лучшем случае, применительно к отдельным их генам.

Метод статических штрафов Моралеса (*K. A. Morales*) и Кесады (*C. C. Quezada*) использует функцию штрафа вида

$$p(X) = b - \sum_{i=1}^{n} \frac{b}{|E| + |G|}\,, \tag{2.6}$$

где $n \in \left[0:|E|+|G|\right]$ – число нарушенных ограничений; b – большое положительное число (свободный параметр метода). Значения модифицированной фитнес-функции $\varphi_p(X)$ вычисляют при этом по формуле, несколько отличной от формулы (1.22):

$$\varphi_p(X) = \begin{cases} \varphi(X), & X \in D, \\ p(X), & X \notin D. \end{cases}$$

Величину b в формуле (2.6) следует выбирать такой, чтобы значения функции $\varphi_p(X)$ были для недопустимых особей значительно меньше значений этой функции для допустимых особей. Отметим, что метод не учитывает то, насколько сильно нарушены ограничения, а учитывает лишь общее число нарушенных ограничений.

Метод динамических штрафов Джойнеса (*J. Joines*) и Хука (*C. Houck*) использует функцию штрафа, близкую к функции, приведенной в 1.3.1:

$$p(X) = -\left(a\,t\right)^b \left(\sum_{i=1}^{|E|} P_e^c(e_i(X)) + \sum_{j=1}^{|G|} P_g^c(g_j(X)) \right). \qquad (2.7)$$

Здесь a, b, c – свободные параметры метода, а функционалы $P_e(e_i(X))$, $P_g(g_j(X))$ имеют вид

$$P_e(e_i(X)) = \begin{cases} 0, & e_i(X) = 0, \\ |e_i(X)|, & e_i(X) \neq 0, \end{cases}$$

$$P_g(g_j(X)) = \begin{cases} 0, & g_j(X) \geq 0, \\ |g_j(X)|, & g_j(X) < 0. \end{cases}$$

Важная особенность метода динамических штрафов состоит в том, что в нем абсолютная величина штрафа увеличивается с ростом числа поколений t. Рекомендуемые значения параметров a, b, c в формуле (2.7) равны (–0,5); 2,0; 2,0 соответственно.

Сегрегированный генетический алгоритм Ле Риша (*L. R. Riche*) преследует цель избежать как слишком малых, так и слишком больших значений штрафов и использует две функции штрафа $p_1(X)$, $p_2(X)$. Итерация сегрегированного алгоритма включает в себя следующую последовательность шагов.

1) Для каждой особи s_i, $i \in [1:|S|]$ текущей популяции $S(t)$ вычисляем значения обеих штрафных функций $p_1(X_i)$, $p_2(X_i)$.

2) Каждую из последовательностей величин $p_1(X_i)$, $p_2(X_i)$, $i \in [1:|S|]$ сортируем в порядке возрастания их значений.

3) Объединяем ранжированные списки $\{p_1(X_i)\}$, $\{p_2(X_i)\}$ в один такой же список длиной $2|S|$ и удаляем из этого списка последние $|S|$ членов, так что длина оставшейся части списка становится равной $|S|$ и в нем оказываются члены, соответствующие особям, в наименьшей степени нарушившим ограничения.

4) Применяем к полученной популяции $S' = (s'_1, ..., s'_{|S|})$ генетические операторы мутации, кроссовера и отбора так, чтобы получить новую популяцию $S(t+1)$.

Возможность использования данного метода ограничивает отсутствие четких рекомендаций по выбору штрафных функций $p_1(X)$, $p_2(X)$. Отметим, что сегрегированный генетический алгоритм содержит в себе некоторые черты многопопуляционных алгоритмов (гл. 6).

Методы редукции недопустимых особей к допустимым. Относим к этим методам процедуру «лечения» локальным поиском и метод редукции Орвоша – Дэвиса.

Процедура «лечения» локальным поиском ориентирована на бинарные генетические алгоритмы. В соответствии с этой процедурой при недопустимом текущем положении X_i агента s_i случайным образом выбираем соседнего допустимого агента s_j. Последовательно каждый бит хромосомы H_i сравниваем с соответствующим битом хромосомы H_j. В случае несовпадения этих битов инвертируем бит первой хромосомы. Процесс продолжаем до тех пор, пока модифицированная хромосома H_i не станет допустимой.

Метод редукции Орвоша (*D. Orvosh*) и Дэвиса (*L. Davis*) основан на использовании редуцирующей функции $r(X)$, которая способна трансформировать недопустимую особь $\tilde{s}$ в допустимую особь s'. Назовем эту особь s' *восстановленной*.

Можно предложить значительное число редуцирующих функций. Рассмотрим для примера оператор равномерной генной мутации. Положим, что в результате применения этого оператора к допустимой особи-родителю s мутировали гены h_{i_j}, $i_j \in [1:|X|]$, $j \in [1:n]$, так что хромосома-потомок $\tilde{H}$ является недопустимой и имеет вид

$$\tilde{H} = (h_1, ..., h_{i_1-1}, \overline{h}_{i_1}, h_{i_1+1}, ..., h_{i_2-1}, \overline{h}_{i_2}, h_{i_2+1}, ..., h_{i_{n-1}}, \overline{h}_{i_n}, h_{i_{n-1}}, ..., h_{|X|}),$$

где надчеркивание гена – символ его мутации. Тогда редуцирующую функцию можно определить следующей последовательностью шагов.

1) Последовательно для $j = 1, 2, ..., n$ возвращаем гену $\overline{h}_{i_j}$ его прежнее значение h_{i_j} и проверяем полученную особь на допустимость. Если получена допустимая особь, прекращаем вычисления, в противном случае переходим к следующему шагу.

2) Последовательно возвращаем паре генов $\overline{h}_{i_k}$, $\overline{h}_{i_l}$, случайным образом выбранной из совокупности генов h_{i_j}, $i_j \in [1:|X|]$, $j \in [1:n]$ их прежние значения h_{i_k}, h_{i_l} и подобно шагу 1 проверяем полученную особь на допустимость.

3) Выполняем аналогичные действия для тройки генов. И так далее.

Наиболее простые редуцирующие функции можно предложить для выпуклой области допустимых значений D простой формы. Рассмотрим в качестве этой области гиперпараллелепипед Π. Пусть в результате мутации допустимой особи s получена недопустимая особь $\tilde{s}$ (рис. 2.10).

В качестве функции $X' = r(X, \tilde{X})$ может быть использован оператор проецирования точки на гиперпараллелепипед, так что $X' = P_D(\tilde{X})$ (см. 1.3.2).

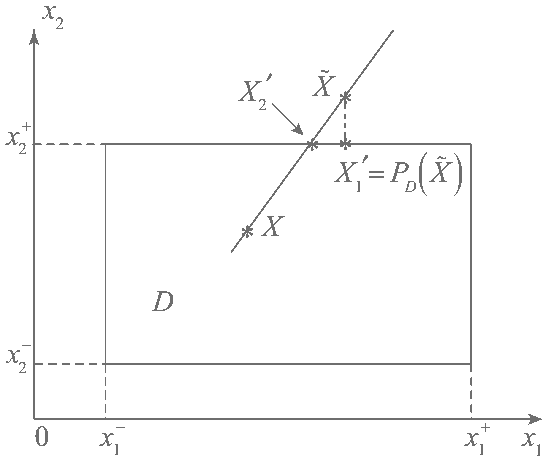


Рис. 2.10. К определению редуцирующей функции: $|X| = 2$

Восстановленную особь s' можно отождествить также с вектором X', соответствующим точке, в которой прямая, соединяющая точки X, $\tilde{X}$, пересекает грань гиперпараллелепипеда D (рис. 2.10).

Гибридные методы. В данном классе методов рассмотрим *метод поведенческой памяти*. В этом случае ограничения подключают по одному по завершении определенного числа итераций алгоритма. Сначала с использованием того или иного из рассмотренных методов учета ограничений добиваемся того, чтобы большая часть особей популяции удовлетворяли первому из ограничений. Затем активизируем второе ограничение и добиваемся удовлетворения как первого, так и второго ограничений. И так далее до исчерпания всех ограничений.

2.7. ТИПОВЫЕ ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ

Приведенный перечень наиболее известных способов кодирования особей, а также операторов мутации, кроссовера, отбора, селекции и, наконец, методов нишевания и учета ограничений показывает, что можно предложить чрезвычайно большое число генетических алгоритмов, отличающихся способами кодирования, структурой, используемыми генетическими операторами и их параметрами и т. д. Некоторые ориентиры в многообразии возможных генетических алгоритмов дают типовые алгоритмы.

Канонический (*canonical*) *генетический алгоритм* Холланда имеет следующие характеристики:

– бинарное кодирование особей;
– фиксированная разрядность генов;
– постоянный размер популяции;
– отсутствие промежуточной популяции;
– управление популяцией методом рулеточного отбора;
– селекция особей для скрещивания методом панмиксии;
– использование одноточечных кроссовера и мутатора.

Простой (*simple*) *генетический алгоритм* Голдберга (*D. E. Goldberg*) отличает от канонического только использование вместо рулеточного отбора метода турнирного отбора.

Генетический алгоритм Genitor Уитли (*D. Whitley*) отличается от канонического генетического алгоритма следующими своими свойствами:

– на каждой итерации алгоритм создает одну особь-потомка от одной случайной родительской пары особей;

– особь-потомок заменяет не родительскую особь, а одну из худших особей популяции (в исходном варианте алгоритма – наихудшую);

– отбор особи для указанной замены производится методом рангового отбора.

Таким образом, генетический алгоритм *Genitor* является примером реализации подхода, в котором популяция обновляется частями, а не вся сразу.

Одним из достоинств алгоритма *Genitor* является то, что он требует в два раза меньше памяти ЭВМ для хранения информации о популяции по сравнению с любым генетическим алгоритмом, использующим промежуточную популяцию. С другой стороны, алгоритм *Genitor* имеет по сравнению с каноническим генетическим алгоритмом значительно худшие диверсификационные свойства, что обусловлено потенциально быстрым обеднением разнообразия популяции за счет возможного длительного сохранения в ней лучших особей и быстрого исключения из нее наименее приспособленных особей.

Генетический алгоритм CHC Эшельмана (*Cross-population selection, Heterogenous recombination and Cataclysmic mutation, Ch. L. Eshelman*) определяют его следующие основные свойства:

– среди особей-родителей и их особей-потомков выбирают некоторое число лучших *различных* особей;

– для скрещивания выбирают случайную пару особей-родителей такую, что хеммингово расстояние между ними не менее заданного и расстояние между крайними различающимися битами также не менее заданного;

– для скрещивания особей-родителей используют полуоднородный кроссовер *HUX (Half Uniform Crossover),* когда особи-потомку переходит ровно половина битов каждой из особей-родителей;

– популяция содержит всего около 50 особей;

– в состоянии стагнации применяют катастрофическую мутацию (*cataclysmic mutation*), когда все особи популяции, кроме самой приспособленной, подвергают сильной мутации, изменяющей около трети битов каждой особи.

Подчеркнем, что алгоритм *CHC* использует мутацию только для преодоления стагнации. В основном цикле алгоритм использует только кроссовер.

Замечание. С точки зрения теоремы шим (см. 2.8), мутация вредит росту числа представителей наиболее приспособленных шим в популяции, поскольку лишь разрушает их. Однако для бинарных генетических алгоритмов с малым размером популяции мутация совершенно необходима, поскольку таким алгоритмам свойственна *преждевременная сходимость* (*premature convergence*), когда в некоторых разрядах все особи имеют один и тот же бит, не соответствующий глобальному экстремуму.

Гибридные (hybrid) генетические алгоритмы чаще всего основаны на комбинации генетического алгоритма и какого-либо локального алгоритма поиска (например, градиентного, если фитнес-функция дифференцируема). На каждой итерации гибридного алгоритма векторы варьируемых параметров, соответствующие каждой особи-потомку, используют в качестве стартовых точек для указанного алгоритма локального поиска. Найденные в результате

этой процедуры особи заменяют в популяции стартовых особей. Полученную популяцию обрабатывают в соответствии с используемым генетическим алгоритмом. Говоря нестрого, в гибридном алгоритме данного класса каждая из особей достигает локального минимума, в окрестности которого она находится, и только после этого подвергается воздействию обычных генетических операторов. Можно считать, что гибридный алгоритм реализует комбинацию моделей дарвиновской и (ошибочной) ламарковской эволюции, в соответствии с которой особь способна обучаться и записывать полученные навыки в свой генотип, чтобы затем передать их особям-потомкам.

Часто в вычислительной практике в качестве алгоритма локальной оптимизации используют так называемый *алгоритм восхождения на гору* (*hill climbing*). В случае бинарного кодирования особей идея алгоритма состоит в поочередном инвертировании каждого бита хромосомы, вычислении приспособленности полученных особей-потомков и выборе наиболее приспособленной из них.

По первому впечатлению гибридные алгоритмы, основанные на комбинации генетического алгоритма и алгоритма локального поиска, сдвигают баланс между интенсивностью и широтой поиска в сторону интенсивности и не могут в этой связи обеспечить высокую вероятность локализации глобального экстремума многоэкстремальной функции. Однако на практике во многих случаях такие алгоритмы оказываются весьма эффективными, поскольку при не слишком большом числе локальных экстремумов фитнес-функции вероятность попадания хотя бы одной особи в окрестность глобального экстремума этой функции достаточно велика.

Путем изменения отношения числа итераций, отводимых на решение каждой из локальных задач оптимизации, к максимально допустимому числу поколений генетического алгоритма можно регулировать баланс между интенсификацией и диверсификацией поиска.

Отметим, что алгоритмы, в которых тем или иным образом комбинируют один из алгоритмов глобальной оптимизации с некоторым алгоритмом локальной оптимизации, относят к классу меметических (*memetic*) алгоритмов (см. 5.7). Эти же алгоритмы иногда называют *ламарковскими* (*Lamarckian*) или алгоритмами с *эффектом Болдуина* (*Baldwin effect*). Последний термин связан с одним из наиболее правдоподобных вариантов ламаркианизма.

Гибридизацию поисковых алгоритмов можно реализовать также путем их переключения по некоторым правилам в процессе поиска. Пример такого подхода – модель *обучаемой эволюции* (*learnable evolution*), в которой осуществляется переключение между эволюционным алгоритмом и методом *машинного обучения*.

Многопопуляционные генетические алгоритмы обычно используют для организации параллельных вычислений, однако эффективным может быть использование таких алгоритмов также на традиционных однопроцессорных ЭВМ. Известными вариантами многопопуляционного генетического алгоритма являются *островной (island) генетический алгоритм* и *клеточный* (*cellular*) *генетический алгоритм*. Многопопуляционные генетические алгоритмы рассматриваются в гл. 6.

Адаптивные генетические алгоритмы. Как было показано выше, многие генетические операторы имеют свободные параметры. Эти параметры могут изменять свои значения в процессе вычислений, то есть быть нестационарными.

Сделать параметр b нестационарным можно, заменив его статическое значение некоторой детерминированной либо стохастической функцией $b(t)$, где t – счетчик поколений генетического алгоритма. Возможен также альтернативной способ, основанный на использовании обратной связи значений этого параметра с эффективностью оператора. По отношению к оператору мутации известным примером такого подхода является «правило 1/5» (*one-fifth rule*), в соответствии с которым доля успешных мутаций должна быть равной 1/5. Так что если эта доля меньше 1/5, то значения соответствующих свободных параметров мутатора изменяют таким образом, чтобы вероятность мутации увеличилась, а если больше 1/5 – уменьшилась. Здесь под успешной понимают мутацию, которая производит особь-потомка, более приспособленную по сравнению со своей особью-родителем. Такого рода параметры называют адаптивными.

Генетический алгоритм, использующий адаптивные параметры своих генетических операторов, называют *адаптивным*. Адаптивные популяционные алгоритмы рассматриваем в гл. 7.

Генетический микроалгоритм представляет собой модификацию канонического генетического алгоритма, предназначенную для решения задач, не требующих больших популяций и длинных хромосом. Такой алгоритм целесообразно использовать в случае жесткого лимита времени, когда решение, пусть и квазиоптимальное, необходимо найти быстро. Можно сказать, что при синтезе генетического микроалгоритма по критериям обеспечиваемого качества решения (его близости к глобальному экстремуму) и необходимым вычислительным затратам преимущество отдано второму критерию. Схема одного из вариантов генетического микроалгоритма имеет следующий вид.

1) Формируем популяцию $S = (s_i, i \in [1:|S|])$, где число особей $|S| = 5$.

2) Вычисляем приспособленность $\varphi(s_i)$ каждой из особей, находим лучшую особь, обозначаем ее s_5' и переносим в следующее поколение (элитарный отбор).

3) Методом турнирной селекции из числа особей $s_i, i \in [1:5]$ выбираем для репродукции четыре родительские особи.

4) Выполняем скрещивание отобранных особей с вероятностью, равной единице, так чтобы получить четыре особи-потомка. Мутацию особей не производим.

5) С помощью того или иного критерия проверяем сходимость алгоритма. В случае сходимости переходим к шагу 1, в противном случае – к шагу 2.

Особенностями генетического микроалгоритма являются: небольшой и фиксированный размер популяции; элитарный отбор особей; детерминированная селекция; скрещивание с вероятностью, равной единице; неиспользование мутации; многократное повторение процедуры рестарта алгоритма.

2.8. ТЕОРИЯ ШИМ

В англоязычной литературе данному понятию соответствует термин *shema* (буквально – схема). Однако слово «схема» чрезвычайно перегружено в русском языке. Поэтому в русскоязычной литературе в качестве аналога термина *shema* часто используют термин «*шима*». Мы также используем этот термин.

Определение 2.1. Шимой H^s называют бинарную строку вида

$$h_1,\ h_2,\ldots,\ h_{|H|},\ h_i \in \{0;1;*\},$$

где символ «*» обозначает неопределенное значение (как 0, так и 1).

Шима, таким образом, задает множество всех возможных бинарных строк длиной $|H|$, в которых на месте символов «*» находятся либо нули, либо единицы. Другими словами, шима определяет общие участки набора всех возможных бинарных строк длиной $|H|$. В контексте задачи оптимизации шима описывает множество точек пространства решения $R^{|X|}$, которые имеют определенное сходство между собой.

С шимой H^s обычно связывают следующие ее параметры.

Порядок (*order*) *шимы* $o(H^s)$ равен числу ее определенных разрядов (символов «0» или «1»). Например, $o(*001)=3$, $o(**0*1**)=2$. Хромосому H будем считать шимой, порядок которой равен $|H|$.

Определяющая длина (*defining length*) шимы $l(H^s)$ представляет собой расстояние между двумя крайними определенными разрядами этой шимы. К примеру, $l(*0*1)=2$, $l(01*0*1)=5$.

Приспособленность шимы $\varphi(H^s)$ есть не что иное, как среднее значение фитнес-функции $\varphi(X)$ на всех решениях, которые определяет данная шима.

Рассмотрим следующий пример. Положим, что $X=(x_1,x_2)$, где величины x_1, x_2 могут принимать только целочисленные значения в интервале [0:3]. Пусть $|H_1^g|=|H_2^g|=2$ и функция приспособленности имеет вид $\varphi(x_1,x_2)=f(x_1,x_2)=x_1^2+x_2^2$. Тогда шима $H^s=*0*1$ определяет следующие бинарные строки.

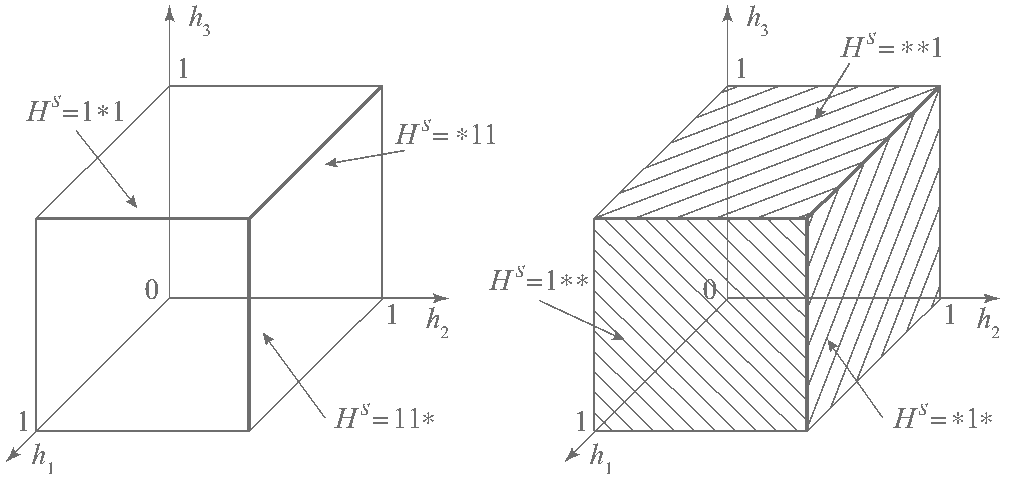
Строка	$\varphi(x_1,x_2)$
0001	$0+1^2=1$
1001	$2^2+1^2=5$
0011	$0+3^2=9$
1011	$2^2+3^2=13$

Приспособленность данной шимы равна

$$\varphi(H^s)=\frac{1+5+9+13}{4}=7{,}0.$$

Содержательной является также следующая интерпретация шимы. Во множестве всех возможных значений бинарных разрядов хромосомы H (в поисковом множестве) шима H^s порядка $o(H^s)<|H|$ определяет гиперплоскость размерности $(|H|-o(H^s))$. Отметим, что мощность указанного множества равна, очевидно, $2^{|H|}$.

Пусть для примера $|H|=3$. Тогда шимы второго порядка $H_1^s=*11$, $H_2^s=1*1$, $H_3^s=11*$ определяют прямые (гиперплоскости единичной размер-



а *б*

Рис. 2.11. Гиперплоскости, определяемые некоторыми шимами, $|H|=3$: *а* – шимы второго порядка; *б* – шимы первого порядка

ности), а шимы первого порядка $H_4^s = **1$, $H_5^s = 1**$, $H_6^s = *1*$ – плоскости (гиперплоскости размерности два) – рис. 2.11.

Теорема 2.2 (*Теорема шим*). Пусть генетический алгоритм на основе пропорциональной селекции и одноточечного кроссовера использует монохромосомы длиной H. Пусть также $m(H^s,t)$ – число экземпляров шимы H^s в поколении с номером t, ξ_c – вероятность выполнения операции кроссовера, а $\overline{\varphi}(t)$ – средняя приспособленность особей популяции в поколении t. Тогда число экземпляров той же шимы в поколении с номером $(t+1)$ определяет формула

$$m(H^s,t+1) \geq m(H^s,t)\frac{\varphi(H^s,t)}{\overline{\varphi}(t)}\left(1-\xi_c\frac{l(H^s)}{|H|-1}\right). \qquad (2.8)$$

Доказательство. Легко видеть, что вероятность попадания точки скрещивания внутрь строки, соответствующей шиме H^s, равна $\xi_c\dfrac{l(H^s)}{|H|-1}$. Отсюда следует, что вероятность неразрушения этой шимы оператором кроссовера определяет выражение

$$1-\xi_c\frac{l(H^s)}{|H|-1}. \qquad (2.9)$$

Вследствие использования пропорциональной селекции вероятность участия в скрещивании строки, соответствующей шиме H^s, равна

$$m(H^s,t)\frac{\varphi(H^s,t)}{\overline{\varphi}(t)} = m(H^s,t)\xi(H^s). \qquad (2.10)$$

Справедливость формулы (2.8) непосредственно следует из формул (2.9), (2.10) •

Заметим, что теорема не учитывает влияние мутации на разрушение шимы.

Теорема 2.3 (*Модифицированная теорема шим*). Пусть выполнены условия теоремы 2.2 и вероятность мутации одного разряда хромосомы равна ξ_m. Тогда число экземпляров шимы H^s в поколении с номером $(t+1)$ равно

$$m(H^s,t+1) \geq m(H^s,t)\frac{\varphi(H^s,t)}{\overline{\varphi}(t)}\left(1-\xi_c\frac{l(H^s)}{|H|-1}\right)\left(1-\xi_m\right)^{o(H^s)}. \qquad (2.11)$$

Доказательство. При мутации шима H^s не изменяется, если мутирующий разряд не попадает ни в одну из ее определенных позиций (содержащих величины «0» либо «1»). Отсюда следует, что вероятность неразрушения шимы H^s равна $\left(1-\xi_m\right)^{o(H^s)}$. Эта оценка вместе с формулой (2.8) и дает формулу (2.11) •

Теоремы 2.2, 2.3 формализуют основные аспекты поведения генетического алгоритма. Так, эти теоремы учитывают, что оператор мутации с большей вероятностью разрушает шимы высокого порядка $o(H^s)$, а оператор кроссовера – шимы с большей определяющей длиной $l(H^s)$. Скорость сходимости популяции пропорциональна мере давления селекции – отношению приспособленности лучшей особи к средней приспособленности в популяции $\frac{\varphi(H^s,t)}{\overline{\varphi}(t)}$. Число шим в популяции, приспособленность которых выше средней, растет с ростом числа поколений экспоненциально. В то же время шимы с приспособленностью ниже средней с ростом числа поколений распадаются с той же скоростью. Увеличение вероятностей ξ_c, ξ_m и уменьшение давления отбора диверсифицирует поиск, но замедляет сходимость алгоритма. Напротив, уменьшение указанных вероятностей и увеличение давления отбора интенсифицируют поиск (ведут к быстрой сходимости алгоритма), но сужают исследованную часть пространства поиска. Варьируя значения указанных величин, можно обеспечить приемлемый баланс между указанными тенденциями для каждой задачи оптимизации.

В теоремах 2.2, 2.3 присутствуют знаки неравенства, поскольку эти теоремы не учитывают тот факт, что операторы кроссовера и мутации могут не только разрушать данную шиму, но и создавать ее из других шим. Теоремы не применимы к генетическим алгоритмам, отличным от канонического алгоритма. Подчеркнем, что теоремы позволяют вычислить число экземпляров данной шимы только в следующем поколении.

Пример 2.1. Пусть $\varphi(X)=\varphi(x_1)=x_1$ и, как показано ниже, длина хромосомы $|H|=2$ и размер популяции $|S|=4$.

Рассмотрим генетический алгоритм, в котором мутации и скрещивания отсутствуют ($\xi_m=0$, $\xi_c=0$). Возможные в этих предположениях шимы и значения их параметров при выполнении одной итерации генетического алгоритма имеют вид, представленный в табл. 2.7. В результате одной итерации генетического алгоритма шимы, имеющие высокую приспособленность (шимы 11, 1*, *1), увеличили свое представительство в популяции. В то же время шимы с низкой приспособленностью (шимы 0*, *0) уменьшили его •

H	Хромосома	$\varphi(H)$
H_1	00	0
H_2	01	1
H_3	10	2
H_4	11	3

Таблица 2.7 **К теореме шим**

H^s	$m(H^s,t)$	$\varphi(H^s)$	$l(H^s)$	$o(H^s)$	$\xi(H^s)$	$m(H^s,t+1)$
00	1	0	1	2	0,00	0
01	1	1	1	2	0,67	1
0*	2	0,5	0	1	0,33	1
10	1	2	1	2	1,33	1
11	1	3	1	2	2,0	2
1*	2	2,5	0	1	1,67	3
*0	2	1	0	1	0,67	1
*1	2	2	0	1	1,33	3
**	4	1,5	0	0	1,0	4

2.9. ЭВОЛЮЦИОННАЯ СТРАТЕГИЯ

Рассмотрим задачу (2.1) глобальной условной минимизации целевой и фитнес-функций с ограничениями типа неравенств. Алгоритм эволюционной стратегии решения этой задачи разработан в 1964 г. Рехенбергом (*I. Rechenberg*). Суть алгоритма передает следующая последовательность его основных шагов.

1) Производим мутации и скрещивания особей текущей популяции.

2) Родительских особей и полученных таким образом особей-потомков включаем в промежуточную популяцию.

3) На основе этой популяции формируем следующую популяцию путем детерминированного *отбора без повторений* наиболее приспособленных особей.

Перечислим основные отличия алгоритма эволюционной стратегии от классического генетического алгоритма. Прежде всего, если генетический алгоритм моделирует эволюцию на уровне геномов особей, то алгоритм эволюционной стратегии ориентирован на эволюцию их фенотипов. Поэтому данный алгоритм использует только вещественное кодирование особей, в отличие от классического генетического алгоритма, предполагающего бинарное кодирование.

В классическом генетическом алгоритме процесс отбора особей из родительской популяции для включения их в новую популяцию осуществляют случайным образом пропорционально их приспособленности. В алгоритме эволюционной стратегии отбор особей в новую популяцию производят без повторений на основе детерминированной процедуры.

Алгоритм эволюционной стратегии предполагает, что на каждой итерации сначала производится модификация особей (мутация), а затем селекция. Классический генетический алгоритм выполняет эти действия в обратном порядке. Кроме того, этот алгоритм использует стационарные значения свободных па-

раметров своих операторов (к примеру, вероятности скрещивания и мутации особей), а алгоритм эволюционной стратегии, напротив, непрерывно изменяет значения этих параметров в режиме самоадаптации (гл. 7).

Отметим также отличия в способах учета ограничений на компоненты вектора варьируемых параметров. В алгоритме эволюционной стратегии особей-потомков, не удовлетворяющих хотя бы одному из ограничений, определяющих область допустимых значений D, отвергают, то есть не включают в новую популяцию. Если число таких отвергнутых особей превышает некоторую заранее заданную величину, то запускается процесс адаптации параметров алгоритма (например, увеличивается вероятность мутации). В классическом генетическом алгоритме значения свободных параметров генетических операторов являются стационарными, и указанные ограничения учитывают с помощью штрафных функций.

Современные генетические алгоритмы значительно отличаются от классического, так что, как отмечалось выше, различия между ними и алгоритмом эволюционной стратегии незначительны.

Основой канонического алгоритма эволюционной стратегии является комбинирование генетических операторов мутации и селекции по следующему правилу.

1) Выбираем из текущей популяции S особь s_i и путем применения к ней оператора мутации получаем особь s_i'.

2) Вычисляем приспособленности $\varphi(s_i)$, $\varphi(s_i')$ указанных особей и помещаем в следующую популяцию $S(t+1)$ наиболее приспособленную из них.

Здесь индекс i последовательно принимает значения $i = 1, 2, ..., |S|$. Алгоритм селекции, используемый на последнем шаге, представляет собой так называемый *алгоритм селекции усечением* (*truncation*).

Для генерации случайной начальной популяции $S(0)$ необходимо, чтобы для каждой компоненты x_j, $j \in [1:|X|]$ вектора варьируемых параметров X был задан интервал $[x_j^{\min}; x_j^{\max}]$ (используемый только на этапе инициализации). Для особи s_i, $i \in [1:|S|]$ начальные значения компонент $x_{i,j}$, $j \in [1:|X|]$ вектора X_i полагают равномерно распределенными в указанном интервале.

Подобно генетическому алгоритму, алгоритм эволюционной стратегии в качестве условия останова может использовать следующие события:

– достижение заданного числа поколений $\hat{t}$;

– достижение заданного состояния популяции, когда, к примеру, приспособленности всех особей популяции опустились ниже некоторого порога (напомним, что речь идет о задаче минимизации фитнес-функции);

– достижение заданного уровня сходимости, когда в некоторой метрике расстояния между особями популяции меньше определенного порогового значения.

Обозначим $\alpha = |S|$ число особей текущей популяции (родительских особей), а $\beta = |S'| > \alpha$ – число особей-потомков. Известны так называемые $(\alpha+\beta)$- и $\alpha\beta$-алгоритмы эволюционной стратегии.

$(\alpha+\beta)$-алгоритм предполагает, что в промежуточную популяцию S' входят все $(\alpha+\beta)$ особи и из их числа производят селекцию и включение в популяцию $S(t+1)$ лучших особей числом α. Из определения $(\alpha+\beta)$-алгоритма

следует, что он обеспечивает отличную от нуля вероятность попадания в популяцию $S(t+1)$ некоторых родительских особей. Заметим, что во введенных обозначениях канонический генетический алгоритм, очевидно, относится к классу (1+1)-алгоритмов эволюционной стратегии.

В случае $\alpha\beta$ -алгоритма в промежуточную популяцию S' включают только β особей-потомков и из их числа производят отбор и включение в популяцию S' лучших особей числом α . Очевидно, что $\alpha\beta$ -алгоритм исключает возможность попадания в популяцию $S(t+1)$ родительских особей.

Свободным параметром $(\alpha+\beta)$ - и $\alpha\beta$ -алгоритмов является отношение $r=\dfrac{\beta}{\alpha}\geq 7$. С увеличением значений этого параметра возрастает вероятность того, что каждая особь-родитель произведет по меньшей мере одну особь-потомка, более приспособленную, чем она сама. При возрастании значений параметра r различия между $(\alpha+\beta)$ - и $\alpha\beta$ -алгоритмами становятся менее существенными.

Для $(\alpha+\beta)$ -алгоритма доказана его сходимость по вероятности, а для $\alpha\beta$ -алгоритма вопрос о сходимости остается открытым.

По сравнению с $\alpha\beta$ -алгоритмом, $(\alpha+\beta)$ -алгоритм, вообще говоря, более «экономно» использует найденные высокоприспособленные особи, поскольку оставляет их для конкуренции с потомками. Это качество $(\alpha+\beta)$ -алгоритма повышает интенсивность поиска, но уменьшает его широту и может привести к преждевременной сходимости алгоритма к некоторому локальному минимуму фитнес-функции.

Оператор мутации в каноническом алгоритме эволюционной стратегии является единственным эволюционным оператором. Оператор использует алгоритм, близкий к алгоритму гауссовой мутации, в соответствии с которым значение гена x'_j хромосомы X' определяют на основе значения гена x_j хромосомы X по формуле

$$x'_j = x_j + N_1(m,\sigma)\,,\ \ j\in[1:|X|], \tag{2.12}$$

где среднее квадратичное отклонение σ интерпретируют как вероятность мутации. Обычно используют нулевое математическое ожидание $m=0$, а значение величины σ меняют в процессе функционирования алгоритма в режиме самоадаптации, так что оператор мутации оказывается самоадаптирующимся (*self adaptive*). При этом параметр σ может включаться в расширенный вектор варьируемых параметров задачи, который эволюционирует по общим правилам изменения особей алгоритма эволюционной стратегии (гл. 7).

В простейшем случае изменение среднего квадратичного отклонения σ может быть подчинено правилу одной пятой (см. 2.7). В соответствии с этим правилом через каждое фиксированное число итераций проверяют число успешных мутаций (породивших более приспособленных особей). Если их относительное количество превышает 1/5 (популяция сосредоточена в окрестности некоторого оптимума), то величину σ по некоторому закону увеличивают, а в противном случае – уменьшают.

2.10. ЭВОЛЮЦИОННОЕ ПРОГРАММИРОВАНИЕ

Аналогично предыдущему параграфу рассматриваем задачу (2.1) глобальной условной минимизации целевой и фитнес-функций с ограничениями типа неравенств.

Основные отличия алгоритма эволюционного программирования от алгоритма эволюционной стратегии заключаются в следующем. Во-первых, алгоритм эволюционного программирования использует только $(\alpha+\beta)$-стратегию и только при $\alpha=\beta$ (когда числа особей $|S|$, $|S'|$ текущей и промежуточной популяций одинаковы). Во-вторых, последний алгоритм допускает использование как вещественного, так и любого другого кодирования особей.

Рассмотрим случай вещественного кодирования. Полагаем, что фитнес-функция $\varphi(X)$ удовлетворяет условию $\varphi(X^*)=0$. Рекомендуемый размер популяции в алгоритме эволюционного программирования равен $|S|>200$. Алгоритм использует следующие основные операторы и процедуры.

Процедура инициализации предназначена для генерации случайной начальной популяции $S(0)$ и может быть реализована по общей для популяционных алгоритмов схеме.

Оператор репликации реализует создание первой части промежуточной популяции S' путем копирования в нее текущей популяции S, так что в популяции S' оказывается $|S'|=|S|$ особей.

Оператор мутации применяют к каждой из особей $s_i \in S$, $i \in [1:|S|]$, и полученные особи $s'_{|S|+i}$ заносят в промежуточную популяцию S' в качестве второй ее части. Таким образом, после выполнения данного оператора в популяции S' оказывается $|S'|=2|S|$ особей.

Алгоритм мутации близок к алгоритму гауссовой мутации и определяется формулой

$$x'_{|S|+i,j} = x_{i,j} + N_1(0;1)\sqrt{a_j\varphi(X_i)+b_j}\ ,\ i \in [1:|S|],\ j \in [1:|X|]. \qquad (2.13)$$

Здесь a_j, b_j – свободные параметры оператора, рекомендуемые значения которых равны 1 и 0 соответственно.

Формула (2.13) означает, что стандартное отклонение случайной величины $N_1(0;1)\sqrt{a_j\varphi(X_i)+b_j}$ (*ширина мутации*) пропорционально приспособленности родительской особи s_i. Если глобальный минимум фитнес-функции, как мы условились, равен нулю, то по мере приближения к точке X^* ширина мутации будет сокращаться. Если фитнес-функция не удовлетворяет указанному условию, то для его соблюдения соответствующим образом подбирают значения параметров a_j, b_j.

Оператор селекции обеспечивает формирование популяции $S(t+1)$, содержащей $|S|$ особей, на основе промежуточной популяции S' из $|S'|=2|S|$ особей. Оператор использует турнирную селекцию и имеет следующий вид.

1) Полагаем $i=1$.

2) Выбираем из популяции S' особь s'_i.

3) С помощью генератора натуральных случайных чисел, равномерно распределенных в интервале $[1:2|S|]$, выбираем из популяции S' особей s'_{i_1},

$s'_{i_2}, \ldots, s'_{i_n}$ таких, что $i_j \neq i$, $j \in [1:n]$. Здесь n – натуральное число (свободный параметр оператора), рекомендуемое значение которого лежит в интервале $[0{,}05|S|:0{,}1|S|]$.

4) Сравниваем приспособленность особи s'_i с приспособленностями отобранных особей и определяем число «побед» v_i особи s'_i по правилу

$$v_i = \sum_{j=1}^{n} \begin{pmatrix} 1, & \varphi(s'_i) \leq \varphi(s'_{i_j}) \\ 0, & \text{иначе} \end{pmatrix}.$$

5) Если популяция S' не исчерпана, то полагаем $i = i+1$ и переходим к шагу 2. В противном случае переходим к следующему шагу 6.

6) Представляем популяцию S' в виде списка, особи которого упорядочены в порядке убывания числа их побед, а при равном числе побед – в порядке возрастания значений их приспособленности.

7) Включаем в популяцию $S(t+1)$ первые $|S|$ особей указанного списка.

Отметим, что рассмотренный механизм формирования популяции $S(t+1)$ не исключает возможности включения в нее малоприспособленных особей. С ростом значения параметра n этот механизм начинает все более приближаться к элитарному отбору.

Процедура останова. Алгоритм эволюционного программирования в качестве условия останова может использовать те же события, что и алгоритм эволюционной стратегии.

Схему канонического алгоритма эволюционного программирования определяет следующая последовательность его шагов.

1) Инициализируем начальную популяцию $S(0)$ и полагаем счетчик числа поколений $t = 0$.

2) Формируем промежуточную популяцию S' путем последовательного выполнения операторов репликации и мутации.

3) С помощью оператора селекции формируем следующую популяцию $S(t+1)$.

4) Проверяем условие окончания итераций. Если это условие выполнено, завершаем вычисления, иначе полагаем $t = t+1$ и переходим к шагу 2.

Как и для генетического алгоритма, свободные параметры алгоритма эволюционного программирования следует выбирать таким образом, чтобы обеспечить желаемый баланс между интенсивностью и широтой поиска.

Для варианта канонической схемы алгоритма эволюционного программирования, в котором свободные параметры оператора мутации имеют значения $a_j = 1$, $b_j = 0$, $j \in [1:|X|]$, известно доказательство его асимптотической сходимости по вероятности, основанное на теории марковских цепей. Доказательство существенно использует предположение, что $\varphi(X) \geq 0$ для любых $X \in D$.

Экспериментальные исследования выявили ряд проблем, возникающих при практическом использовании алгоритма эволюционного программирования. Во-первых, если глобальный минимум фитнес-функции отличается от нуля, то алгоритм может терять устойчивость, а если значения этой функции

достаточно велики, то поиск становится квазислучайным. В то же время в ситуации, когда априорная информация о величине глобального минимума отсутствует (что типично для практических задач оптимизации), затруднительным является подбор значений параметров a_j, b_j, $j \in [1:|X|]$ в формуле (2.13), позволяющих трансформировать фитнес-функцию так, чтобы можно было избежать указанного эффекта.

Во-вторых, алгоритм эволюционного программирования имеет в общем случае примерно $2|X|$ свободных параметров. Выбор оптимальных или хотя бы разумных значений этих параметров представляет собой серьезную проблему. Напомним, что генетический алгоритм также не свободен от данной проблемы.

Известно большое число модификаций канонического алгоритма эволюционного программирования. Например, процедура формирования промежуточной популяции может быть модифицирована следующим образом.

1) По рассмотренной схеме выполняем оператор репликации, так что в промежуточной популяции S' оказывается $|S'| = |S|$ особей.

2) С помощью генератора натуральных случайных чисел, равномерно распределенных в интервале $[1:|S|]$, выбираем из популяции S' особь s'_i.

3) Реализуем мутацию родительской особи s'_i и полученную особь помещаем в популяцию S', так что число особей в ней становится равным $|S|+1$.

4) Выполняем детерминированную селекцию полученной популяции S' – исключаем из нее наименее приспособленную особь, в результате чего в популяции вновь остается $|S|$ особей.

5) Повторяем шаги 2–5 некоторое число раз (например, $|S|$ раз) либо до достижения заданного качества популяции S' (например, заданного уровня ее разнообразия).

2.11. ДИФФЕРЕНЦИАЛЬНАЯ ЭВОЛЮЦИЯ

В отличие от двух предыдущих параграфов здесь речь идет о задаче глобальной условной максимизации целевой и фитнес-функций с ограничениями типа неравенств.

Алгоритм дифференциальной эволюции был разработан Прайсом (*K. Price*) и Сторном (*R. Storn*) в 1995 г. Алгоритм использует адаптивную мутацию особей по правилу, подобному правилу одной пятой (см. 2.7). При этом если особи популяции сильно распределены в пространстве поиска, то алгоритм мутации производит их существенные вариации. В противном случае, то есть когда особи распределены в пространстве поиска компактно, алгоритм реализует небольшие изменения особей.

Классический адаптивный оператор мутации для каждой особи текущей популяции s создает особь-потомка s' на основе трех других различных родительских особей s_1, s_2, s_3, случайным образом выбранных из этой популяции. Здесь s_1 – так называемая *базовая особь*. Хромосому X' получают путем линейной комбинации хромосом X_1, X_2, X_3, интерпретируемых как вещественные векторы:

$$X' = X_1 + a\,(X_2 - X_3).$$

Здесь a – скалярный *коэффициент влияния* (свободный параметр).

Легко видеть, что на начальных итерациях алгоритма, когда особи популяции S находятся «далеко» друг от друга (популяция сильно распределена), модуль *вектора мутации* ($X_2 - X_3$), вероятнее всего, будет иметь большое значение, и вектор X' окажется сильно отличающимся от вектора X_1. Напротив, на завершающих итерациях поиска, когда все особи текущей популяции находятся «вблизи» друг друга (и «вблизи» глобального либо локального экстремума фитнес-функции), модуль указанного вектора с высокой вероятностью будет иметь небольшое значение, так что векторы X', X_1 оказываются также «близкими».

После мутации определяют приспособленность полученной особи s' и, если она выше приспособленности родительской особи s_1, заменяют последнюю особью s'.

Рассмотрим прежде ситуацию, когда множество допустимых значений вектора варьируемых параметров представляет собой гиперпараллелепипед Π. Вообще говоря, алгоритм дифференциальной эволюции может использовать бинарное, целочисленное и вещественное кодирования особей. Ограничимся рассмотрением вещественного алгоритма дифференциальной эволюции.

Начальную популяцию $S(0) = \{s_i(0), i \in [1:|S|]\}$ в алгоритме дифференциальной эволюции получают по правилу

$$x_{i,j} = x_i^- + U_1(x_i^-; x_i^+)\,,\ \ j \in [1:|X|].$$

Каждая итерация алгоритма состоит из следующих основных шагов.

1) В текущей популяции $S = \{s_i,\ i \in [1:|S|]\}$ выбираем особь s_i.

2) С помощью генератора натуральных случайных чисел, равномерно распределенных в интервале $[1:|S|]$, выбираем из популяции S особей s_{i_1}, s_{i_2}, s_{i_3}; $i_1 \neq i_2 \neq i_3 \neq i$.

3) С помощью аналогичного генератора случайных чисел находим индекс $j_1 \in [1:|X|]$ гена в хромосоме.

4) Формируем особь s_i' такую, что

$$x'_{i,j} = \begin{cases} x_{i_1,j} + a\,(x_{i_2,j} - x_{i_3,j}), & U_1(0;1) \le b \text{ и } (j = j_1), \\ x_{i,j}, & U_1(0;1) > b, \end{cases} \qquad j \in [1:|X|]. \qquad (2.14)$$

Здесь $a \in [0;2] = const$, $b \in [0;1]$ – свободный параметр алгоритма.

5) Производим селекцию особей по схеме

$$s_i' = \begin{cases} s_i', & \varphi(s_i') \ge \varphi(s_i), \\ s_i, & \text{иначе}. \end{cases}$$

Отметим, что в соответствии с формулой (2.14) случайный индекс j_1 в каждой из особей $s_i(t)$ текущей популяции обеспечивает изменение хотя бы одного гена.

Формула (2.14) не гарантирует, что все компоненты полученного вектора X_i' окажутся допустимыми, то есть не выйдут за границы гиперпараллелепипеда Π. Положим, что ограничение нарушено для компоненты $x_{i,j}'$, так что $x_{i,j}' < x_i^-$ либо $x_{i,j}' > x_i^+$. Тогда вычисляют новое значение этой компоненты по правилу

$$x_{i,j}' = x_i^- + U_1(x_i^-; x_i^+) .$$

В случае, когда множество допустимых значений вектора варьируемых параметров определяют функциональные ограничения типа неравенств $G(X) \geq 0$, эти ограничения учитывают посредством алгоритма штрафных функций. При этом фитнес-функцию $\varphi(X)$ формируют в виде

$$\varphi(X, A) = (f(X) + b) \prod_{k=1}^{|G|} (\tilde{g}_k(X, A))^{c_i} ,$$

где $A = (a_1, a_2, ..., a_{|G|})$ – вектор неотрицательных коэффициентов штрафа; b, $c_i \geq 1$ – свободные параметры алгоритма;

$$\tilde{g}_k(X, A) = \begin{cases} 1 + a_k g_k(X), & g_k(X) < 0, \\ 1, & \text{иначе.} \end{cases}$$

Значение параметра b выбирают таким, чтобы функция $\varphi(X, A)$ была неотрицательной при всех $X \in D$.

Известно большое число модификаций канонической схемы алгоритма дифференциальной эволюции. Во-первых, в качестве базовой особи можно использовать не особь, случайно выбранную из текущей популяции, а наиболее приспособленную особь. Во-вторых, вместо вектора мутации $\left(X_{i_2} - X_{i_3}\right)$ может быть использован вектор градиента функции $\varphi(X)$. Такой способ построения вектора мутаций порождает гибридный метод оптимизации, основанный на применении *ускоряющего оператора*. Компоненты хромосомы $X_i(t)$ в следующем поколении определяют в этом случае по правилу

$$X_i' = \begin{cases} X_i', & \varphi(s_i') > \varphi(s_i), \\ X_i' - a\,\nabla\varphi(X_i'), & \text{иначе.} \end{cases}$$

Здесь в правой части выражения X_i' представляет собой хромосому, полученную по схеме канонического алгоритма дифференциальной эволюции; $\nabla\varphi(X_i')$ – вектор градиента функции $\varphi(X)$ в точке X_i'; $a \in [0;1]$ – коэффициент влияния, имеющий смысл величины шага в направлении градиента указанной функции.

Ускоряющий оператор может также использовать лучшую особь s_* текущей популяции S, то есть наиболее приспособленную особь среди всех особей этой популяции.

В качестве еще одного примера модификации алгоритма дифференциальной эволюции рассмотрим *оператор миграции*, использующий указанную

особь s_* и ориентированный на интенсификацию поиска. Новые компоненты хромосомы X_i вычисляют в этом случае по правилу

$$x'_{i,j} = \begin{cases} x_{*,j} + U_1(x_i^-; x_{*,j}), & U_1(0;1) < \dfrac{x_{*,j} - x_i^-}{x_i^+ - x_i^-}, \\ x_{*,j} + U_1(x_{*,j}; x_i^+), & \text{иначе}, \end{cases} \quad i \in [1:|S|], \; j \in [1:|X|].$$

Здесь $X_* = (x_{*,1}, x_{*,2}, ..., x_{*|X|})$ – генотип особи s_*.

Оператор миграции используют при слишком однородной популяции. Мерой разнообразия популяции S может являться величина

$$\overline{\rho}(S) = \frac{\sum_{i=1}^{|S|} \sum_{j=1}^{|X|} \rho_j(s_i, s_*)}{|X|(|S|-1)},$$

где $\rho_j(s_i, s_*)$ – расстояние между j-ми компонентами генотипов особей s_i, s_*, определяемое формулой

$$\rho_j(s_i, s_*) = \begin{cases} 1, & abc\left(\dfrac{x_{i,j} - x_{*,j}}{x_{*,j}}\right) > b, \\ 0, & \text{иначе}. \end{cases}$$

Здесь $b \in [0;1]$ – свободный параметр оператора, имеющий смысл характерной величины покомпонентного различия хромосом популяции.

Малые значения величины $\overline{\rho}(S)$ говорят о концентрации всех особей популяции S в относительно небольшой окрестности лучшей особи, то есть об однородности популяции. Таким образом, оператор миграции используют в случае, когда $\overline{\rho}(S) \le \rho_{\max}$, где $\rho_{\max}$ – свободный параметр оператора.

2.12. ГЕНЕТИЧЕСКИЙ КОЭВОЛЮЦИОННЫЙ АЛГОРИТМ

Генетический коэволюционный алгоритм (*Genetic Coevolutionary Algorithm*, *GCA*) являет собой пример многопопуляционных многоагентных алгоритмов оптимизации, которые будут рассмотрены в гл. 6. Алгоритмы этого класса предполагают одновременную эволюцию в пространстве поиска мультипопуляции $\boldsymbol{S} = \{S_i, i \in [1:|\boldsymbol{S}|]\}$, включающей в себя $|\boldsymbol{S}|$ *субпопуляций* S_i.

Вообще говоря, алгоритм коэволюции может быть построен на основе различных популяционных алгоритмов поисковой оптимизации. Рассматриваем классический алгоритм коэволюции, предполагающий использование генетических алгоритмов в качестве субпопуляционных. Полагаем, что эти алгоритмы имеют одинаковую структуру и значения всех свободных параметров,

за исключением параметров, определяющих генетические операторы формирования нового поколения и селекции. По одной из рекомендаций указанные параметры следует выбирать таким образом, чтобы в субпопуляциях были примерно в одинаковой степени представлены три сорта генетических алгоритмов: алгоритмы, обладающие, в первую очередь, свойством диверсификации поиска; алгоритмы, ориентированные на интенсификацию поиска; алгоритмы, обеспечивающие как широту, так и точность поиска.

Можно предложить большое число других принципов выбора субпопуляционных алгоритмов: случайный выбор, исключающий лишь вариант выбора одного и того же алгоритма два раза; выбор по некоторым критериям лишь лучших алгоритмов; аналогичный выбор только худших алгоритмов и т. д.

К основным параметрам алгоритма коэволюции относятся: число субпопуляций $|\boldsymbol{S}|$; величина ресурса $n_f^{\max}$ – максимально допустимое число испытаний; интервал адаптации t_a; величина штрафа n_p, назначаемая проигравшим субпопуляционным алгоритмам; минимально допустимый размер субпопуляции $|S|_{\min} = \min\limits_{i\in[1:|\boldsymbol{S}|]} |S_i|$. Некоторые авторы рекомендуют использовать следующие значения указанных параметров: число субпопуляций $|\boldsymbol{S}| = (3 \div 5)$, интервал адаптации $t_a = (5 \div 7)$, величина штрафа $n_p = (0{,}02 \div 0{,}15)$, минимально допустимый размер субпопуляции $|S|_{\min} = (0{,}05 \div 0{,}35)|S_i|$.

Если на каждой итерации каждого из субпопуляционных алгоритмов производится одно испытание, то ресурс $n_f^{\max}$ определяет формула

$$n_f^{\max} = \hat{t} \sum_{i=1}^{|\boldsymbol{S}|} |S_i|,$$

где, напомним, $\hat{t}$ – максимально допустимое число итераций. Таким образом, задание ресурса $n_f^{\max}$ эквивалентно в этом случае заданию начального размера субпопуляций $|S_i|$ и величины $\hat{t}$. Изначально ресурс распределяют обычно между субпопуляциями поровну.

Величину интервала адаптации $t_a \in (0; \hat{t})$ назначаем, исходя из следующих соображений. Если эта величина мала, то субпопуляционные алгоритмы не успеют продемонстрировать особенности своего поведения, то есть все субпопуляции покажут близкие результаты и адаптация потеряет смысл. Большие значения величины t_a уменьшают эффективность алгоритма коэволюции, поскольку в этом случае значительное число субпопуляций может «уйти» в неперспективные подобласти области поиска.

Важной величиной в алгоритме коэволюции является также величина штрафа $n_p \in [0; 1]$, имеющая смысл доли, на которую сокращают размеры проигравших субпопуляций. Значение этой величины также не должно быть слишком маленьким (иначе субпопуляционные алгоритмы не «почувствуют» изменений) или слишком большим (поскольку генетический алгоритм с маленьким размером популяции априори неэффективен).

Минимально допустимый размер субпопуляции $|S|_{\min}$ ограничивает снизу размер проигравших субпопуляций (как и в случае параметра n_p, исходя из со-

ображений низкой эффективности генетического алгоритма с маленьким размером популяции).

Основные шаги общей схемы генетического коэволюционного алгоритма определяет следующая последовательность.

1) Инициализируем мультипопуляцию.

2) Выполняем t_a независимых итераций для каждой из субпопуляций.

3) Оцениваем текущие эффективности субпопуляций.

4) Проверяем выполнение условий останова.

5) Перераспределяем ресурсы между субпопуляциями и повторяем шаги 2–5 до выполнения условий останова.

Инициализация мультипопуляции. На данном шаге алгоритма задаем число и значения свободных параметров субпопуляций, значения параметров коэволюционного алгоритма, а также по схеме генетического алгоритма инициализируем субпопуляции.

Выполнение независимых итераций включает в себя, вообще говоря, асинхронное выполнение t_a итераций каждого из субпопуляционных генетических алгоритмов.

Оценку эффективности субпопуляций производим на основе оценки текущих значений их функции пригодности $\phi_i = \phi_i(t), i \in [1:|S|]$. В качестве функции $\phi_i(t)$ может быть использована, например, функция

$$\phi_i(t) = \sum_{\tau=0}^{t_a-1} \frac{t_a - \tau}{\tau + 1} a_i(\tau), \tag{2.15}$$

где $a_i(\tau) = 1$, если субпопуляция S_i на итерации $(t - \tau)$ включала в себя лучшего среди всех субпопуляций индивида, и $a_i(\tau) = 0$ в противном случае. Заметим, что в сумме (2.15) вес величины $a_i(\tau)$, определяемый отношением $\frac{t_a - \tau}{\tau + 1}$, убывает с ростом τ, так что при $\tau = 0$ (текущая итерация) этот вес равен t_a, а при $\tau = t_a - 1$ (первая итерация данного интервала адаптации) тот же вес равен $\frac{1}{t_a}$.

Проверка выполнения условий останова. В случае выполнения одного из условий останова, в качестве решения задачи принимаем наилучшее найденное решение и прекращаем вычисления. В противном случае переходим к следующему шагу.

В качестве условий останова могут быть использованы стандартные для генетического алгоритма условия, такие как достижение заданного числа итераций или стагнация вычислений. Новым по сравнению с генетическим алгоритмом является условие исчерпания ресурса $n_f^{\max}$.

Перераспределение ресурсов производим путем сокращения размера каждой из проигравших субпопуляций на величину n_p и увеличения размера победившей субпопуляции на число, равное сумме потерь проигравших (так что общий размер популяции остается неизменным). Если при этом размер проигравшей субпопуляции оказывается меньшим величины $|S|_{\min}$, то принимаем его равным этот величине. Таким образом, если $|S_{i_j}|$ – размер

субпопуляции S_{i_j}, проигравшей на текущем интервале адаптации, то на следующем интервале размер этой субпопуляции будет равен

$$\left|S'_{i_j}\right| = \max\left(n_p\left|S_{i_j}\right|, |S|_{\min}\right),\ i_j \in [1:|\boldsymbol{S}|]$$

и в популяцию S'_{i_j} войдут $\left|S'_{i_j}\right|$ лучших индивидов популяции S_{i_j}.

Наряду с перераспределением ресурсов процесс коэволюции может включать в себя миграцию через определенное число поколений лучших индивидов в другие субпопуляции.

Известны результаты исследований, показывающие, что, даже после значительных затрат на настройку, средняя эффективность генетического коэволюционного алгоритма превосходит среднюю эффективность базового генетического алгоритма.

Как отмечалось выше, одной из основных проблем эффективного использования популяционных алгоритмов вообще, и генетических алгоритмов в частности, является проблема обоснованного выбора значений многочисленных свободных параметров этих алгоритмов. Коэволюционный подход можно считать одним из перспективных способов решения этой проблемы. Действительно, в этом случае вместо того, чтобы предварительно искать оптимальные (для данной задачи оптимизации) значения указанных параметров, можно сформировать достаточно представительный набор субпопуляционных алгоритмов, имеющих различные их значения. Коэволюционный алгоритм в процессе самоадаптации выберет для данной задачи лучший из этих алгоритмов. Детально указанный аспект генетического коэволюционного алгоритма и коэволюционных алгоритмов в целом обсуждается в гл. 7.

Заслуживает внимания использование генетического коэволюционного алгоритма для решения задачи условной оптимизации с ограничениями типа неравенств (задачи нелинейного программирования) на основе отыскания седловой точки соответствующей функции Лагранжа.

Рассмотрим задачу нелинейного программирования (2.1) с ограничениями типа неравенств вида $g_i(X) \geq 0$, $i \in [1:|G|]$, в которой функция $f(X)$ и все функции $g_i(X)$ всюду в области D являются дифференцируемыми. Как известно, *функцию Лагранжа* для такой задачи определяет формула

$$L(X, \Lambda) = f(X) + \sum_{i=1}^{|G|} \lambda_i g_i(X) = f(X) + \Lambda^{\mathrm{T}} G(X),$$

где $\Lambda = (\lambda_i, i \in [1:|G|])$ – $(|G| \times 1)$-*вектор множителей Лагранжа*.

Из известной *теоремы Куна – Такера* следует, что в принятых соглашениях точка X^* локального минимума рассматриваемой задачи совпадает с седловой точкой функции Лагранжа, т. е.

$$\nabla_X L(X^*, \lambda) = 0\ ,\ \nabla_\Lambda L(X^*, \Lambda) = 0. \qquad (2.16)$$

При этом точка X^* является точкой минимума функции Лагранжа по компонентам вектора X и точкой максимума – по компонентам вектора Λ.

На основе этого факта разработано большое число алгоритмов решения рассматриваемой задачи условной оптимизации, использующих безусловную минимизацию функции Лагранжа по переменным X (при фиксированном векторе Λ) и такую же максимизацию функции Лагранжа по переменным Λ (при фиксированном векторе X).

Данный подход естественным образом может быть реализован с помощью генетического коэволюционного алгоритма (и других коэволюционных алгоритмов). Для этого создаем популяцию S_1, особи которой соответствуют вектору варьируемых переменных X, а также популяцию S_2, соответствующую вектору множителей Лагранжа Λ. Функцию пригодности строим на основе функции Лагранжа $L(X, \Lambda)$, где значения компонент вектора X берутся из популяции S_1, а значения компонент вектора Λ – из популяции S_2. Очевидно, что при этом значение функции пригодности индивида из популяции S_1 зависит от значения функции пригодности индивида из популяции S_2, и наоборот.

2.13. ПРИМЕР ПРИМЕНЕНИЯ ГЕНЕТИЧЕСКОГО АЛГОРИТМА

Рассмотрим задачу управления спуском космического аппарата в атмосфере Земли. Начало системы координат $0x_1x_2x_3$, в которой рассматриваем движение аппарата, находится в центре Земли. Ось $0x_1$ направлена на исходное положение аппарата, ось $0x_3$ – в сторону его движения и перпендикулярна оси $0x_1$, ось $0x_2$ образует с указанными осями правую тройку.

Математическую модель объекта управления представляет система обыкновенных дифференциальных уравнений (ОДУ):

$$\begin{aligned}
&\frac{dx_1}{d\tau} = x_2,\\
&\frac{dx_2}{d\tau} = -\frac{\mu}{r^3}x_1 - (1-u)qx_2,\\
&\frac{dx_3}{d\tau} = x_4,\\
&\frac{dx_4}{d\tau} = -\frac{\mu}{r^3}x_3 - (1-u)qx_4
\end{aligned} \tag{2.17}$$

при заданных начальных условиях $x_i(0) = x_i^0$, $i \in [1:4]$. Здесь приняты следующие обозначения:

x_1, x_3 – координаты центра масс космического аппарата;

x_2, x_4 – компоненты его скорости;

u – управление;

$r = \sqrt{x_1^2 + x_3^2}$ – величина радиус-вектора аппарата;

μ – гравитационная постоянная Земли;

$q = ce^{\frac{h-x_1}{\eta}} v$ – составляющая аэродинамической силы;

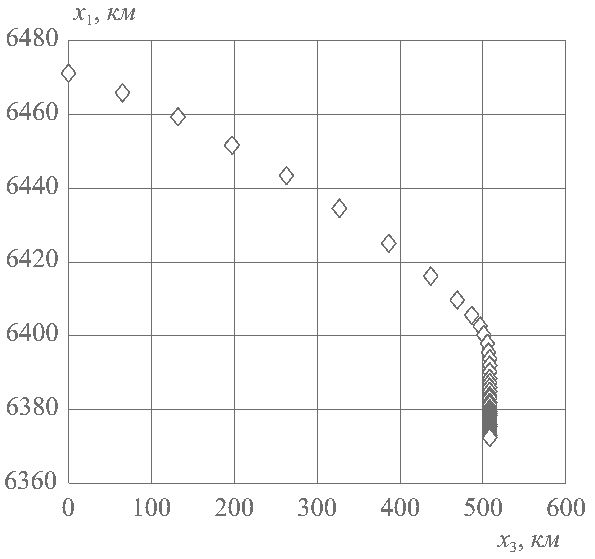


Рис. 2.12. Траектория спуска космического аппарата при постоянном управлении $u(\tau) = -0,2$

$v = \sqrt{x_2^2 + x_4^2}$ – величина скорости аппарата;

c – его аэродинамическая характеристика;

h – высота атмосферы;

η – согласующий коэффициент модели атмосферы.

Определены функционалы качества управления

$$f_1(u) = \max_{\tau \in [0;\hat{\tau}]} \frac{(x_1^2(\tau) + x_3^2(\tau)\sqrt{x_2^2(\tau) + x_4^2(\tau)}}{g_0 r_З^2} \to \min_{u \in D_u}, \tag{2.18}$$

$$f_2(u) = x_3(\hat{\tau}) - l \to \min_{u \in D_u}, \tag{2.19}$$

имеющие смысл максимальной перегрузки и отклонения от заданной точки на поверхности Земли соответственно. Здесь приняты следующие обозначения:

$r_З = 6371$ км – радиус Земли;

$g_0 = 9,81$ м/с^2 – ускорение свободного падения;

$\hat{\tau}$ – длительность полета;

l – координата по оси $0x_3$ заданной точки на поверхности Земли;

$D_u = \{u(\tau) \,|\, |u(\tau)| \le u^+\}$ – множество допустимых управлений;

u^+ – заданная положительная константа.

Задача состоит в определении допустимого управления $u^*(\tau) \in D_u$, удовлетворяющего на решениях системы (2.17) условиям (2.18), (2.19).

Положим, что имеют место начальные условия $x_1(0) = 6471$ км, $x_3(0) = 0$, $x_2(0) = 0,1$ км/с, $x_4(0) = 7$ км/с. Характер соответствующей траектории спуска космического аппарата при фиксированном управлении $u(\tau) = const = -0,2$ иллюстрирует рис. 2.12. Для интегрирования системы (2.17) использован метод Рунге-Кутты четвертого порядка с шагом, равным одной секунде.

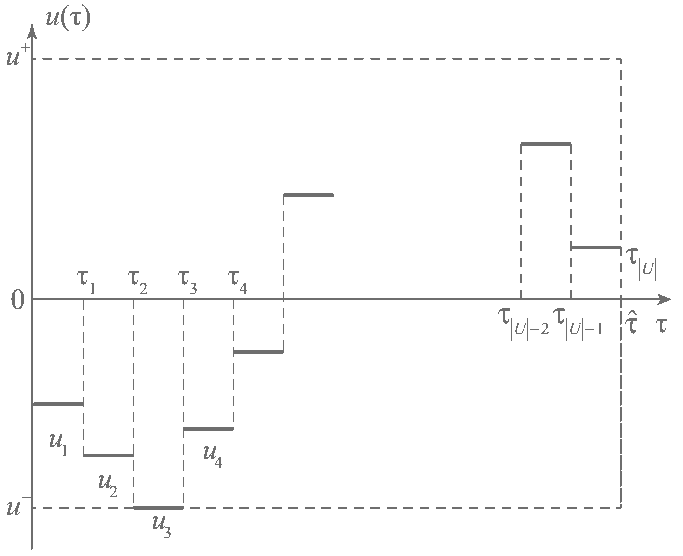


Рис. 2.13. Кусочно-постоянная аппроксимация управления

Алгоритм решения задачи

Задача (2.17) – (2.19) представляет собой задачу оптимального управления. При всех хорошо известных недостатках метода решения данной задачи, основанного на сведении ее к задаче нелинейного программирования, используем именно этот метод.

Покроем интервал $[0;\hat{\tau}]$ равномерной сеткой с узлами τ_i, $i \in [0:|U|]$ и будем искать оптимальное управление $u^*(\tau)$ в классе кусочно-постоянных функций (рис. 2.13). Обозначим $U = (u_1, u_2, ..., u_{|U|})$ – $(|U| \times 1)$-вектор, где $u_i = u(\tau_i)$. Тогда задачи (2.18), (2.19) примут вид

$$\min_{U \in D_U} f_1(U) = \min_{U \in D_U} \max_{\tau \in [0;\hat{\tau}]} \frac{(x_1^2(\tau) + x_3^2(\tau)\sqrt{x_2^2(\tau) + x_4^2(\tau)}}{g_0 r_3^2}, \tag{2.20}$$

$$\min_{U \in D_U} f_2(U) = \min_{U \in D_U} (x_3(\hat{\tau}) - l), \tag{2.21}$$

где $D_U = \{u_i \mid |u_i| \le u^+, i \in [1:|U|]\}$.

Таким образом, имеем две задачи глобальной условной оптимизации с целевыми функциями $f_1(U)$, $f_2(U)$, вектором варьируемых параметров U и множеством допустимых значений вектора варьируемых параметров D_U.

Для решения указанных задач используем генетический алгоритм с вещественным кодированием особей. Хромосома при этом представляет собой совокупность $|U|$ генов (рис. 2.14), в которой ген H_i соответствует компоненте u_i вектора варьируемых параметров U; $i \in [1:|U|]$.

Используем оператор случайной мутации с высокой вероятностью мутации $\xi_m = 0,95$. В качестве кроссовера применим расширенный линейный кроссовер. Селекцию реализуем с помощью метода отбора на основе элитизма. Размер популяции $|S|$ варьируем в пределах от 20 до 1000, а число поколений $\hat{t}$ – от 20 до 500.

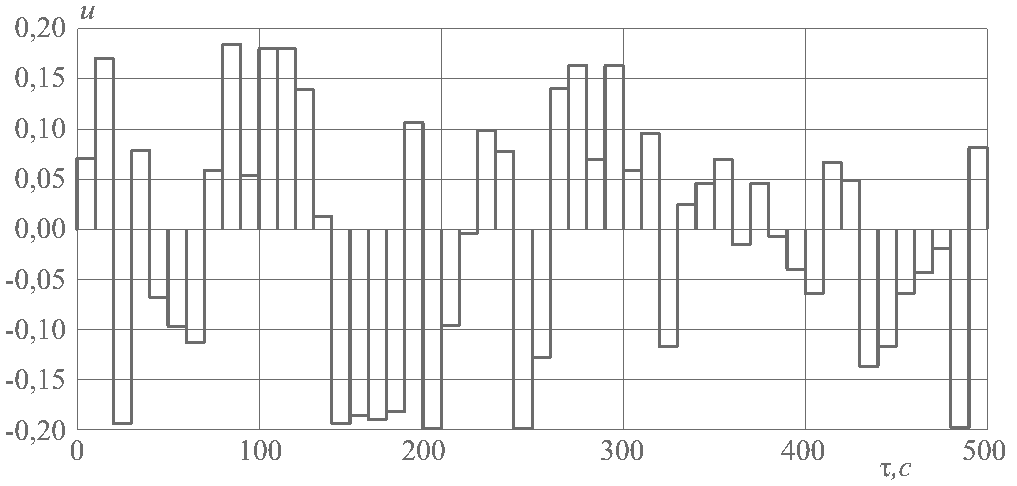


Рис. 2.14. Один из векторов начальной популяции: $u^{+} = 0,2$; $|U| = 50$

Результаты

Рассмотрим прежде задачу минимизации максимальной перегрузки (2.17), (2.18). Для случая $u^{+} = 0,4$ характер полученного решения иллюстрирует рис. 2.15. Поскольку перегрузка обусловлена действием аэродинамической силы, пропорциональной квадрату скорости космического аппарата, после 150 –200 секунд полета, когда начальная скорость аппарата в значительной степени уже погашена, величина перегрузки становится несущественной. Поэтому время на рис. 2.15 ограничено именно этим интервалом. Максимальное значение перегрузки составляет 8,37*g*.

Перейдем к рассмотрению задачи (2.17), (2.19) о минимизации отклонения космического аппарата от заданной точки на поверхности Земли. Для того же

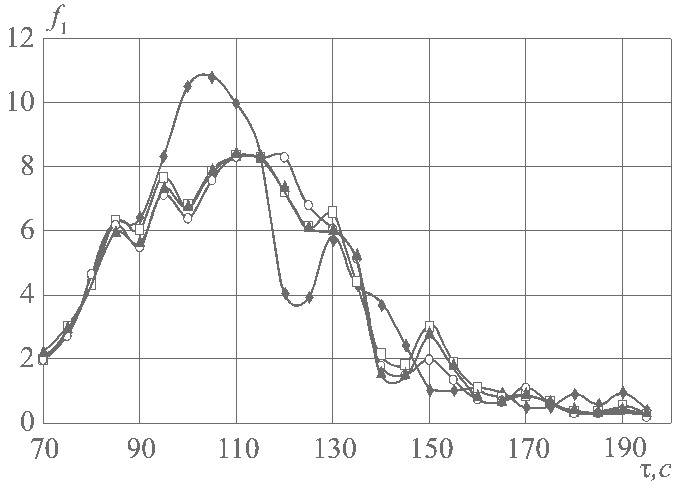


Рис. 2.15. Характер изменения оптимальной перегрузки во времени: $u^{+} = 0,4$: ♦ – одно поколение; □ – 100 поколений; ○ – 500 поколений; ▲ – 900 поколений

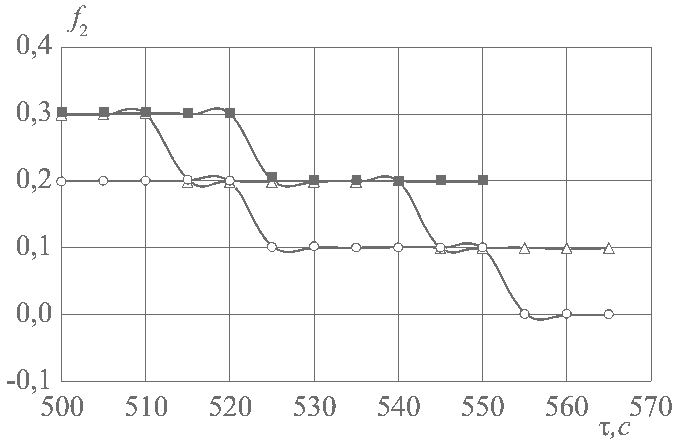


Рис. 2.16. Характер изменения минимального отклонения космического аппарата от заданной точки на поверхности Земли ($u^+ = 0,4$): ■ – одно поколение; △ – 30 поколений; ○ – 70 поколений

случая $u^+ = 0,4$ характер полученного решения иллюстрирует рис. 2.16, который показывает, что найденное оптимальное управление гарантирует точное попадание аппарата в заданную точку на поверхности Земли.

Вопросы для самопроверки

1. Представьте схему канонического генетического алгоритма.

2. Дайте определение монохромосомного, мультихромосомного, адаптивного, символьного и вещественного кодирования хромосом.

3. Приведите примеры бинарных операторов мутации, использующих инверсию одного или нескольких битов родительской особи, а также операторов, основанных на перестановке битов этой особи.

4. Назовите отличительные признаки следующих вещественных операторов мутации: оператор случайной мутации, гауссов оператор мутации, арифметический оператор вещественного сдвига, геометрический оператор вещественного сдвига, *BGA*-оператор мутации, степенной оператор мутации, неравномерный мутатор Михалевича, динамический оператор мутации.

5. Перечислите основные двухродительские и многородительские бинарные кроссоверы.

6. Дайте определение следующих вещественных кроссоверов, которые на основе двух родительских особей продуцируют одну особь-потомка: плоский кроссовер, эвристический кроссовер, смешанный кроссовер, расширенный линейный кроссовер, дискретный кроссовер.

7. Назовите основные особенности следующих вещественных кроссоверов с двумя особями-потомками: простейший кроссовер, арифметический кроссовер, геометрический кроссовер, нечеткий кроссовер; кроссовер, имитирующий бинарный кроссовер.

8. Приведите пример вещественного кроссовера с тремя особями-потомками.

9. Определите следующие методы управления популяцией, учитывающие только приспособленность особей: метод рулетки, метод пропорционального отбора с остатком, метод стохастического универсального отбора, метод турнирного отбора, метод рангового отбора, метод отбора на основе элитизма.

10. Приведите пример метода управления популяцией, учитывающего как приспособленность особей, так и близость их генотипов.

11. Дайте определение функции отбора. Назовите основные классы функций отбора.

12. Приведите примеры следующих методов селекции: случайный метод селекции, не учитывающий приспособленность и генотип особей; метод, использующий только приспособленность особей; метод, основанный на генотипе особей.

13. Какие методы масштабирования фитнес-функции вы знаете?

14. Дайте определения следующих методов нишевания: метод разделяющей (сепарирующей) функции, метод детерминированного сжатия, метод последовательного нишевания.

15. Представьте схемы следующих типовых генетических алгоритмов: канонический генетический алгоритм, генетический алгоритм *Genitor*, генетический алгоритм *CHC*, генетический микроалгоритм.

16. Приведите определения основных понятий теории шим. Сформулируйте основную и модифицированную теоремы шим.

17. Назовите основные особенности алгоритма эволюционной стратегии. Дайте схему канонического алгоритма эволюционной стратегии.

18. Перечислите основные операторы алгоритма эволюционного программирования. Представьте схему этого алгоритма.

19. Дайте схему алгоритма дифференциальной эволюции.

20. Представьте схему генетического коэволюционного алгоритма и раскройте суть его основных операторов.

Глава 3

АЛГОРИТМЫ РОЯ ЧАСТИЦ, КОЛОНИИ МУРАВЬЕВ И ПЧЕЛИНОГО РОЯ

Кроме рассмотренных в предыдущей главе эволюционных алгоритмов, из числа популяционных алгоритмов в настоящее время наиболее развитыми являются инспирированные живой природой алгоритмы роя частиц, муравьиной колонии и пчелиного роя. В данной главе эти алгоритмы рассмотрены в указанной последовательности.

3.1. ОПТИМИЗАЦИЯ РОЕМ ЧАСТИЦ

В основу *алгоритма оптимизации роем частиц* (*Particle Swarm Optimization*, *PSO*) положена социально-психологическая поведенческая модель толпы. Развитие алгоритма инспирировали такие задачи, как моделирование поведения птиц в стае и рыб в косяке. Целью было обнаружить базовые принципы, благодаря которым, например, птицы в стае ведут себя удивительно синхронно, меняя как по команде направления своего движения, так что стая движется как единое целое. К современному времени концепция алгоритма роя частиц развилась в высокоэффективный алгоритм оптимизации, часто составляющий конкуренцию лучшим модификациям генетического алгоритма.

Существует значительное число алгоритмов роя частиц. В каноническом алгоритме, предложенном в 1995 г. Кеннеди (*J. Kennedy*) и Эберхартом (*R. Eberhart*), при определении следующего положения частицы учитывается информация о наилучшей частице из числа ее «соседей», а также информация о данной частице на той итерации алгоритма, на которой этой частице соответствовало наилучшее значение фитнес-функции. Известно большое число модификаций данного алгоритма. Например, модификация *FIPS* (см. 3.1.2) при определении следующего положения частицы учитывает значения фитнес-функции, соответствующие всем частицам популяции.

Ниже представлены канонический алгоритм *PSO*, наиболее известные модификации канонического алгоритма, определено понятие топологии соседства частиц. Кроме того, рассмотрены основные топологии, используемые в вычислительной практике, дан обзор алгоритмов *PSO* с динамической топологией соседства частиц, представлен гибридный алгоритм на основе роя частиц и имитации отжига. В завершении параграфа приведен пример использования алгоритма *PSO*.

3.1.1. Канонический алгоритм роя частиц

Рассмотрим задачу многомерной глобальной безусловной минимизации. В каноническом *алгоритме PSO* координаты частицы s_i на итерации t определяет вектор $X_i = (x_{i,1}, x_{i,2}, ..., x_{i,|X|})^T$, а на итерации $(t+1)$ – вектор $X'_i = (x'_{i,1}, x'_{i,2}, ..., x'_{i,|X|})^T$. Начальные координаты частицы s_i равны $X_i(0)$; $i \in [1:|S|]$.

Итерации в алгоритме *PSO* выполняем по схеме

$$X_i' = X_i + V_i, \tag{3.1}$$

$$V_i = b_I V_i^- + U_{|X|}(0;b_C) \otimes \left(X_i^* - X_i\right) + U_{|X|}(0;b_S) \otimes \left(X_i^{**} - X_i\right). \tag{3.2}$$

Здесь использованы следующие обозначения: $\otimes$ – символ прямого произведения векторов; b_I, b_C, b_S – свободные параметры алгоритма; $V_i = V_i(t) = (v_{i,1}, v_{i,2}, ..., v_{i,|X|})^T$ – $(|X| \times 1)$-вектор приращения координат частицы; $V_i^- = V_i(t-1)$; X_i^* – вектор координат частицы s_i, соответствующий ее наилучшему значению фитнес-функции за время поиска $[0:\hat{t}]$, то есть

$$\min_{t \in [0:\hat{t}]} \varphi(X_i(t)) = \varphi(X_i^*(t)); \tag{3.3}$$

X_i^{**} – вектор координат соседней с данной частицы с наилучшим за то же время значением приспособленности, то есть

$$\min_{j \in N_i} \varphi(X_j^*) = \varphi(X_i^{**}), \tag{3.4}$$

где N_i – множество номеров частиц, являющихся «соседями» данной частицы s_i.

Вектор V_i имеет смысл шага, который частица s_i совершает при переходе из текущего положения X_i в положение X_i'. Часто этот вектор называют *скоростью частицы*. В процессе итераций вектор $X_i^* = (x_{i,1}^*, ..., x_{i,|X|}^*)$ образует так называемый *собственный путь* (*private guide*) частицы s_i, а вектор $X_i^{**} = (x_{i,1}^{**}, ..., x_{i,|X|}^{**})$ – *локальный путь* (*local guide*) этой частицы.

Пересчет координат частиц по формулам (3.1), (3.2) может происходить по *синхронной схеме*, когда обновление координат частиц выполняется только после определения текущих приращений V_i координат всех $|S|$ частиц, или по *асинхронной схеме*, когда расчет новых координат частиц производится до завершения указанных вычислений.

Первое слагаемое в формуле (3.2) обозначаем V_i^I. Это слагаемое получило название *инерционной компоненты*, выполняющей функции памяти частицы о ее перемещении на предыдущей итерации. Компонента предотвращает сильные скачкообразные изменения направления движения частицы.

Второе слагаемое в той же формуле обозначаем V_i^C. Слагаемое называют *когнитивной компонентой*, поскольку оно формализует тенденцию частицы s_i вернуться из текущего положения X_i в положение X_i^*, в котором она имела на предыдущих итерациях наилучшее значение фитнес-функции. Эта компонента представляет собой случайный вектор, который, при равных значениях составляющих вектора $U_{|X|}(0;b_C)$, направлен из точки X_i в точку X_i^* и длина которого пропорциональна евклидову расстоянию между этими точками. Данная компонента реализует функцию памяти частицы о своем наиболее оптимальном положении.

Третье слагаемое формулы (3.2) обозначаем V_i^S. Данное слагаемое именуют *социальной компонентой*. Компонента отражает стремление частицы s_i

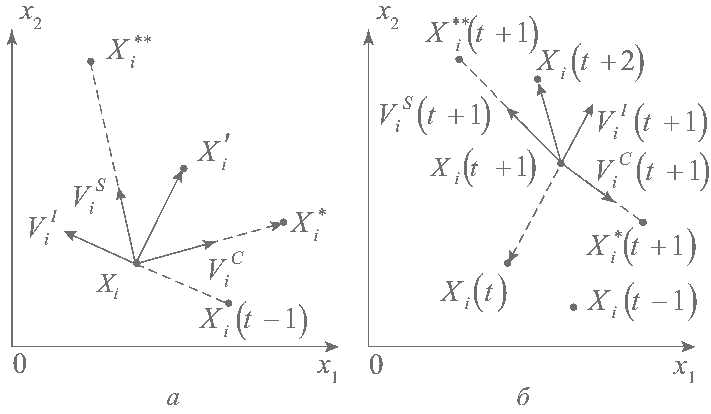


Рис. 3.1. К схеме алгоритма *PSO*, $|X| = 2$: *а* – поколение $t+1$; *б* – поколение $t+2$

переместиться в направлении своей наиболее успешной соседней частицы s_i^{**}. Аналогично когнитивной компоненте социальная компонента представляет собой случайный вектор с длиной, пропорциональной расстоянию между точками X_i^{**}, X_i, направленный при равных значениях составляющих вектора $U_{|X|}(0;b_S)$ из точки X_i в точку X_i^{**}. Можно сказать, что социальная компонента представляет собой реализацию памяти частицы об оптимальном положении ее наиболее успешной соседней частицы.

Суммарный эффект взаимодействия указанных компонентов для двумерного вектора варьируемых параметров иллюстрирует рис. 3.1, на котором принято, что все компоненты случайных векторов $U_{|X|}(0;b_C)$, $U_{|X|}(0;b_S)$ одинаковы, $X_i^*(t+1) = X_i^*(t)$, $X_i^{**}(t+1) = X_i^{**}(t)$.

Свободный параметр b_I определяет вес «инерционных» свойств частицы (при $b_I < 1$ ее движение, очевидно, замедляется). Рекомендуемое значение параметра $b_I = 0{,}7298$. Известно, что в процессе оптимизации может быть эффективным постепенное уменьшение коэффициента b_I от 0,9 до 0,4. При этом большие значения параметра обеспечивают широкий обзор пространства поиска, а малые – точную локализацию минимума фитнес-функции.

Значения параметров b_C, b_S определяют относительные веса когнитивной и социальной компонент. В случае нулевых значений этих параметров траектория частицы представляет собой прямую, уходящую в бесконечность. При $b_C > 0$, $b_S = 0$ частицы пытаются достичь минимума фитнес-функции независимо друг от друга. Успех решения задачи оптимизации алгоритмом *PSO* обеспечивает координация движения частиц, которую определяет значение параметра b_S. При малых значениях параметров b_C, b_S частицы перемещаются по гладким траекториям. С увеличением значений этих параметров движения частиц становятся все более стохастическими. Рекомендуемые значения параметров $b_C = b_S = 1{,}49618$.

Как и в любом популяционном алгоритме поисковой оптимизации, в алгоритме *PSO* с увеличением размера популяции $|S|$ повышается вероятность локализации глобального экстремума фитнес-функции, поскольку такая популяция способна более равномерно покрывать пространство поиска. С другой стороны,

опять же аналогично другим популяционным алгоритмам, при этом повышается вычислительная сложность каждой итерации (хотя, может быть, уменьшается их общее число). Исследования показывают, что алгоритм *PSO* во многих случаях обеспечивает высокую эффективность поиска даже при использовании малочисленной популяции из 10–30 частиц. Общей рекомендацией является подбор оптимального размера популяции для каждой задачи в отдельности.

Оптимальное число итераций $\hat{t}$ также является задачезависимым. При малом числе итераций работа алгоритма может закончиться преждевременно, до достижения глобального экстремума, а при большом – могут быть безосновательно увеличены вычислительные затраты.

Схема алгоритма *PSO* имеет следующий вид.

1) Задаем значения свободных параметров алгоритма и инициализируем популяцию. Полагаем счетчик числа поколений $t=1$.

2) Для каждой из частиц популяции s_i, $i \in [1:|S|]$ по формуле (3.3) находим лучшую локальную позицию X_i^*, а по формуле (3.4) – глобально лучшую позицию X_i^{**}.

3) По формулам (3.1), (3.2) находим новые позиции всех частиц популяции X_i'.

4) Проверяем выполнение условия окончания итераций. Если это условие выполнено, завершаем итерации, в противном случае полагаем $t=t+1$ и переходим к шагу 2.

Инициализация популяции заключается в задании начальных позиций $X_i(0)$ и скоростей $V_i(0)$ частиц. Обычно точки $X_i(0)$ принимают случайными, равномерно распределенными в гиперпараллелепипеде

$$\Pi = \{X \mid x_j^- \le x_j \le x_j^+,\ j \in [1:|X|]\},$$

то есть полагают $x_{i,j}(0) = U_1(x_j^-; x_j^+)$. Начальные скорости частиц также инициализируем случайными значениями.

В качестве условий окончания итераций используют типичные для популяционных алгоритмов оптимизации условия: достижение максимально допустимого числа итераций $\hat{t}$; достижение решения, удовлетворяющего ЛПР; стагнация итерационного процесса, то есть отсутствие улучшений фитнес-функции на протяжении δ_t итераций, где δ_t – свободный параметр алгоритма; равенство нулю с заданной точностью оценки первой производной фитнес-функции в окрестности глобально лучшей частицы либо всех частиц популяции.

Вместо формулы (3.2) часто используют ее вариант

$$V_i = b_r\left(V_i^- + \tilde{U} \otimes \left(Y_i - X_i\right)\right), \tag{3.5}$$

где

$$\tilde{U} = U_{|X|}(0;b_*) + U_{|X|}(0;b_{**}),$$

$$Y_i = \frac{(U_{|X|}(0;b_*) \otimes X_i^*) + (U_{|X|}(0;b_{**}) \otimes X_i^{**})}{\tilde{U}}.$$

Свободный параметр b_r в формуле (3.5) называют *ограничивающим фактором* (*restriction factor*). Рекомендуемые значения параметров b_*, b_{**} в этом

случае равны 2,05, а их оптимальные значения определяет, в конечном счете, класс целевой функции.

Для j-й компоненты вектора X_i' выражения (3.1), (3.2) в скалярной форме можно записать в виде

$$x_{i,j}' = x_{i,j} + b_l v_{i,j}^- + U_1(0; b_C)\left(x_{i,j}^* - x_{i,j}\right) + U_1(0; b_S)\left(x_{i,j}^{**} - x_{i,j}\right),\ j \in [1:|X|]. \quad (3.6)$$

В терминах генетического алгоритма канонический алгоритм роя частиц можно определить следующим образом. Алгоритм использует только вещественное кодирование особей. В популяцию S' переходят все особи текущей популяции S, то есть отбор и селекция особей не производятся. Поэтому можно сказать, что, в отличие от генетического алгоритма, особи в алгоритме *PSO* «бессмертны». Итерационную схему (3.2) можно интерпретировать как схему специфического оператора кроссовера, построенного на основе линейной комбинации векторов X_i, V_i^-, X_i^*, X_i^{**}.

Пример 3.1. Рассмотрим в качестве примера решение задачи глобальной безусловной оптимизации для функций Шекеля, Розенброка и Растригина (Приложение Б) с помощью алгоритма *PSO*. Примем, что во всех случаях фитнес-функция совпадает с целевой функцией.

Поскольку алгоритм *PSO* является стохастическим, его эффективность может сильно зависеть от начальной популяции. Поэтому используем мультистарт с числом стартов $\hat{n} = 80$ и последующее осреднение результатов. В качестве условия окончания итераций используем условие стагнации вычислительного процесса в течение $\delta_t = 10$ итераций. Число итераций, при котором начинается стагнация, обозначаем t_{end}.

Некоторые результаты вычислительных экспериментов представлены на рис. 3.2 – 3.4.

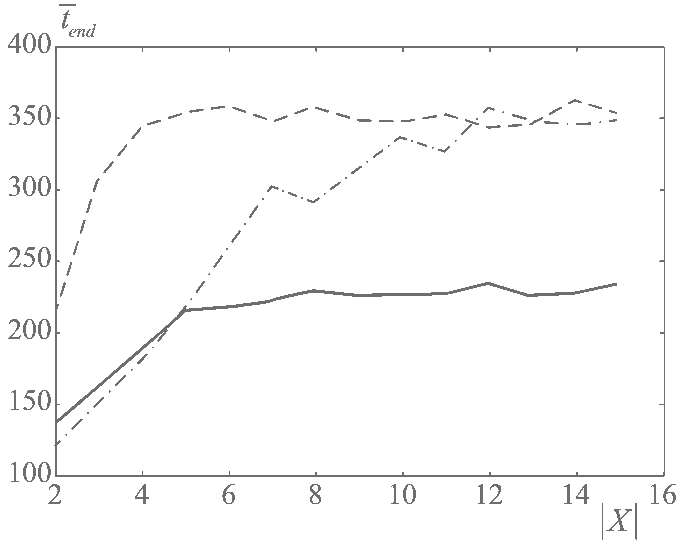


Рис. 3.2. Среднее число итераций $\overline{t}_{end}$ в функции размерности пространства поиска $|X|$: -·-·- функция Шекеля, —— функция Розенброка, – – – функция Растригина

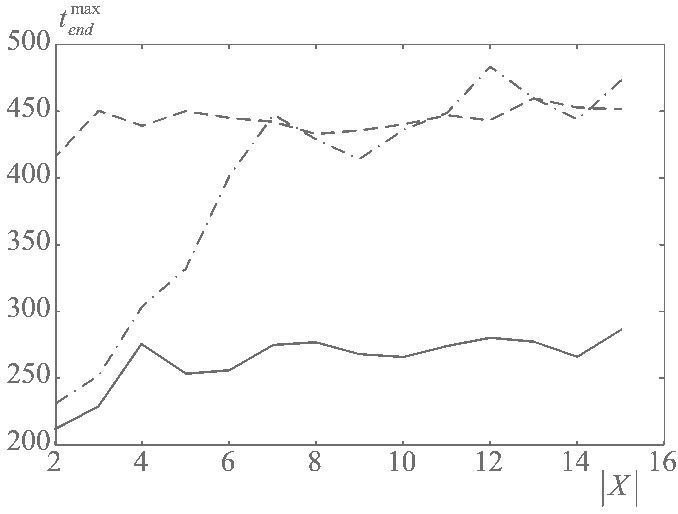


Рис. 3.3. Максимальное число итераций t_{end}^{max} в функции размерности пространства поиска $|X|$: -·–·- функция Шекеля, —— функция Розенброка, – – – функция Растригина

Прежде всего, рис. 3.2 – 3.4 показывают, что алгоритм *PSO* обеспечивает локализацию глобального минимума всех указанных функций. Число итераций t_{end}, которое для этого потребовалось, сильно зависит от типа функции и размерности ее вектора варьируемых параметров X. Так, из рис. 3.2 следует, что величина $\overline{t}_{end}$ для одноэкстремальной (но овражной) функции Розенброка может отличаться от такой же величины для многоэкстремальной функции Растригина более чем в два раза. Из того же рис. 3.2 следует, что для функции

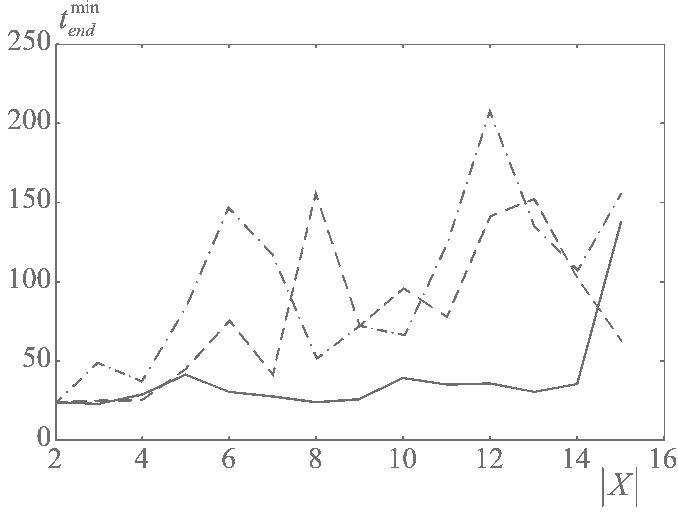


Рис. 3.4. Минимальное число итераций t_{end}^{min} в функции размерности пространства поиска $|X|$: -·–·- функция Шекеля, —— функция Розенброка, – – – функция Растригина

Шекеля с ростом размерности вектора X с 2 до 16 величина $\overline{t}_{end}$ увеличивается более чем в три раза. Отметим, что, начиная с некоторого значения величины $|X|$, имеет место стабилизация числа итераций $\overline{t}_{end}$, и для разных функций эта величина существенно различна.

На рис. 3.3, 3.4 величины $t_{end}^{\max}$, $t_{end}^{\min}$ представляют собой максимальное и минимальное значения величины t_{end}, достигнутые в $\hat{n}$ вычислительных экспериментах. По рисункам видно, что отношение $t_{end}^{\max}/t_{end}^{\min}$ может достигать пяти •

3.1.2. Модификации канонического алгоритма роя частиц

Известно большое число модификаций канонического алгоритма *PSO*. Рассмотрим наиболее характерные модификации, использующие стационарные топологии соседства частиц.

Притягательно-отталкивающий алгоритм роя частиц (*Attractive and Repulsive PSO, ARPSO*). Алгоритм *PSO* функционирует по принципу перемещения области поиска каждой из частиц s_i к ее текущему глобально оптимальному решению X_i^{**}, хотя искомый глобальный минимум фитнес-функции может находиться совсем в другой части пространства поиска. Данное обстоятельство является основной причиной преждевременной сходимости алгоритма *PSO*.

Алгоритм *ARPSO*, предложенный в работах Вестерстрома (*J. Vesterstrom*), в значительной мере свободен от указанного недостатка алгоритма *PSO*. Суть алгоритма заключается в последовательном чередовании *фазы притяжения* (*attraction phase*) и *фазы отталкивания* (*repulsion phase*). В фазе притяжения итерации выполняются по формулам (3.1), (3.2). В фазе отталкивания последняя формула приобретает вид

$$V_i = b_I V_i^- - U_{|X|}(0;b_C)\otimes\left(X_i^* - X_i\right) - U_{|X|}(0;b_S)\otimes\left(X_i^{**} - X_i\right),$$

то есть знаки когнитивной и социальной компонент меняются на противоположные.

Вестерстром предложил модификацию алгоритма *PSO*, которая использует еще одну фазу, получившую название *промежуточной фазы* (*between phase*) или *фазы положительного конфликта* (*positive conflict phase*). Промежуточная фаза представляет собой комбинацию фаз притяжения и отталкивания и определяется формулой

$$V_i = b_I V_i^- + U_{|X|}(0;b_C)\otimes\left(X_i^* - X_i\right) - U_{|X|}(0;b_S)\otimes\left(X_i^{**} - X_i\right).$$

Можно сказать, что в этой фазе частица s_i притягивается к своей наилучшей позиции X_i^* и отталкивается от наилучшей позиции соседних частиц X_i^{**}.

Алгоритм роя частиц с управлением скоростью частиц. В алгоритме *PSO* возможна ситуация, при которой скорости отдельных частиц экспоненциально возрастают с ростом номера поколения. В результате такие частицы могут покидать представляющую интерес часть пространства поиска, значительно уменьшая эффективность алгоритма. Алгоритм роя частиц с управлением скоростью частиц призван исключить возможность такой ситуации. С этой

целью предусматривается постоянный контроль скорости каждой из частиц и ее уменьшение при превышении некоторого порога.

Формально, алгоритм использует следующее правило управления скоростью частиц (см. формулу (3.2)):

$$v_{i,j} = \begin{cases} v_{i,j}, & v_{i,j} < v_j^{\max} \\ v_j^{\max}, & \text{иначе}, \end{cases}, \; i \in [1:|S|], \; j \in [1:|X|].$$

Здесь $v_j^{\max}$ – максимально допустимые значения соответствующих компонент вектора скорости (свободные параметры алгоритма).

Величины $v_j^{\max}$ оказывают сильное влияние на баланс между интенсивностью и широтой поиска. Большие значения этих величин увеличивают широту поиска и уменьшают его интенсивность. Малые значения $v_j^{\max}$ могут вынудить частицы остаться вблизи найденных ими локальных минимумов. Кроме того, такие значения $v_j^{\max}$ увеличивают число итераций, необходимых для достижения глобального экстремума фитнес-функции.

Величины $v_j^{\max}$ рекомендуют выбирать по правилу

$$v_j^{\max} = a\,(x_j^+ - x_j^-), \; j \in [1:|X|],$$

где x_j^-, x_j^+ – границы области поиска по j-му измерению, а $a \in (0;1)$ – коэффициент, определяющий часть этой области, доступную за одну итерацию алгоритма.

Отметим следующие обстоятельства. Рассмотренный механизм ограничивает скорость частиц по каждому из измерений в отдельности, так что имеется возможность управлять поиском в областях, далеких от гиперкуба. Этот же механизм может менять направления движения частиц. Данный эффект иллюстрирует рис. 3.5, на котором $\tilde{v}'_{i,2}$, $\tilde{X}'_i$ – скорректированные компонента скорости и положение частицы соответственно.

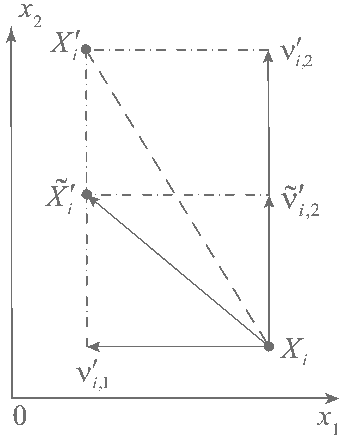


Рис. 3.5. Изменение направления движения частицы в алгоритме роя частиц с управлением их скоростью

Изменение направления движения частиц в результате ограничения их скорости может иметь как положительный, так и отрицательный эффекты. Положительный эффект заключается в возможном расширении поиска, а отрицательный – в снижении его интенсивности.

Очевидной является модификации алгоритма, при которой значения коэффициента a меняются в процессе поиска, уменьшаясь с ростом номера итерации. При этом на начальных итерациях алгоритма обеспечивается широта поиска, а на конечных итерациях – достаточная скорость сходимости.

Алгоритм роя частиц с полной информацией (*Fully Informed Particle Swarm*, *FIPS*) предложен в работах Мендеса (*R. Mendes*). В основе алгоритма *FIPS* лежит наблюдение, что поведение особи в стае определяет не только ее наиболее успешный сосед, но и вся совокупность соседей. Отличие алгоритма *FIPS* от ка-

нонического алгоритма *PSO* состоит в том, что в алгоритме *FIPS* учитывается влияние на данную частицу всех соседних с ней частиц. Очевидно, что в этом случае различные топологии соседства частиц должны существенно влиять на процесс оптимизации. В формуле (3.2) в данном случае исчезает необходимость в слагаемом, содержащем вектор X_i^{**}, то есть глобальный путь частицы s_i не существует.

В итерационной схеме алгоритма *FIPS* координаты частиц задает формула (3.1), где шаг V_i определяет формула (3.5), в которой схема вычисления вектора Y_i заменяется на схему

$$Y_i = \frac{\sum\limits_{j \in N_i} \left(\tilde{U}_j \otimes X_j^* \right)}{\sum\limits_{j \in N_i} \tilde{U}_j}, \ \tilde{U}_j = U_{|X|}\left(0; \frac{b_*}{|N_i|} \right),$$

где N_i – множество частиц, являющихся соседями частицы s_i.

Алгоритм *FIPS* обычно показывает лучшую сходимость, чем канонический алгоритм *PSO*. Основной проблемой алгоритма *FIPS* является сильное влияние на его эффективность используемой топологии соседства частиц. Особенностью алгоритма является тенденция его сходимости к центру начального положения частиц $X_i(0)$, $i \in [1:|S|]$. В этой связи рекомендуется использовать асимметричное начальное расположение.

Алгоритм FIPS с весами (*weighted FIPS*, *WFIPS*) использует модификацию канонического алгоритма *PSO*, которая заключается в добавлении весов в последнее слагаемое итерационной формулы (3.5). При соответствующем выборе весов такая модификация позволяет усилить влияние тех соседних частиц, у которых лучшие значения фитнес-функции. Итерационная формула алгоритма *FIPS* с весами имеет вид

$$V_i = b_r \left(V_i^- + \frac{\sum\limits_{j \in N_i} \left(w_{i,j}\, U_{|X|}(0; b_*) \otimes \left(X_j^* - X_i \right) \right)}{\sum\limits_{j \in N_i} w_j} \right),$$

где $w_{i,j}$ – скалярный весовой множитель, который определяет «силу» влияния на данную частицу j-й соседней частицы и в общем случае меняется в процессе итераций.

В качестве веса $w_{i,j}$ чаще всего берут величину, обратную значению фитнес-функции в точке X_j (напомним, что рассматриваем задачу минимизации):

$$w_{i,j} = \frac{b_w}{\varphi(X_j)}, \ \varphi(X_j) \neq 0. \tag{3.7}$$

Здесь b_w – свободный параметр, значение которого определяется классом функции $\varphi(X)$. Вес $w_{i,j}$ может быть вычислен также на основе, например, расстояния между частицами s_i, s_j, измеренного в той или иной метрике. Обычно веса масштабируют таким образом, чтобы имело место равенство

$$\sum_{j \in N_i} w_{i,j} = 1, \ i \in [1:|S|].$$

Ранжированный алгоритм FIPS (*Ranked FIPS*, *RFIPS*) сконструирован Джорданом (*J. Jordan*) с целью преодоления следующего недостатка алгоритма *FIPS* с весами, который может иметь место при использовании в качестве весов $w_{i,j}$ величин типа (3.7). В этом случае на поздних итерациях алгоритма, когда основные улучшения фитнес-функции уже произведены и все частицы находятся в окрестности минимума этой функции, значения фитнес-функции оказываются для всех частиц близкими. В результате значения весов $w_{i,j}$ для всех $j \in N_i$ оказываются примерно одинаковыми, так что эти веса теряют смысл. Как следствие, поиск замедляется или даже останавливается.

Идея ранжированного алгоритма *FIPS* состоит в замене весов соседних частиц $w_{i,j}$, $j \in N_i$ их рангами $r_{i,j}$, построенными на основе значений фитнес-функции по следующей схеме.

1) Для всех частиц s_{i_j}, $j \in N_i$, являющихся соседями частицы s_i, вычисляем значения их фитнес-функции $\varphi(X_{i_j}) = \varphi_{i_j}$ и упорядочиваем частицы в виде списка в порядке возрастания величины φ_{i_j}.

2) Первой частице в этом списке присваиваем ранг $r_{i,1} = 0,5$.

3) Каждой из последующих частиц списка присваиваем в два раза меньший ранг, так что $r_{i,j} = 0,5\ r_{i,j-1}$, $j \in [2 : |N_i|]$.

Легко видеть, что с ростом величины $|N_i|$ сумма всех так определенных рангов стремится к единице, то есть автоматически выполняется их нормировка.

Формула (3.2) в ранжированном алгоритме *FIPS* принимает вид

$$V_i = b_r\left(V_i^- + U_{|X|}(0;b_*)\otimes\left(X_i^* - X_i\right) + \sum_{j\in N_i} r_{i,j}U_{|X|}(0;b_{**})\otimes\left(X_i^{**} - X_i\right)\right).$$

Ранжированный алгоритм *FIPS* кроме устранения указанного недостатка алгоритма *FIPS* с весами обеспечивает также уменьшение отрицательного влияния на эффективность поиска топологии соседства частиц.

FDR алгоритм роя частиц (*Fitness-Distance Ratio based PSO*) разработан Вирамаханени (K. *Veeramachaneni*). Алгоритм основан на так называемом отношении *значение – расстояние* и отличается от канонического алгоритма *PSO* добавлением к инерциальной, когнитивной и социальной компонентам формулы (3.2) еще одной компоненты вида

$$\beta_L U_{|X|}(0;b_L)\otimes\left(X_i^0 - X_i\right). \tag{3.8}$$

Таким образом, в алгоритме *FDR* формула (3.2) приобретает вид

$$V_i = b_I V_i^- + \beta_C U_{|X|}(0;b_C)\otimes\left(X_i^* - X_i\right) + \beta_S U_{|X|}(0;b_S)\otimes\left(X_i^{**} - X_i\right) + \\ + \beta_L U_{|X|}(0;b_L)\otimes\left(X_i^0 - X_i\right).$$

Здесь $\beta_C, \beta_S, \beta_L$ – еще одна группа свободных параметров алгоритма, рекомендованные значения которых равны 1,0; 1;0; 2,0 соответственно; $X_i^0 = X_i^0(t)$ – вектор координат соседней с данной частицы s_j, $j \in N_i$, которой соответствует максимальное по $k \in N_i$ значение величины

$$d\left(X_i, X_k\right) = \frac{\varphi(X_i) - \varphi(X_k)}{\left\|X_i - X_k\right\|}.$$

Здесь *значению* соответствует числитель, а *расстоянию* – знаменатель; $\|\cdot\|$ – некоторая векторная норма (обычно – евклидова). Другими словами, частицу s_j определяют по правилу

$$\max_{k \in N_i} d(X_i, X_k) = d(X_i, X_j).$$

Легко видеть, что величина $d\left(X_i, X_k\right)$ представляет собой оценку константы Липшица фитнес-функции в окрестности точки X_i.

Экспериментально показано, что алгоритм малочувствителен к значениям параметров β_C, β_S, так что даже их обнуление не приводит к существенному уменьшению скорости сходимости алгоритма. Основной вклад в скорость сходимости алгоритма обеспечивает компонента (3.8).

3.1.3. Топологии соседства частиц

Эффективность алгоритмов роя частиц в значительной мере зависит от используемой *топологии соседства* частиц (*population topology*, *neighborhood topology*, *swarm topology*, *sociometry*). Топология соседства определяется неориентированным *графом соседства*, вершины которого соответствуют частицам популяции, а ребра связывают непосредственных соседей. Соседство частиц не зависит от геометрического расположения частиц в пространстве поиска, а определяется только их индексами (номерами) и сохраняется на протяжении всего поиска (если не используется та или иная динамическая топология соседства).

Такое определение соседства частиц, во-первых, не требует больших вычислительных затрат при определении соседних с данной частиц. В подходах, где соседство определяют на основе, например, евклидовой нормы разности координат частиц, на каждой итерации требуется $O(|S|^2)$ арифметических операций только для установления текущей топологии соседства. Во-вторых, в данном случае соседними могут считаться частицы, располагающиеся в далеких областях пространства поиска. Это обстоятельство диверсифицирует поиск, предупреждая раннюю сходимость итерационного процесса.

В вычислительной практике чаще всего используются следующие топологии соседства частиц: клика; кольцо; двумерный тор; кластерная топология.

Глобально оптимальную топологию типа *клика* (*gbest-topology*) определяет полносвязный граф, в котором соседями каждой из частиц s_i, $i \in [1:|S|]$ являются остальные $(|S|-1)$ частиц. Диаметр такого графа равен единице. Под диаметром неориентированного графа понимаем максимальное расстояние между любыми двумя его вершинами, равное минимальному числу ребер, соединяющих эти вершины. Глобально оптимальную топологию соседства частиц иногда называют социальной сетью.

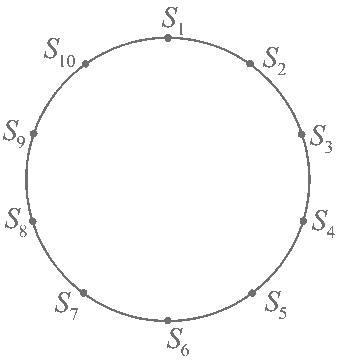


Рис. 3.6. Топология соседства частиц типа кольцо: $|S| = 10$

Локально оптимальной топологии типа *кольцо* (*lbest-topology*) соответствует граф соседства, в котором соседями каждой из частиц s_i являются две частицы. Например, в графе, приведенном на рис. 3.6, соседями частицы s_1 являются частицы s_2, s_{10}. Диаметр соответствующего графа равен $\sim |S|/2$.

В *топологии двумерный тор* (*топология фон Неймана*) соседями каждой из частиц s_i, $i \in [1:|S|]$ являются четыре частицы. Например, в графе, приведенном на рис. 3.7, который соответствует данной топологии, соседями частицы s_1 являются частицы s_2, s_5, s_6, s_{16}. Диаметр графа типа двумерный тор, построенного на основе $(n \times m)$-двумерной решетки, равен $\lceil n \rceil + \lceil m \rceil$, где $\lceil \cdot \rceil$ – символ ближайшего целого большего.

В *кластерной топологии* граф соседства частиц имеет в качестве вершин клики из c_{clicue} узлов в каждой. Клики числом n_{clicue} объединяются в полносвязный граф, ребра которого связаны с теми узлами в кликах, степени которых равны $(c_{clicue} - 1)$. Очевидно, что диаметр такого графа равен трем. Соседями каждой из частиц s_i, $i \in [1:|S|]$ являются c_{clicue} частиц. К примеру, в графе, приведенном на рис. 3.8, $n_{clicue} = 4$, $c_{clicue} = 5$, диаметр равен трем, соседями частицы s_3 являются пять частиц $s_1, s_2, s_4, s_5, s_{15}$.

Диаметр графа соседства частиц определяет скорость распространения информации в популяции. В популяции с топологией соседства типа клика лучшее значение фитнес-функции, достигнутое той или иной частицей, сразу становится известным всем остальным частицам популяции. Из числа рассмотренных топологий самую низкую скорость распространения информации обеспечивает топология соседства типа кольцо. Промежуточные значения скорости достигаются в популяциях, обладающих топологией соседства частиц типа двумерный тор или кластерной топологией.

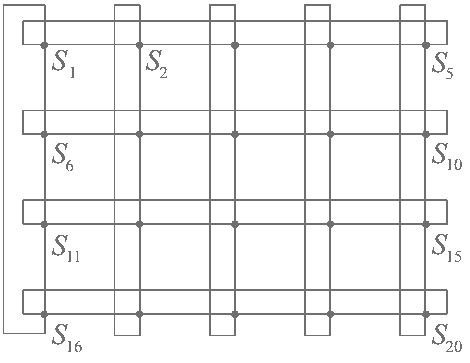


Рис. 3.7. Топология соседства частиц типа двумерный тор: $|S| = 20$

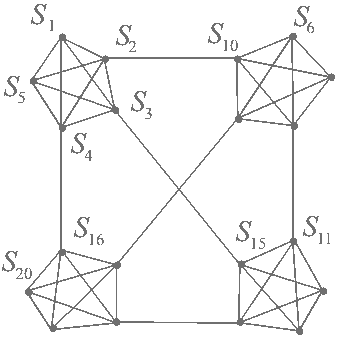


Рис. 3.8. Кластерная топология соседства частиц: $|S| = 20$

Уменьшение диаметра графа соседства частиц ускоряет сходимость итерационного процесса, то есть повышает интенсивность поиска. С другой стороны, небольшой диаметр этого графа уменьшает широту поиска и повышает вероятность преждевременной сходимости итераций к локальному минимуму фитнес-функции.

Резюмируя, можно сказать, что топологию соседства частиц типа кольцо целесообразно использовать в случае многомерных многоэкстремальных целевых функций. При оптимизации более простых целевых функций лучшим решением может быть использование топологии типа клика. Топология соседства типа двумерный тор и кластерная топология с точки зрения баланса интенсивность – широта поиска являются промежуточными между указанными топологиями. Их считают универсальными топологиями, обеспечивающими достаточно высокие как интенсивность поиска, так и его широту.

Пример 3.2. Проиллюстрируем влияние топологии соседства частиц на эффективность поиска численным примером. Рассмотрим в качестве целевой функции многоэкстремальную функцию Экли, глобальный минимум которой достигается в точке с координатами $(0,0,...,0)$ и равен нулю (Приложение Б).

Результаты, представленные на рис. 3.9, 3.10, получены для канонического алгоритма *PSO*. Число частиц в популяции во всех случаях равно $|S| = 50$. Аналогично примеру 3.1 использован мультистарт с числом прогонов, равным $\hat{n} = 100$. Величина $\overline{f}^*$ представляет собой усредненные по $\hat{n}$ прогонам минимальные достигнутые значения целевой функции.

Для количественной оценки влияния топологии соседства на эффективность алгоритма *PSO* используем величины $\overline{\varepsilon}_{2,1}(t)$, $\overline{\varepsilon}_{3,1}(t)$ – усредненные по $\hat{n}$

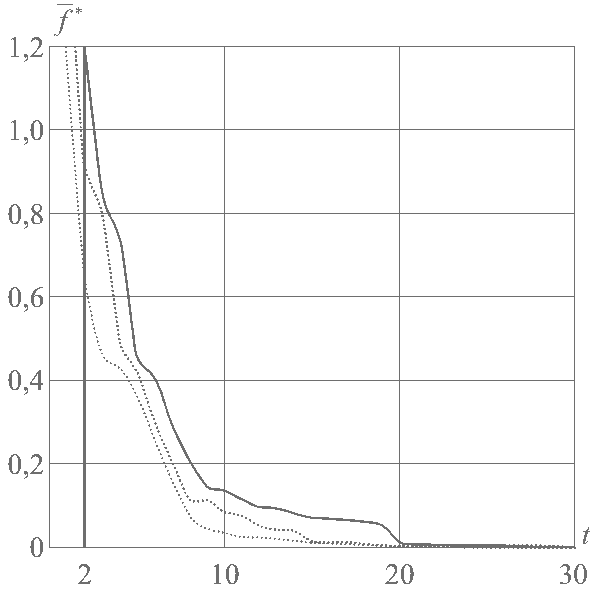


Рис. 3.9. К сходимости канонического алгоритма *PSO*, функция Экли; $|X| = 2$: —— – топология клика, ····· – топология двумерный тор, ····· – топология кольцо

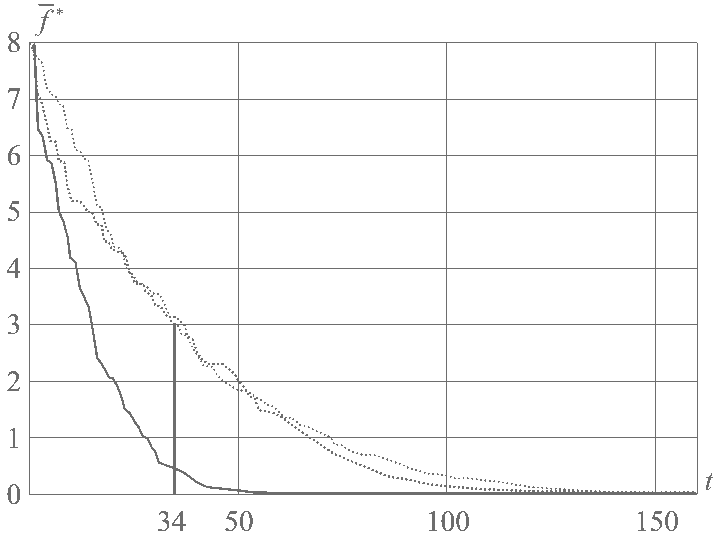


Рис. 3.10. К сходимости канонического алгоритма *PSO*, функция Экли; $|X| = 10$: —— – топология клика, ····· – топология двумерный тор, ····· – топология кольцо

отношения минимальных значений целевой функции на итерации t для топологий двумерный тор и клика, кольцо и клика соответственно:

$$\overline{\varepsilon}_{2,1}(t) = \frac{\overline{f}_2^*(t)}{\overline{f}_1^*(t)}, \ \overline{\varepsilon}_{3,1}(t) = \frac{\overline{f}_3^*(t)}{\overline{f}_1^*(t)}.$$

Рис. 3.9, 3.10 показывают, что, как утверждалось выше из общих соображений, скорость сходимости алгоритма *PSO* сильно зависит от используемой топологии соседства частиц. Так, в случае $|X| = 10$ (рис. 3.10) топология клика обеспечила сходимость алгоритма в среднем при числе итераций $\hat{t} = 60$. В тех же условиях топологии двумерный тор и кольцо завершили отыскание глобального минимума при $\hat{t} = 150$.

Сравнение рис. 3.9, 3.10 показывает, что разрыв в эффективностях топологий быстро увеличивается с ростом размерности целевой функции. Так, для двумерной функции (рис. 3.9) $\max_t \overline{\varepsilon}_{2,1}(t) = \overline{\varepsilon}_{3,1}(2) \approx 2{,}0$. В то же время для десятимерной функции (рис. 3.10) $\max_t \overline{\varepsilon}_{2,1}(t) = \overline{\varepsilon}_{3,1}(34) \approx 6{,}5$. Вертикальные отрезки указывают числа итераций, при которых достигаются указанные максимумы●

Аналогичное исследование, выполненное для функции Розенброка (Приложение Б), показывает, что топологии кольцо и двумерный тор обеспечивают примерно одинаковую скорость сходимости итераций и уступают топологии клика по критерию $\overline{\varepsilon}$ в два – три раза.

Как отмечалось выше, эффективность алгоритма *PSO* в значительной степени зависит также от размера популяции $|S|$. Эта зависимость может иметь различный характер для различных топологий соседства частиц.

Для топологии клика и популяции большого размера канонический алгоритм *PSO* может привести к ранней сходимости. В тех же условиях в алгоритме *FIPS*

большая популяция может охватывать несколько локальных минимумов фитнес-функции, и значительное число частиц могут колебаться между этими минимумами. Такие же колебания частиц возможны и в каноническом алгоритме *PSO*.

Топология кольцо более устойчива к большому размеру популяции, поскольку среди рассмотренных топологий является наименее связной и обеспечивает самую низкую скорость передачи информации.

Что касается эффективности алгоритма *PSO,* при больших популяциях в качестве универсальной топологии рекомендуют использовать топологию двумерный тор, которая обеспечивает достаточно высокую эффективность любой из рассмотренных выше модификаций алгоритма *PSO*.

3.1.4. Алгоритмы с динамической топологией соседства частиц

Канонический алгоритм *PSO*, а также рассмотренные выше его модификации основаны на *статической топологии соседства* частиц. Для этих алгоритмов типичной является проблема ранней сходимости. Причина в том, что частица, нашедшая какой-либо локальный минимум фитнес-функции, оказывает сильное и постоянное влияние на соседние частицы, стремясь локализовать их в окрестности того же локального минимума. В результате уменьшается вероятность того, что эти частицы могут исследовать другие возможные локальные минимумы.

Проблема ранней сходимости особенно остра для топологий, обеспечивающих сильную связанность частиц (клика, например), и менее остра для слабосвязанных популяций частиц (кольцо). Поэтому естественным образом возникает идея использовать на разных стадиях поиска различные топологии, или, другими словами, использовать *динамическую топологию соседства* частиц. Другим решением проблемы являются многопопуляционные (многороевые) алгоритмы, разные популяции в которых инициализируются независимо друг от друга и имеют, вообще говоря, различные топологии соседства и численности частиц (гл. 6).

Клубный алгоритм роя частиц (*Clubs-Based PSO*, *CBPSO*). Суть алгоритма *CBPSO* состоит в следующем. Частицы популяции образуют динамические группы, называемые клубами. Каждая частица может входить в несколько клубов. В процессе итераций частица может мигрировать между клубами. Соседними считаются частицы из одного клуба, то есть можно считать, что частицы в клубах имеют топологию соседства типа клика. Перемещения частиц определяются формулами (3.1), (3.2) канонического алгоритма *PSO*.

Число клубов m_i, в которые входит частица s_i, называют *уровнем вхождения* частицы в клубы (*membership level*). В начальный момент времени $t = 0$ каждую из частиц популяции S случайным образом распределяют по $m_i = n_{cl}$ клубам, где n_{cl} – так называемый *стандартный уровень вхождения*. При росте величины n_{cl} скорость сходимости алгоритма *CBPSO* возрастает. Обычно ограничиваются выбором уровня вхождения в пределах от четырех до 15.

На каждой итерации алгоритма *CBPSO* находят частицы s_{best} и s_{worse} с наилучшим и наихудшим значениями фитнес-функции. Частицу s_{best} исключают из произвольного клуба, к которому она принадлежит, а частицу

s_{worse} – включают в произвольный клуб, в котором она отсутствует. Через некоторое фиксированное число итераций для частиц, не являющихся наилучшими или наихудшими в клубе, производят случайное изменение их уровней вхождения в клубы в направлении стандартного уровня вхождения. Такой прием улучшает стабильность алгоритма и равномерность распределения частиц по клубам.

Исключение частицы s_{best} из клуба имеет целью предотвратить преждевременную сходимость алгоритма, так как эта частица оказывает сильное влияние на соседние частицы в клубе. Включение частицы s_{worse} в некоторый другой клуб позволяет увеличить число соседних с нею частиц. Поскольку частицы в новом клубе, в общем случае, не хуже частиц, находящихся в исходном клубе (по значениям фитнес-функции), миграция частицы s_{worse} дает ей возможность улучшить свою приспособленность.

На параметр m_i накладывается ограничение $m_i \in [m_{\min} : m_{\max}]$, так что каждая из частиц s_i, $i \in [1:|S|]$ во всех случаях входит не менее чем в $m_{\min}$ и не более чем в $m_{\max}$ клубов.

Алгоритм *CBPSO* по сравнению с каноническим алгоритмом *PSO* обеспечивает лучшие результаты с точки зрения сходимости и выхода из локальных минимумов фитнес-функции.

Можно предложить значительное число вариантов алгоритма *CBPSO*. Так в клубах можно использовать топологии соседства частиц, отличные от клики. Возможны варианты, в которых клубы связаны между собой с помощью той или иной топологии соседства.

Алгоритм роя частиц со стереотипированием (*PSO with stereotyping*). Основная идея алгоритма состоит в статическом разделении частиц по фиксированному числу кластеров. Критерием разделения является геометрическая близость частиц в пространстве поиска $R^{|X|}$. В качестве центра кластера используют геометрический центр координат частиц, образующих этот кластер. В формулах для вычисления приращений координат частицы s_i, $i \in [1:|S|]$, ее собственные координаты заменяют координатами центра кластера, которому принадлежит эта частица. Например, в формуле (3.2) канонического алгоритма *PSO* указанным образом заменяют компоненту X_i.

Алгоритм может быть скомбинирован как с каноническим алгоритмом *PSO*, так и с его модификацией *FIPS*.

Алгоритм с дополнением графа соседства частиц (*Dynamic Sociometry PSO*, *DSPSO*) является самым ярким представителем алгоритмов роя частиц с динамической топологией соседства. Алгоритм *DSPSO* реализует технику так называемого *перехода от обзора к поиску* (*switch from exploration to exploitation*).

Как отмечалось выше, разреженные графы соседства частиц обеспечивают высокую эффективность на начальных итерациях алгоритмов роя частиц, когда требуется широкий обзор поискового пространства. Напротив, плотные графы эффективны на завершающих итерациях поиска, когда необходима быстрая локализация минимума фитнес-функции, в окрестности которого находятся частицы популяции.

Идея алгоритма *DSPSO* состоит в следующем. На начальных итерациях алгоритма используем сильно разреженный граф соседства частиц. Через фик-

сированное число итераций k в этот граф добавляем новое ребро, например, по следующему правилу.

1) С помощью генератора натуральных случайных чисел, равномерно распределенных в интервале $[1:|S|]$, выбираем два числа i_1, i_2 таких, что $i_1 \neq i_2$ и частицы s_{i_1}, s_{i_2} не являются соседями в текущей топологии соседства.

2) В текущий граф соседства частиц добавляем ребро $\left(s_{i_1}, s_{i_2}\right)$.

Наряду со случайным выбором чисел i_1, i_2 может использоваться то или иное детерминированное правило их выбора. В результате рассмотренного процесса, в конце концов, при достаточном числе итераций получим граф соседства типа клика.

Допустим, что в популяции из $|S|$ частиц, имеющей начальную топологию соседства кольцо, необходимо получить топологию клика после выполнения $0{,}8\hat{t}$ итераций, где $\hat{t}$ – максимально допустимое число итераций. Тогда легко видеть, что добавлять ребро в текущий граф соседства частиц следует в общей сложности $(|S|-2)$ раз, то есть каждые

$$k = \left\lfloor \frac{0{,}8\,\hat{t}}{|S|-2} \right\rfloor$$

итераций. Здесь $\lfloor \cdot \rfloor$ – символ ближайшего целого меньшего.

3.1.5. Гибридный алгоритм на основе роя частиц и имитации отжига

Известны примеры гибридизации алгоритма *PSO* практически со всеми популяционными алгоритмами, а также с целым рядом классических и неклассических алгоритмов локальной оптимизации. Рассмотрим в качестве примера гибридизацию алгоритма *PSO* с алгоритмом имитации отжига *SA* (см. 1.4.2). Можно предложить большое число способов такой гибридизации (гл. 6). Используем конвейерную гибридизацию по следующей схеме (рис. 3.11).

1) Выполняем t_1 итераций алгоритма роя частиц.

2) Находим частицу $s_*(t_1) \in \{s_i(t_1), i \in [1:|S|]\}$ с лучшим значением фитнес-функции на момент окончания указанного числа итераций и передаем ее координаты X_* в качестве начальных алгоритму имитации отжига.

3) Выполняем t_2 итераций алгоритма имитации отжига – находим частицу $s_*(t_2)$ и новый вектор X_*.

4) Проверяем выполнение условия завершения итераций. Если это условие выполнено, полагаем, что вектор X_* является решением задачи, и заканчиваем вычисления. В противном случае координаты этого вектора передаем алгоритму роя частиц в качестве нового начального положения частицы s_* (остальные частицы роя оставляем без изменений) и переходим к шагу 1.

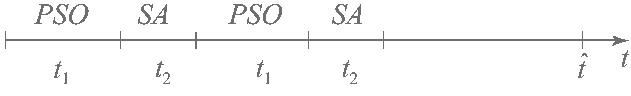


Рис. 3.11. К схеме конвейерной гибридизации алгоритмов *PSO* и *SA*

Очевидным вариантом данной схемы, требующим больших вычислительных затрат, является применение отжигового алгоритма к каждой из частиц популяции.

Проиллюстрируем эффективность рассмотренного гибридного алгоритма примером.

Пример 3.3. Рассмотрим задачу безусловной оптимизации функций Розенброка, Химмельблау и Растригина (Приложение Б) комбинацией канонического алгоритма роя частиц и алгоритма сверхбыстрого отжига при следующих значениях свободных параметров: число частиц в популяции $|S| = 30$; начальная «температура» $\overline{\varepsilon}_0(0) = 100000$; декремент затухания $\beta = 0,75$. Положим, что фитнес-функция совпадает с целевой функцией, то есть $\varphi(X) = f(X)$. Для генерации начального положения частиц используем параллелепипед *П*, в котором $x_i^- = -100$, $x_i^+ = 100$; $i \in [1:|X|]$.

В качестве критерия окончания итераций используем стагнацию вычислительного процесса в течение $\delta_t = 30$ итераций, а в качестве критериев эффективности алгоритма – число испытаний n_f (суммарное число вычислений значений целевой функции), а также достигнутые точности $\varepsilon_X, \varepsilon_f$ локализации глобального минимума по вектору варьируемых параметров и целевой функции соответственно.

Рассмотрим зависимость указанных критериев эффективности алгоритма от значений параметра $m = t_1/t_2$ – отношения числа итераций алгоритма роя частиц к такому же числу итераций отжигового алгоритма. Проведенные исследования показывают, что по критериям $\varepsilon_X, \varepsilon_f$ лучшие результаты, полученные по схеме мультистарта из $\hat{n} = 30$ запусков алгоритма, практически не зависят от величины m и имеют значения, представленные в табл. 3.1.

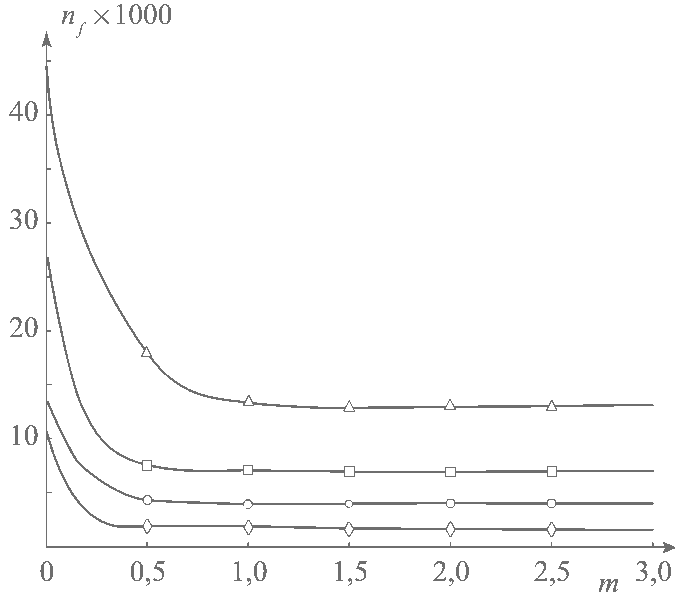


Рис. 3.12. Суммарное число испытаний n_f в функции значений параметра *m*, функция Розенброка: ◊ – |*X*| = 2; ○ – |*X*| = 4; □ – |*X*| = 8; Δ – |*X*| = 16

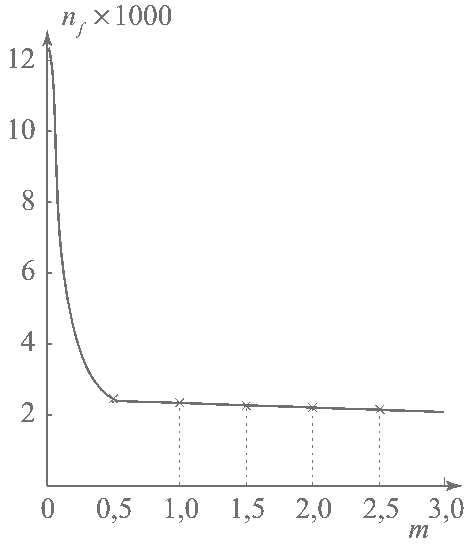


Рис. 3.13. Суммарное число испытаний n_f в функции значений параметра m, функция Химмельблау: $|X| = 2$

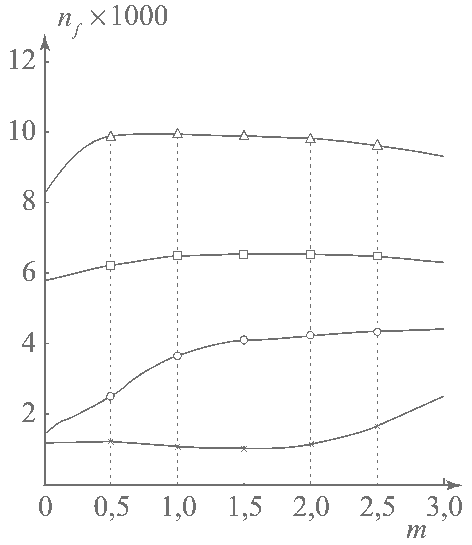


Рис. 3.14. Суммарное число испытаний n_f в функции значений параметра m, функция Растригина: × – $|X| = 2$; ○ – $|X| = 4$; □ – $|X| = 8$; Δ – $|X| = 16$

Таблица 3.1 **Некоторые результаты исследования эффективности гибридного алгоритма: $m \in [0{,}02;\ 3{,}0]$**

Функция	Эффективность							
	$\lvert X\rvert = 2$		$\lvert X\rvert = 4$		$\lvert X\rvert = 8$		$\lvert X\rvert = 16$	
	ε_X	ε_f	ε_X	ε_f	ε_X	ε_f	ε_X	ε_f
Розенброка	10^{-3}	10^{-2}	10^{-2}	10^{-2}	1	10^{-1}	10	10
Химмельблау	10^{-5}	10^{-8}	–	–	–	–	–	–
Растригина	10^{-5}	10^{-9}	10^{-3}	10^{-7}	10^{-1}	1	1	10

В тех же условиях эффективность гибридного алгоритма по критерию числа испытаний n_f иллюстрируют рис. 3.12 – 3.14.

Рис. 3.12, 3.13 показывают, что для одноэкстремальной функции Розенброка и четырехэкстремальной функции Химмельблау число испытаний n_f быстро уменьшается с уменьшением относительного числа итераций отжигового алгоритма (увеличением значений параметра m). При больших значениях данного параметра число испытаний n_f стабилизируется, и при тем больших значениях параметра, чем выше размерность вектора варьируемых параметров задачи $|X|$.

Рис. 3.14 иллюстрирует иной характер зависимости величины n_f от величины m для многоэкстремальной функции Растригина. В данном случае при $|X| = 2$, $|X| = 4$ число испытаний n_f незначительно, но увеличивается с ростом значений параметра m, то есть при уменьшении доли итераций отжигового алгоритма. Напротив, при $|X|>4$ величина n_f незначительно уменьшается с увеличением значений параметра m •

3.1.6. Пример решения задачи с использованием алгоритма роя частиц

Рассмотрим одну из обратных кинетических задач реакции циклоалюминирования олефинов, которые находят широкое применение в органическом и металлоорганическом синтезе как эффективный способ получения практически важных алюминийорганических соединений заданной структуры.

Уравнения материального баланса процесса циклоалюминирования олефинов триэтилалюминием в присутствии Cp_2ZrCl_2 можно представить в виде задачи Коши для системы ОДУ

$$\frac{dx_i}{d\tau} = f_i,\ i \in [1:13],\ \tau \in [0;\hat{\tau}],\ x_i(0) = x_i^0, \tag{3.9}$$

где x_i – концентрация (мольная доля) i-й компоненты смеси, а правые части системы определяют равенства

$$\begin{aligned}
f_1 &= -2w_1 - w_2 + 2w_3 + 2w_5 - w_7, & f_8 &= w_5,\\
f_2 &= -w_1, & f_9 &= -w_6,\\
f_3 &= w_1 - w_2 - 2w_3 - w_4 + w_7 + w_9, & f_{10} &= w_6 - w_7,\\
f_4 &= w_1 - w_2 - 2w_3 + 2w_8, & f_{11} &= w_7,\\
f_5 &= w_4 - w_5 - w_6 + w_9, & f_{12} &= w_2,\\
f_6 &= w_3 - w_8, & f_{13} &= w_3 + w_4 + w_5.\\
f_7 &= w_8 - w_9,
\end{aligned}$$

Скорости реакций $w_1, w_2, ..., w_9$ зависят от неизвестных констант $k_j,\ j \in [1:9]$:

$$\begin{aligned}
w_1 &= k_1\, x_1^2\, x_2 - k_{10}\, x_3\, x_4, & w_4 &= k_4\, x_3, & w_7 &= k_7\, x_1\, x_{10},\\
w_2 &= k_2\, x_1\, x_3, & w_5 &= k_5\, x_1\, x_5, & w_8 &= k_8\, x_1^2\, x_6,\\
w_3 &= k_3\, x_3^2\, x_4^2, & w_6 &= k_6\, x_5\, x_9, & w_9 &= k_9\, x_7.
\end{aligned}$$

Приведенная система ОДУ является сверхжесткой, так что для ее численного интегрирования приходится использовать специальные алгоритмы (в нашем случае – алгоритм Мишельсена с адаптивным шагом).

Известен некоторый набор экспериментальных значений величин $x_i^{эксп}(\tau_l)$, $l \in [1:m]$, $\tau_l \in [0;\hat{\tau}]$. Ставится задача найти такие значения коэффициентов $k_j,\ j \in [1:9]$, которые обеспечивают наилучшее согласие экспериментальных и модельных данных. В качестве меры близости этих данных используем величину

$$\phi(K) = \sum_{i=1}^{13}\sum_{l=1}^{m}\left| x_i(\tau_l) - x_i^{эксп}(\tau_l)\right|, \tag{3.10}$$

где $K = (k_1, k_2, ..., k_9)$.

Таким образом, рассматриваемую обратную кинетическую задачу сводим к задаче многомерной безусловной оптимизации с девятимерным вектором варьируемых параметров K и целевой функцией $\phi(K)$.

Используем для решения поставленной задачи многороевый алгоритм роя частиц *GIPSO* (п. 9.4.1), в котором каждый из суброев эволюционирует в соот-

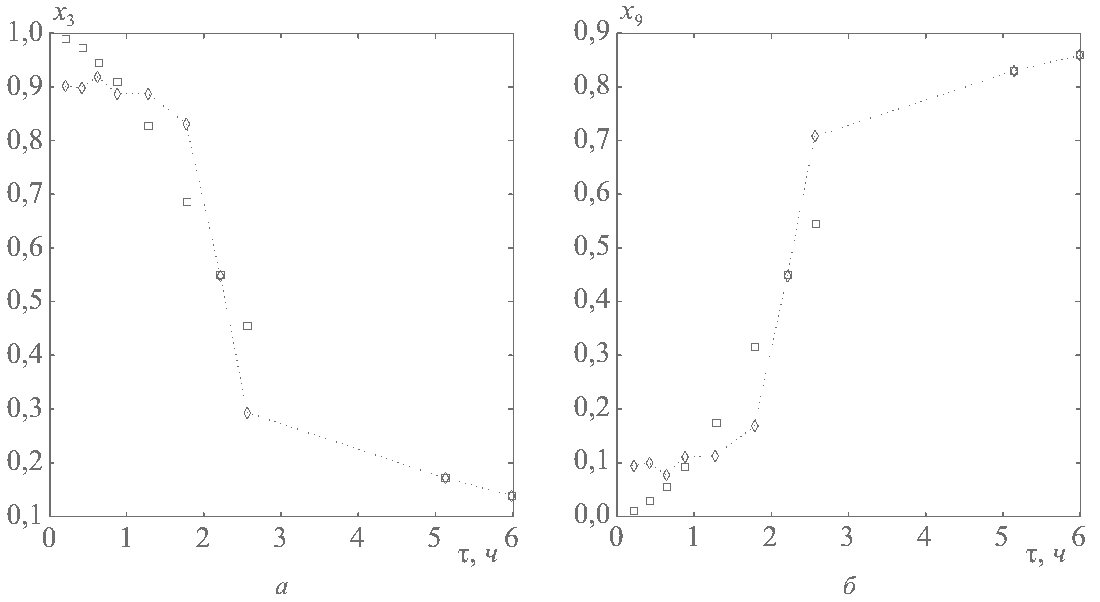


Рис. 3.15. Расчетные и экспериментальные результаты решения обратной задачи химической кинетики: *а* — $x_3(\tau)$, *б* — $x_9(\tau)$; □ – расчет; ◊ – эксперимент

ветствии с каноническим алгоритмом *PSO*. Вычисление значений целевой функции производим по следующей схеме:

1) при заданных коэффициентах $k_1, k_2, ..., k_9$ интегрируем систему ОДУ (3.9) и определяем расчетные значения фазовых переменных $x_i(\tau_l)$, $l \in [1:m]$;

2) подставляем полученные величины в выражение (3.10).

Некоторые результаты решения задачи иллюстрирует рис. 3.15. Представленные результаты соответствуют невязке $\phi(K^*) = \phi^* = 3,08$ и

$$K^* = (1000,0;\ 0,020;\ 0,021;\ 0,972;\ 651,818;\ 0;\ 988,40;\ 532,473;\ 0,813).$$

3.2. МУРАВЬИНАЯ ОПТИМИЗАЦИЯ

Муравьиная оптимизация была предложена Дориго (*M. Dorigo*), Маньеццо (*V. Maniezzo*) и Колорни (*A. Colorni*) в 1992 г. как алгоритм решения задачи коммивояжера (*Traveling Salesman Problem*, *TSP*). Эта задача предполагает выбор кратчайшего пути в графе, так что имеется прямая аналогия с задачей, которую каждодневно решают муравьи, оптимизируя свои пути к пище. В настоящее время известно большое число дискретных и непрерывных алгоритмов муравьиной оптимизации, получивших объединяющее наименование *алгоритмы оптимизации колонией муравьев* (*Ant Colony Optimization*, *ACO*).

Первым из алгоритмов *ACO*, предназначенным для решения задачи коммивояжера, стал *алгоритм муравьиной системы* (*Ant System*, *AS*). Затем появилось несколько модификаций этого алгоритма: плотностный (*ant-density*), количественный (*ant-quality*), циклический (*ant-cycle*). Эксперименты показали

значительное преимущество последнего алгоритма перед двумя другими. Поэтому плотностный и количественный алгоритмы были отброшены и сейчас каноническим алгоритмом *ACO* для решения задачи коммивояжера считают циклический алгоритм.

В последующем был предложен ряд более развитых алгоритмов *ACO* для решения задачи коммивояжера, наиболее известными из которых являются алгоритм муравьиных систем, основанный на элитной стратегии; такой же алгоритм, основанный на ранжировании; алгоритм системы муравьиных колоний; максиминный алгоритм муравьиных систем.

Успехи в решении задачи коммивояжера с помощью алгоритмов *ACO* стимулировали разработку *ACO*-алгоритмов для решения других задач дискретного программирования, таких как задача о назначениях, задача календарного планирования, транспортная задача и так далее. Оказалось, что алгоритмы *ACO* показывают хорошие результаты при решении широкого класса задач, которые могут быть сведены к задаче поиска на графах.

Первым вариантом алгоритма *ACO*, разработанным для решения задачи непрерывной оптимизации, стал алгоритм *непрерывной оптимизации колонией муравьев* (*Continuous Ant Colony Optimization*, *CACO*), наметки которого были предложены Билчевым (*G. Bilchev*) и Парми (*I. C. Parmee*) в 1995 г.

В настоящее время известно множество модификаций алгоритма *CACO*, например: алгоритм непрерывно взаимодействующей муравьиной колонии (*Continuous Interaction Ant Colony*, *CIAC*); расширенный алгоритм оптимизации колонией муравьев для непрерывных областей (*extended Ant Colony Optimization to continuous domains*, ACO_R); алгоритм непрерывной системы муравьиной колонии (*Continuous Ant Colony System*, *CACS*); алгоритм бинарной муравьиной системы (*Binary Ant System*, *BAS*); алгоритм прямого назначения (*Direct Application of Ant Colony Optimization*, *DACO*).

К недостаткам *ACO*-алгоритмов поисковой оптимизации можно отнести затрудненность их теоретического анализа и наличие большого числа свободных параметров, так как, с одной стороны, от значений этих параметров во многих случаях сильно зависит эффективность поиска, с другой стороны, отсутствуют правила априорного выбора их оптимальных значений.

Далее рассматриваем бионические предпосылки муравьиных алгоритмов; наиболее известный из непрерывных муравьиных алгоритмов – алгоритм *CACO*; алгоритм *CIAC*; непрерывный ортогональный муравьиный алгоритм *COAC*; гибридный алгоритм непрерывно взаимодействующей муравьиной колонии *HCIAC*. В завершение параграфа приводим пример использования одного из алгоритмов *ACO*.

3.2.1. Бионические предпосылки

Муравьи относятся к социальным насекомым, живущим внутри некоторого коллектива – колонии. Число муравьев в одной колонии достигает нескольких миллионов. Поведение муравьев при транспортировании пищи, преодолении препятствий, строительстве муравейника и других действиях зачастую приближается к теоретически оптимальному. Например, известно, что структура взаимосвязи колоний в мегаколонии муравьев *Formica lugubris* в Швей-

царии близка к минимальному остовному дереву, соединяющему все колонии мегаколонии.

«Социальное» поведение муравьев строится на основе самоорганизации, то есть на основе множества механизмов, обеспечивающих достижение системой глобальной цели в результате низкоуровневого взаимодействия ее элементов. Принципиальной особенностью такого взаимодействия является использование элементами системы только локальной информации. Любое централизованное управление в этом случае отсутствует.

Муравьи используют два способа передачи информации. Прямой способ реализуется через *канал индивидуальных взаимодействий* (*direct inter-individual communication*) между парой муравьев посредством обмена пищей, мандибулярных (челюстных) манипуляций, визуального и химического контактов. Назовем этот канал *прямым каналом*.

Непрямому способу обмена информацией между муравьями соответствует *канал стигмергии* (*stigmergy*) или *следовой* (*trail*) *канал*. В широком смысле данный канал представляет собой разнесенный во времени тип взаимодействия, когда один субъект взаимодействия изменяет некоторую часть окружающей среды, а остальные используют информацию о состоянии этой части среды позже, когда находятся в ее окрестности. У муравьев стигмергия реализуется посредством *феромона* (*pheromone*) – специального секрета, который откладывается в виде следа при перемещении муравья и который может восприниматься муравьями в течение нескольких суток. Чем выше концентрация феромона на тропе, тем больше муравьев будет по ней двигаться. Со временем феромон испаряется, что позволяет муравьям адаптировать свое поведение к изменениям внешней среды.

Общественное поведение муравьев в колонии подчинено идее *гетерархии*, которая, в противоположность иерархии, предполагает не вертикальные связи между агентами системы, а горизонтальное распределение системных функций между этими агентами, при котором сами агенты равноправны между собой. В колонии муравьев мы имеем дело с *плотной гетерархией*, когда каждый из муравьев может связаться с любым другим в любой момент времени. При этом информация передается через два рассмотренных выше канала связи – прямой канал и канал стигмергии. Отметим, что система, построенная на основе концепции гетерархии, обладает эмерджентными свойствами (*emergent properties*), то есть свойствами, не присущими составляющим ее агентам, рассматриваемым вне системы, но присущими системе в целом.

Канал обмена информацией между элементами гетерархической системы определяют, вообще говоря, следующие его характеристики: область действия канала – размер группы особей популяции, в пределах которой возможен обмен информацией; объем и время хранения информации в канале; способ изменения информации в канале – какие особи или внешние по отношению к системе факторы и по каким правилам могут изменять информацию. Различные комбинации указанных характеристик канала порождают широкое множество их вариантов.

По приведенной классификации область действия канала стигмергии в муравьиной колонии охватывает всю колонию, поскольку феромонная метка мо-

жет быть оставлена каждым из муравьев популяции и принципиально доступна любому другому муравью. Количество феромона постепенно уменьшается, так что феромонная метка рано или поздно исчезает. Таким образом, время хранения информации в канале равно времени активности феромонного следа.

Информация по прямому каналу также может быть передана каждым муравьем популяции любому другому муравью, но, в отличие от канала стигмергии, в каждый данный момент времени – только одному муравью. Каждым муравьем сообщение может быть получено от нескольких других муравьев.

Всюду далее в данном параграфе речь идет о задаче максимизации фитнес-функции.

3.2.2. Алгоритм непрерывной оптимизации колонией муравьев

Основными особенностями алгоритма *CACO* являются комбинирование глобального и локального поиска, использование для глобального поиска генетического алгоритма, использование для локального поиска только канала стигмергии.

Рассматриваем задачу многомерной глобальной условной максимизации в гиперпараллелепипеде *П*. В каждой точке $X_i(\tau)$, в которой побывал муравей, $i \in [1:|S|]$, он оставляет *феромонную точку* $\phi_i(t,\tau)$, интенсивность которой (количество феромона) на данной итерации t равна $\theta_i(t,\tau)$, где $\tau \in [0:t]$ – номер итерации, на которой оставлена эта точка. Количество феромона в феромонных точках с ростом числа итераций уменьшается (феромон испаряется) в соответствии с рекуррентной формулой

$$\theta_i(t,\tau) = \begin{cases} b_\theta\, \theta_i(t-1,\tau), & \theta_i(t-1,\tau) \ge \theta_{\min}, \\ 0, & \text{иначе}, \end{cases} \tag{3.11}$$

где b_θ – коэффициент устойчивости феромонных точек. Формула (3.11) означает, что феромонная точка исчезает, если количество феромона в ней становится меньшим величины $\theta_{\min}$.

Совокупность феромонных точек

$$\Phi_i(t) = \{\phi_i(t,\tau) \mid \tau \in [0:t],\ \theta_i(t,\tau) > \theta_{\min}\},\ i \in [1:|S|]$$

образует феромонный след муравья s_i. Число феромонных точек в этом следе обозначаем $|\Phi_i(t)|$.

На текущей итерации *t активными* являются феромонные точки, принадлежащие всем феромонным следам $\Phi_i(t)$, $i \in [1:|S|]$. Совокупность этих точек обозначим $\Phi(t) = \{\Phi_i(t),\ i \in [1:|S|]\}$. Тогда $|\Phi(t)| = \sum_{i=1}^{|S|} |\Phi_i(t)|$ – общее число активных феромонных точек на текущей итерации *t*.

Введем еще обозначения: $S^g = \{s_i^g,\ i \in [1:|S^g|]\}$, $S^l = \{s_i^l,\ i \in [(|S^g|+1):|S|]\}$ – множества муравьев, предназначенных для глобального и локального поиска соответственно; $|S^g| + |S^l| = |S|$.

Во введенных обозначениях схема алгоритма *CACO* имеет следующий вид.

Этап инициализации. Создаем $|S|$ *регионов поиска*. Инициализируем в каждом из регионов $X_i(0)$, $i \in [1:|S|]$ феромонную точку. Полагаем счетчик числа итераций $t = 1$.

Этап оптимизации.

1) Вычисляем в каждом из регионов поиска значение фитнес-функции $\varphi(X_i) = \varphi_i$.

2) Сортируем регионы в порядке возрастания величины φ_i и представляем результат сортировки в виде линейного списка.

3) С вероятностью мутации ξ_m выполняем генетическую операцию мутации для первых $0{,}9|S^g|$ (наименее приспособленных) муравьев указанного списка.

4) Аналогично выполняем генетическую операцию кроссовера для следующих $0{,}1|S^g|$ муравьев того же списка.

5) Инициализируем в каждом из $|S^g|$ регионов поиска, полученных в результате предыдущих операций, феромонную точку.

6) На основе канала стигмергии для каждого из $|S^l|$ муравьев выполняем локальный поиск лучшего региона:

– если регион с лучшим значением фитнес-функции найден, то инициализируем соответствующую феромонную точку и перемещаем одного из муравьев множества S^l в этот регион;

– в противном случае случайным образом делаем попытку найти новый лучший регион.

7) По формуле (3.11) моделируем испарение феромона.

Этап завершения вычислений. Проверяем условие окончания итераций. Если это условие выполнено, завершаем вычисления, в противном случае возвращаемся к этапу оптимизации.

Каждый регион поиска $X_i(0)$, создаваемый на этапе инициализации, представляет собой случайную точку в области *П*, координаты которой определяем по правилу $x_{i,j}(0) = U_1(x_i^-; x_i^+)$, $i \in [1:|S|]$, $j \in [1:|X|]$. Региону $X_i(0)$ ставим в соответствие феромонную точку $\phi_i(0,0)$ интенсивностью $\theta_i(0,0) = \theta_{init}$, где θ_{init} – начальное количество феромона.

Для мутации используем гауссов мутатор

$$x'_{i,j} = x_{i,j} + N_1(0,\sigma),\ j \in [1:|X|],$$

где среднее квадратичное отклонение σ называем *шагом мутации* (*mutation step size*). С ростом числа итераций уменьшаем (редуцируем) эту величину по закону

$$\sigma(t) = \sigma_{\max}\left(1 - \left(U_1(0;1)\right)^{(1-\frac{t}{\bar{t}})^{b_m}}\right),$$

где $\sigma_{\max}$ – максимальная величина шага мутации, b_m – натуральный *параметр нелинейности*. Нелинейная редукция шага мутации позволяет на начальных итерациях поиска обеспечить широту поиска, а на завершающих итерациях – точность локализации максимума.

В качестве оператора кроссовера используем арифметический кроссовер

$$x'_{i,k} = ux_{i,k} + (1-u)x_{j,k},\ i, j \in [1:|S^g|],\quad k \in [1:|X|],\ i \neq j,$$

где $u = U_1(0;1)$. Родительские особи s_i, s_j для скрещивания выбираем из числа $0{,}1 \cdot |S^g|$ муравьев множества S^g с равной вероятностью.

Локальный поиск лучшего региона выполняем по следующей схеме.

1) Случайным образом выбираем муравья s_i из числа последних $|S^l|$ муравьев указанного выше линейного списка.

2) С учетом *интереса* данного муравья вычисляем координаты центра тяжести текущих активных феромонных точек

$$W_i(t) = \sum_{j,k} \overline{\varpi}_{i,j,k}(t) X_{j,k},\ j \in [1:|S|],\ k \in [1:|\Phi_j(t)|], \tag{3.12}$$

где $X_{j,k}$ – k-я точка феромонного следа $\Phi_j(t)$;

$$\overline{\varpi}_{i,j,k}(t) = \frac{\varpi_{i,j,k}(t)}{\sum_{l=1}^{|S|} \sum_{m=1}^{|\Phi_j(t)|} \varpi_{i,l,m}(t)} \ -$$

нормированный интерес i-го муравья к k-й точке феромонного следа $\Phi_j(t)$ (см. ниже).

3) Вычисляем координаты нового возможного положения муравья s_i по формуле

$$X'_i = X_i + V_i.$$

Здесь V_i – шаг в направлении $(X_i - W_i(t))$, величина которого равна $abs\,(b_{beg} - b_{inc} a_i)$, где b_{beg}, b_{inc} – положительные определяемые пользователем константы, имеющие смысл начальной величины шага и его инкремента соответственно; a_i – *слабость* или *возраст* региона X_i (в оригинале – *weakness* и *age* соответственно).

Начальное значение величина a_i получает при создании соответствующего региона поиска. Если значение фитнес-функции в регионе X'_i выше значения той же функции в регионе X_i, то значение величины a_i увеличиваем, а в противном случае – уменьшаем. Данный механизм позволяет уменьшать шаг локального поиска с приближением муравья s_i к положению максимума фитнес-функции.

Интерес муравья s_i к k-й точке феромонного следа $\Phi_j(t)$ определяет формула

$$\varpi_{i,j,k}(t) = \frac{\overline{\rho}(t)}{2} \theta_{j,k}(t) \exp(-\rho_{i,j,k}(t)),\ i, j \in [1:|S|],\ k \in [1:|\Phi_j(t)|],$$

в которой $\overline{\rho}(t)$ – среднее текущее расстояние между двумя муравьями популяции, $\theta_{j,k}(t)$ – количество феромона в указанной феромонной точке, $\rho_{i,j,k}(t) = \left\| X_i(t) - X_{j,k} \right\|_E$ – евклидово расстояние между регионом $X_i(t)$ и той же феромонной точкой.

Положим, что на данной итерации t активными являются все регионы, в которых побывал каждый из муравьев популяции. Тогда из формулы (3.12) следует, что для муравья s_i, $i \in [1:|S|]$ координаты центра тяжести всех активных феромонных точек можно вычислить по формуле

$$W_i(t) = \sum_{j,k} \overline{\varpi}_{i,j,k}(t) X_{j,k}, \; j \in [1:|S|], \; k \in [0:t].$$

В качестве условия окончания итераций используют типичные для популяционных алгоритмов оптимизации условия.

Алгоритм *CACO* содержит следующие основные свободные параметры:

θ_{init} – начальное количество феромона (рекомендуемое значение $\theta_{init} = 1,0$);

$\theta_{\min}$ – количество феромона, ниже которого феромонная точка становится неактивной (рекомендуемое значение $\theta_{\min} = 0,00000001$);

$b_\theta \in (0;\infty)$ – коэффициент устойчивости феромонных точек (рекомендуемое значение $b_\theta = 0,1$);

ξ_m, σ_m, $\sigma_{\max}$, b_m – вероятность мутации, шаг мутации, максимальная величина шага мутации и параметр нелинейности мутации соответственно;

b_{beg}, b_{inc} – начальная величина шага локального поиска и его инкремент соответственно;

$a_i(0), i \in [1:|S|]$ – начальное значение возраста регионов.

По общему свойству популяционных алгоритмов поисковой оптимизации с увеличением размера популяции $|S|$ алгоритм *CACO* повышает вероятность локализации глобального максимума фитнес-функции (ценой повышения вычислительной сложности каждой итерации). Для функций, имеющих простую топологию, хорошие результаты может показать популяция из всего примерно 10 муравьев. Для сложных многомерных многоэкстремальных функций число муравьев должно быть порядка 100.

Авторы алгоритма рекомендуют использовать число особей в популяции, равное $|S| = |S|_{\max}\left(1 - \exp\left(-0,1|X|\right)\right) + 5$, где $|S|_{\max} = 1000$ – максимально допустимое число муравьев в популяции.

В терминах алгоритма роя частиц уменьшение количества феромона $\theta_{\min}$ и увеличение коэффициента устойчивости феромонных точек b_θ повышает влияние социальной компоненты на эволюцию муравьев. В этом случае феромонные следы $\Phi_j(t)$, $j \in [1:|S|]$ «удлиняются» (возрастают величины $|\Phi_j(t)|$) и влияние остальных муравьев на каждого данного муравья увеличивается. В результате повышается вероятность «застревания» муравьев в локальных максимумах целевой функции.

3.2.3. Алгоритм непрерывно взаимодействующей колонии муравьев

Реализация идеи стигмергии является не единственным способом использования муравьиного интеллекта при решении задач поисковой оптимизации. Естественной является идея использования для обмена информацией между муравьями наряду с каналом стигмергии также канала прямого обмена. Алгоритм, использующий оба указанных канала, называют *алгоритмом CIAC*. Алгоритм предложен в работах Дрео (*J. Dreo*) и Сиарри (*P. Siarry*).

Рассмотрим задачу многомерной глобальной условной максимизации в гиперпараллелепипеде *П*. Используем обозначения и допущения предыдущего пункта. Кроме того, положим, что каждый муравей s_i имеет *память*, в которую другие муравьи могут присылать *сообщения*. Сообщение $m_{i,j}$, полученное муравьем s_i от муравья s_j, $i, j \in [1:|S|]$, $j \neq i$, содержит следующую информацию: текущее положение муравья X_j; значение фитнес-функции в этой точке φ_j. Таким образом, в алгоритме *CIAC* муравья s_i определяет набор

$$\left\langle X_i, \varphi_i, m_{i,j_1}, m_{i,j_2}, \ldots \right\rangle, \; j_1, j_2, \ldots \in [1:|S|], \; j_1 \neq j_2 \neq \ldots \neq i .$$

Схема алгоритма *CIAC* имеет следующий вид.

1) По схеме алгоритма *CACO* создаем начальную популяцию муравьев $S = \{s_i, i \in [1:|S|]\}$ и инициируем феромонные точки. Инициируем также память каждого из муравьев s_i, помещая в нее координаты векторов $X_i(0)$ и значения фитнес-функции $\varphi_i(0)$. Полагаем счетчик числа итераций $t = 1$.

2) По схеме, близкой к схеме, используемой алгоритмом *CACO*, моделируем работу канала стигмергии. Отличие заключается лишь в том, что перемещение муравья s_i в направлении центра тяжести $W_i(t)$ производится на случайное расстояние в пределах *размерного параметра* муравья

$$r_i = \frac{N_1(0,5; \sigma_r)}{b_r}, \; i \in [1:|S|],$$

где b_r, σ_r – *делитель размерного параметра* и среднеквадратичное отклонение соответственно.

3) Моделируем функционирование канала прямого взаимодействия между муравьями.

3а) Случайным образом выбираем одно из сообщений из памяти муравья $s_i \in S$, к примеру сообщение от муравья $s_j \in S$; $i, j \in [1:|S|]$, $j \neq i$.

3б) Сравниваем значения фитнес-функции φ_i, φ_j. Если $\varphi_j > \varphi_i$, то перемещаем муравья s_i в случайную точку гиперсферы с центром в точке X_j и радиусом, равным величине размерного параметра r_i этого муравья.

3в) В противном случае выполняем следующие действия: посылаем случайному муравью $s_k \in S$, $k \in [1:|S|]$, $k \neq i$ сообщение $m_{k,i}$, содержащее координаты точки X_i и значение фитнес-функции φ_i; случайным образом перемещаем муравья s_i в пределах гипершара радиуса r_i с центром в точке X_i.

4) По формуле (3.11) моделируем испарение феромонных точек.

5) По схеме алгоритма *CACO* проверяем выполнение условия окончания итераций. Если это условие выполнено, завершаем вычисления, в противном случае полагаем $t = t + 1$ и переходим к шагу 2.

За одну итерацию каждый из муравьев может получить более одного сообщения. Число сообщений, которые могут поместиться в памяти муравья, ограничено натуральной величиной $m_{\max}$. Если муравью поступает новое сообщение, а его память уже исчерпана, то это сообщение игнорируется.

Кроме свободных параметров, аналогичных параметрам алгоритма *CACO*, алгоритм *CIAC* содержит следующие параметры:

b_r – делитель размерного параметра;

$\sigma_r \in [0;1]$ – среднеквадратичное отклонение величины размерного параметра при $b_r = 1,0$ (рекомендуемое значение равно $\sigma_r = 0,9$);

$m_{\max} \in [0;\infty)$ – максимально допустимое число сообщений в памяти муравья (рекомендуемое значение $m_{\max} = 2/3\,|S|$).

Имеет место сильная зависимость основных характеристик алгоритма *CIAC* от значений параметров b_r, σ_r, определяющих величину размерного параметра r_i, $i \in [1:|S|]$. Увеличение значений делителя размерного параметра b_r, очевидно, уменьшает величину r_i и тем самым широту поиска. Повышение значения среднеквадратичного отклонения σ_r увеличивает дисперсию размерного параметра и диверсифицирует поиск. Можно рекомендовать использование малых значений величины b_r и больших значений величины σ_r на начальных стадиях поиска. На завершающих итерациях величину b_r целесообразно увеличивать, а σ_r – уменьшать.

На характеристики алгоритма *CIAC* существенное влияние может оказывать также величина $m_{\max}$. Увеличение значений этой величины диверсифицирует поиск, но снижает влияние социальной компоненты (за которую отвечает канал стигмергии).

Вычислительные эксперименты показывают, что алгоритм *CIAC*, использующий только прямой канал, с высокой вероятностью завершается в локальном максимуме фитнес-функции. Наоборот, канал стигмергии позволяет избежать подобного поведения алгоритма, но существенно увеличивает вычислительные затраты. Комбинация обоих каналов может обеспечить высокую вероятность локализации глобального максимума с приемлемыми вычислительными затратами.

Для повышения вероятности локализации глобального экстремума сложных многоэкстремальных функций может быть использована следующая модификация алгоритма *CIAC* на основе процедуры *встряски популяции*.

1) Выполняем t_c итераций алгоритма *CIAC*.

2) Отбираем n_w муравьев с худшими значениями фитнес-функции и исключаем из популяции.

3) Вместо удаленных муравьев случайным образом создаем в пространстве поиска n_w новых муравьев.

4) Если условие окончания итераций выполнено, завершаем вычисления, в противном случае переходим к шагу 1.

Процедура встряски популяции имеет свободные параметры t_c, n_w. Кроме того, возникает задача выбора правила, по которому следует назначать количество феромона в начальных точках расположения указанных n_w муравьев.

В качестве величины t_c целесообразно использовать некоторую монотонно убывающую функцию номера итерации t, например, функцию вида

$$t_c = t_c^0 + \alpha t,$$

где t_c^0 – начальное значение величины t_c, $\alpha \in (0;1)$ – заданный коэффициент.

Для вычисления значений феромона в новых точках можно использовать, например, формулу вида

$$\theta_i(t+1) = \nu\theta_i(t)\frac{\varphi(X_i(t))}{\varphi(X_i(t+1))},$$

где $\theta_i(t), \theta_i(t+1)$ – значения феромона, соответствующие положениям муравья s_i до и после его перемещения; $\varphi(X_i(t)), \varphi(X_i(t+1))$ – аналогичные значения фитнес-функции; $\nu > 1$ – свободный параметр.

Пример 3.4. Рассмотрим решение задачи глобальной условной минимизации пятимерной ($|X| = 5$) функции Растригина (Приложение Б) алгоритмом *CIAC* в гиперпараллелепипеде *П*, в котором $x_i^- = -5$, $x_i^+ = 5$, $i \in [1:5]$. Положим, что фитнес-функция обратна целевой функции.

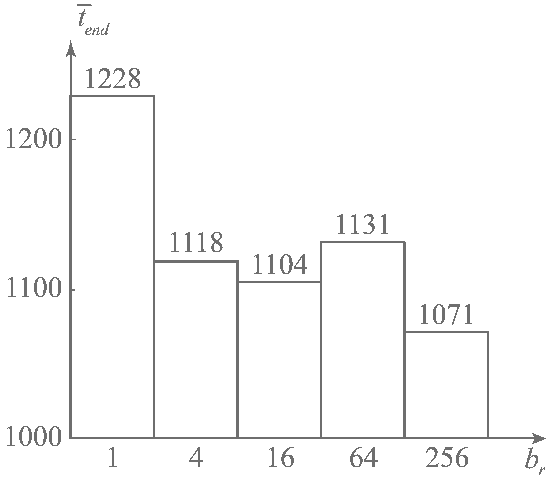


Рис. 3.16. Число итераций $\overline{t}_{end}$ в функции величины b_r

Аналогично примеру 3.1, для получения усредненных оценок эффективности алгоритма используем мультистарт с числом прогонов $\hat{n} = 30$. В качестве критерия окончания итераций используем стагнацию вычислительного процесса в течение $\delta_t = 10|S|$ итераций. Полагаем, что максимально допустимое число итераций $\hat{t} = 1500$.

Если не оговорено противное, то в качестве значений свободных параметров алгоритма используем их рекомендованные значения (см. выше). Эффективность алгоритма оцениваем средним числом итераций $\overline{t}_{end}$, средней абсолютной точностью $\overline{\varepsilon}$ локализации максимума фитнес-функции, а также оценкой вероятности его локализации $\tilde{\xi}$ с заданной точностью.

Рассмотрим прежде зависимость значений указанных критериев эффективности алгоритма от максимально допустимого числа сообщений в памяти муравья $m_{\max}$ при числе муравьев в популяции $|S| = 398$ и величине делителя размерного параметра $b_r = 30$. Результаты исследования показывают, что в диапазоне $1 \le m_{\max} \le 31$ величины $\overline{t}_{end}$, $\overline{\varepsilon}$, $\tilde{\xi}$ не существенно зависят от этого числа:

$$\max_{m_{\max}} \overline{t}_{end}(m_{\max}) = \overline{t}_{end}(1) = 1144; \ \min_{m_{\max}} \overline{t}_{end}(m_{\max}) = \overline{t}_{end}(6) = 1047;$$

$$\max_{m_{\max}} \overline{\varepsilon}(m_{\max}) = \overline{\varepsilon}(11) = 1{,}76 \cdot 10^{-3}; \ \min_{m_{\max}} \overline{\varepsilon}(m_{\max}) = \overline{\varepsilon}(6) = 9{,}95 \cdot 10^{-4};$$

$$\max_{m_{\max}} \tilde{\xi}(m_{\max}) = \tilde{\xi}(16) = 0{,}93; \ \min_{m_{\max}} \tilde{\xi}(m_{\max}) = \tilde{\xi}(26) = 0{,}63.$$

Зависимость указанных критериев от величины делителя размерного параметра $b_r \in [1:256]$ иллюстрируют рис. 3.16 – 3.18. Число муравьев в популяции

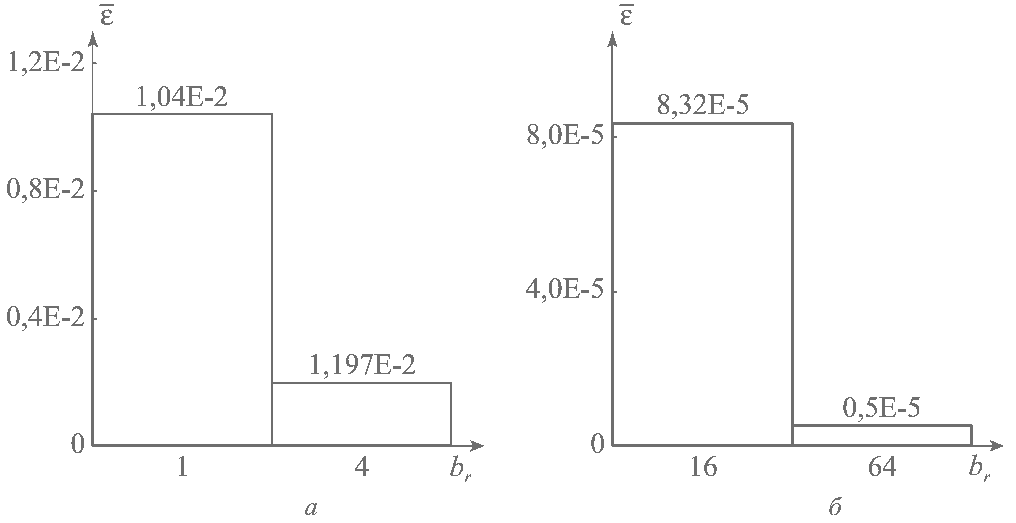


Рис. 3.17. Погрешность решения $\overline{\varepsilon}$ в функции величины b_r: *а* – $b_r = 1; 4$; *б* – $b_r = 16; 64$

также принято равным $|S| = 398$, максимальное число сообщений в памяти муравья положено равным $m_{\max} = 30$.

Рис. 3.16 показывает, что с ростом значений делителя размерного параметра b_r имеет место не слишком существенное снижение числа итераций $\overline{t}_{end}$ (со значения 1228 до значения 1071). Усредненная погрешность решения $\overline{\varepsilon}$ уменьшается с ростом этого параметра примерно на три порядка (рис. 3.17). С другой стороны, оценка вероятности локализации глобального максимума $\tilde{\xi}$ в тех же условиях уменьшается практически до нуля (рис. 3.18).

Таким образом, как и следовало ожидать, увеличение значений делителя размерного параметра b_r, с одной стороны, повышает точность локализации максимума фитнес-функции, а с другой – уменьшает вероятность его локализации.

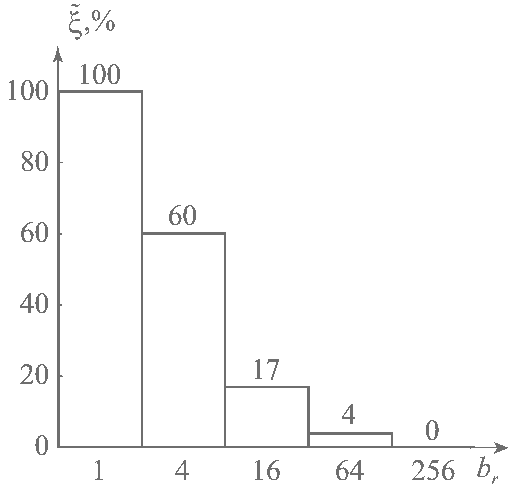


Рис. 3.18. Вероятность $\tilde{\xi}$ в функции величины b_r

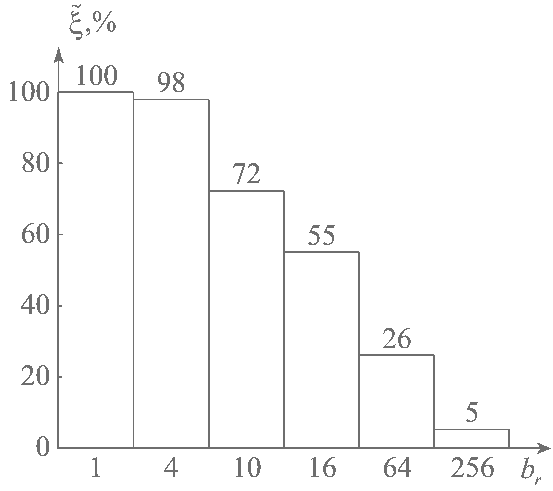


Рис. 3.19. Вероятность $\tilde{\xi}$ в функции величины b_r: модифицированный алгоритм *CIAC*

Эффект модификации алгоритма *CIAC* встряской популяции иллюстрирует рис. 3.19. Здесь и далее, если не оговорено иное, приняты следующие значения свободных параметров:

$$\nu = 1;\ \alpha = 0{,}01;\ \delta_t = |S|;\ \eta = \frac{n_w}{|S|} = 0{,}99.$$

Сравнение рис. 3.18, 3.19 показывает, что модифицированный алгоритм *CIAC* обеспечивает значительное повышение вероятности локализации глобального минимума функции Растригина.

Зависимости вероятности $\tilde{\xi}$ от значений параметров α, ν, η иллюстрируют рис. 3.20 – 3.22. Рис. 3.20 демонстрирует ожидаемое уменьшение вероят-

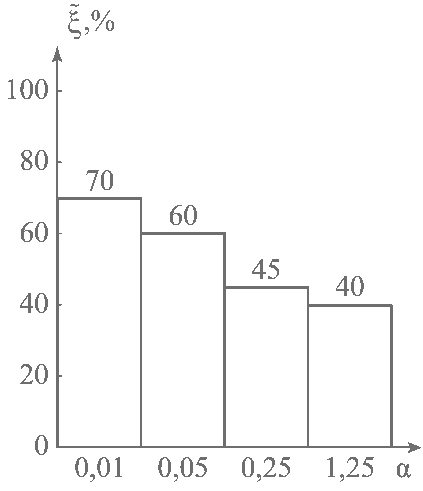


Рис. 3.20. Вероятность $\tilde{\xi}$ в функции величины α: модифицированный алгоритм *CIAC*

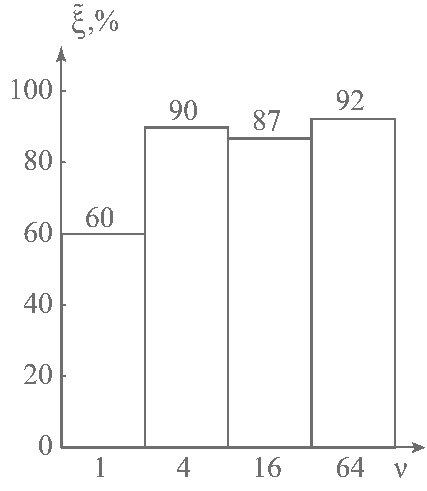


Рис. 3.21. Вероятность $\tilde{\xi}$ в функции величины ν: модифицированный алгоритм *CIAC*; $\alpha = 0{,}025$

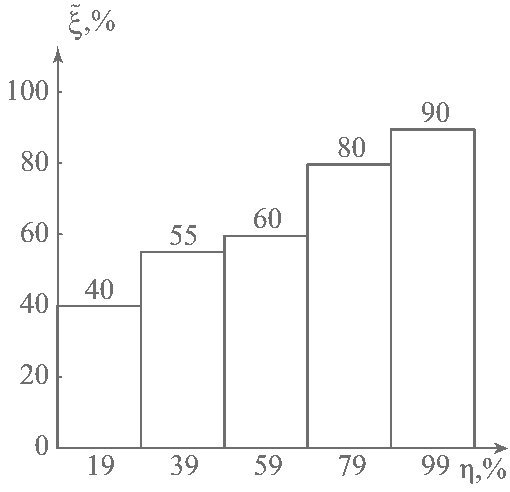


Рис. 3.22. Вероятность $\tilde{\xi}$ в функции величины η: модифицированный алгоритм *CIAC*; $\alpha = 0,025$; $\nu = 4$

ности $\tilde{\xi}$ с ростом значений параметра α, поскольку при этом эффект модификации уменьшается и свойства гибридного алгоритма приближаются к свойствам исходного алгоритма *CIAC*. Рис. 3.21 показывает, что с увеличением значений параметра ν вероятность $\tilde{\xi}$ повышается. Поскольку увеличение этого параметра означает большие начальные значения феромона в новых точках, время его испарения в результате увеличивается, что приводит к увеличению числа испытаний. Характер зависимости вероятности $\tilde{\xi}$ от параметра η, который демонстрирует рис. 3.22, также оказывается вполне ожидаемым – увеличение значений данного параметра диверсифицирует поиск и, как следствие, повышает указанную вероятность •

3.2.4. Непрерывный ортогональный алгоритм муравьиной колонии

Непрерывный ортогональный алгоритм муравьиной колонии (*Continuous Orthogonal Ant Colony*, *COAC*) предложен в работах Ху (*X. Hu*), Чжана (*J. Zhang*) и Ли (*Y. Li*). Алгоритм существенно использует так называемые *ортогональные планы эксперимента*, разработанные в рамках теории планирования экспериментов.

Общая схема алгоритма выглядит следующим образом. После выбора муравьем *региона поиска* он обследует этот регион на основе ортогонального плана эксперимента, позволяющего рационализировать и ускорить поиск. В лучших точках региона поиска (соответствующих максимальному значению фитнес-функции) муравей оставляет феромонные метки. Аналогично алгоритму *CACO*, каждый муравей может воспринять количество феромона в этих метках и на этой основе выбрать новый регион поиска. Количество феромона в точках, принадлежащих регионам, в которых поиск уже закончен, уменьша-

ется. Регионы, количество феромона в которых ниже определенного уровня, исключаются из дальнейшего рассмотрения. Таким образом, алгоритм использует канал стигмергии и не использует прямой канал.

Элементы теории планирования эксперимента

Варьируемые параметры в теории планирования эксперимента называют *факторами*, а исследуемые значения этих параметров – *уровнями факторов*. Оптимизируемая функция (аргументами которой являются факторы) называется *функцией отклика*. Эксперимент, в котором каждый из факторов принимает каждый из своих возможных уровней, называют *полным факторным экспериментом* (ПФЭ).

Рассмотрим для примера функцию отклика $f(X)$, зависящую от $|X|$ факторов $(x_1, x_2, ..., x_{|X|})$. Положим, что эксперимент производится в точке $X_i = (x_{i,1}, x_{i,2}, ..., x_{i,|X|})$ и число уровней каждого из факторов равно $(2l+1)$. Тогда ПФЭ в указанной точке требует вычисления значений функции $f(X)$ в точках $x_{i,j} \pm k\delta_{i,j}$, $j \in [1:|X|]$, $k \in [0:l]$, где $\delta_{i,j}$ – шаг изменения j-го фактора (рис. 3.23). Каждое вычисление значения функции отклика называют *опытом*. Легко видеть, что ПФЭ требует $n = (2l+1)^{|X|}$ опытов.

С помощью соответствующего линейного преобразования факторов X их можно преобразовать таким образом, чтобы трансформированный вектор $\tilde{X}$ в точке X_i имел нулевые значения всех своих компонент, а в остальных точках ПФЭ принимал целочисленные значения, равные $(\pm k)$, $k \in [0:l]$.

Значения, которые каждый из факторов принимает во всех n опытах, образуют $(n \times |X|)$-*матрицу плана эксперимента* P. Например, для ПФЭ, в котором $|X| = 2$ и $l = 1$, эта матрица имеет вид, представленный в табл. 3.2.

Столбец матрицы P, соответствующий j-му фактору, обозначаем P_j, $j \in [1:|X|]$.

Определение 3.1. Матрица плана эксперимента P называется ортогональной, если для ее двух любых несовпадающих столбцов P_{j_1}, P_{j_2} имеет место равенство $(P_{j_1}, P_{j_2}) = 0$, где $(\cdot,\cdot)$ – символ скалярного произведения векторов.

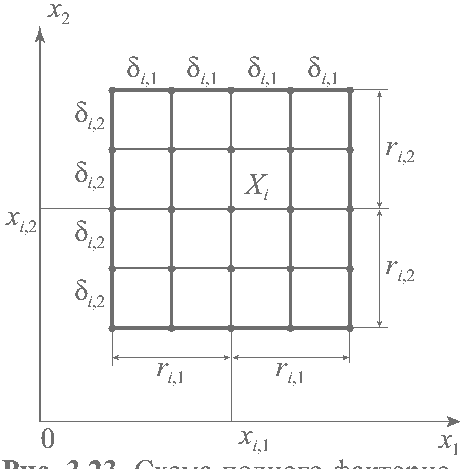


Рис. 3.23. Схема полного факторного эксперимента: $|X| = 2$; $l = 2$; $n = 25$

Легко видеть, что матрица плана эксперимента, представленная в табл. 3.2, является ортогональной.

При высокой размерности пространства поиска ПФЭ требует очень большого числа опытов (даже при малом числе факторных уровней). Поэтому в вычислительной практике обычно используют так называемый *дробный факторный эксперимент* (ДФЭ), суть которого состоит в исключении из матрицы P некоторых строк таким образом, чтобы получившаяся матрица была ортогональной, и число строк в ней было равно или превышало число факторов. Известно большое число алгоритмов выбора исключаемых строк,

Таблица 3.2 **Пример матрицы плана ПФЭ: $|X| = 2$; $l = 1$**

Номер опыта	Фактор		Номер опыта	Фактор	
	$\tilde{x}_1$	$\tilde{x}_2$		$\tilde{x}_1$	$\tilde{x}_2$
1	–1	–1	6	1	0
2	0	–1	7	–1	1
3	1	–1	8	0	1
4	–1	0	9	1	1
5	0	0			

обеспечивающих оптимальность получающейся матрицы плана эксперимента по некоторым критериям. Из матрицы, которую представляет табл. 3.2, ортогональную матрицу ДФЭ можно получить, к примеру, путем исключения всех строк, содержащих один ноль.

Схема алгоритма СОАС

Текущее множество регионов поиска обозначим $A = (A_i, i \in [1:|A|])$, где регион поиска A_i определяет набор

$$\left\langle X_i, R_i, \boldsymbol{X}_i, \Theta_i, X_i^*, \theta_i^*, \varphi_i^*, \rho_i^*, n_i^* \right\rangle.$$

Здесь X_i – вектор координат центра региона; $R_i = (r_{i,1}, r_{i,2}, ..., r_{i,|X|})$ – вектор радиусов региона (положительные величины); $\boldsymbol{X}_i = \{X_{i,1}, X_{i,2}, ..., X_{i,|\Phi_i|}\}$ – множество векторов координат активных феромонных точек Φ_i, принадлежащих данному региону; $\Theta_i = \{\theta_{i,1}, \theta_{i,2}, ..., \theta_{i,|\Phi_i|}\}$ – множество соответствующих количеств феромона; $X_i^* \in \boldsymbol{X}_i$ – вектор координат наиболее активной феромонной точки региона; θ_i^* – соответствующее количество феромона; φ_i^* – наилучшее значение фитнес-функции активных феромонных точек региона; ρ_i^* – ранг региона (см. ниже); n_i^* – общее число посещений муравьями данного региона. Таким образом, региону A_i соответствует гиперпараллелепипед (рис. 3.24)

$$\Pi_i = \{(x_{i,j} - r_{i,j}) \le x_{i,j} \le (x_{i,j} + r_{i,j}),\ j \in [1:|X|]\}.$$

Начальные положения центров регионов поиска $X(0) = (X_i(0), i \in [1:|A|])$ являются случайными, равномерно распределенными в области поиска D. Начальные радиусы регионов $r_{i,1}(0), r_{i,2}(0), ..., r_{i,|X|}(0)$ представляют собой случайные величины, равномерно распределенные в интервале $[-r^0; r^0]$, где $r^0 > 0$ – свободный параметр алгоритма. Другими словами, $r_{i,j}(0) = U_1(-r^0; r^0)$.

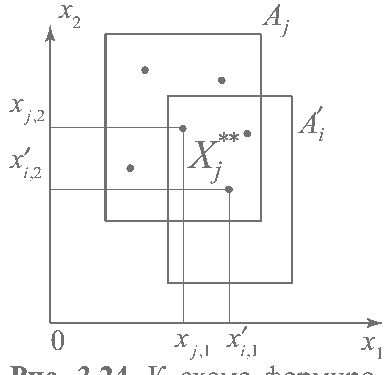


Рис. 3.24. К схеме формирования новых регионов поиска: $|X| = 2$

На текущей итерации для муравья s_i, $i \in [1:|S|]$ новый регион A_i' формируем по следующему правилу.

1) Генерируем случайное число $u = U_1(0;1)$. Если $u \le b_A$, то выбираем тот еще не занятый регион A_j, которому принадлежит точка X_j^{**}, соответствующая наиболее интенсивной феромонной метке:

$$\max_k \theta_k^* = \theta_j^{**}, \; k \in [1:|A|].$$

В противном случае выбираем регион по правилу рулетки с вероятностью

$$\xi_i = \frac{\theta_i^*}{\sum_k^{|\Phi_j|} \theta_k^*}, \; i \in [1:|S|].$$

2) Строим новый регион A_i', используя в качестве его центра точку X_j^{**} (рис. 3.24). Адаптивные радиусы региона $r_{i,j}$, $j \in [1:|X|]$ определяем по схеме, рассмотренной ниже.

Здесь b_A – свободный параметр алгоритма, имеющий смысл вероятности выбора муравьем s_i региона A_i.

После формирования региона поиска A_i' строим $(n \times |X|)$-матрицу плана ПФЭ, соответствующего данному региону, и случайным образом выбираем $\nu \ge |X|$ строк, формируя таким образом $(\nu \times |X|)$-матрицу P' плана ДФЭ. Матрица определяет подлежащие обследованию точки региона A_i'. Число уровней факторов обычно принимают равным трем (когда $l = 1$), так что $n = 3^{|X|}$.

Последовательно перемещаем муравья во все точки, которые определяет матрица P', и вычисляем в каждой из них значение фитнес-функции. Если лучшая точка региона не меняется в течение фиксированного числа итераций, то в качестве центра нового региона используем другую случайно выбранную точку текущего региона.

Если в результате ортогонального обследования региона поиска A_i' значение фитнес-функции удалось улучшить, то радиусы региона увеличиваем с тем, чтобы разнообразить поиск в «удачном» месте. В противном случае радиусы региона уменьшаем, чтобы детальнее исследовать окрестность найденной лучшей точки. Уменьшение радиусов осуществляем по формуле

$$r_{i,j}' = \begin{cases} b_r r_{i,j}, & r_{i,j} \ge b_r^{\max}, \\ U_1(b_r^{\max}; 2r^0), & \text{иначе}, \end{cases}$$

а увеличение – по формуле

$$r_{i,j}' = \frac{r_{i,j}}{b_r}; \; i \in [1:|S|], \; j \in [1:|X|].$$

Здесь $b_r, b_r^{\max}$ – свободные параметры алгоритма (см. ниже).

При сходимости алгоритма к глобальному решению с увеличением числа итераций частота уменьшения радиуса возрастает, а частота его увеличения – уменьшается. В результате в конце концов поиск сосредотачивается в одном или нескольких регионах с потенциально лучшими точками.

Алгоритм *COAC* использует еще так называемую *процедуру модуляции* (*modulation*), которая заключается в следующем.

1) Текущие регионы A_i, $i \in [1:|A|]$ упорядочиваем в виде списка в порядке убывания их приспособленности. Ранг региона ρ_i^* полагаем равным его номеру в указанном списке.

2) Первые $b_\rho |A|$ регионов списка переносим в формируемое множество регионов A', где $b_\rho \in [0;1]$ – *параметр элитарности* (*elitist rate*).

3) Изменяем количество феромона в активных точках региона A_i', $i \in [1:b_\rho |A|]$, по формуле

$$\theta_i' = (1-b_\theta)\theta_i + b_\theta \theta_{init}\left(b_\rho |A| + 1 - \rho_i^* + n_i^*\right).$$

4) Оставшиеся во множестве A' не определенными $(1-b_\rho)|A|$ регионов заменяем новыми случайными регионами по схеме их начальной инициализации. Количество феромона в феромонных точках, оказавшихся в этих регионах, полагаем равным θ_{init}.

Кроме параметров, используемых алгоритмом *CACO*, алгоритм *COAC* содержит следующие свободные параметры:

r^0 – величина, определяющая начальные радиусы регионов;

$b_A \in (0;1]$ – порог выбора нового региона поиска (рекомендуемое значение $b_A \in [0{,}3;0{,}5]$);

$\nu \geq |X|$ – число строк в используемом плане ДФЭ;

$b_r, b_r^{\max}$ – параметры, определяющие закон уменьшения радиусов регионов (рекомендованное значение $b_r \in [0{,}2;0{,}9]$);

b_ρ – параметр элитарности (рекомендованное значение $b_\rho = 0{,}1$).

В зависимости от сложности фитнес-функции в алгоритме *COAC* рекомендуют использовать популяции с числом муравьев, равным $|S| \in [20:100]$, и отношение числа регионов $|A|$ к числу муравьев $|S|$ – в диапазоне от единицы до двух.

Известны результаты исследования эффективности алгоритма *COAC*, показывающие, что в сравнении с алгоритмом *CACO* данный алгоритм обеспечивает в большинстве случаев значительные преимущества по точности локализации глобального максимума и скорости сходимости.

3.2.5. Гибридный алгоритм непрерывно взаимодействующей муравьиной колонии

Гибридный алгоритм непрерывно взаимодействующей муравьиной колонии (*Hybrid Continuous Interacting Ant Colony*, *HCIAC*) является модификацией алгоритма *CIAC*, призванной исправить некоторые недостатки последнего и повысить его эффективность. Алгоритм *HCIAC* развит в работах Дрео (*J. Dreo*) и Сиарри (*P. Siarry*), авторов алгоритма *CIAC*. Алгоритм использует комбинацию глобального поиска муравьиным алгоритмом и локального поиска классическим алгоритмом Нелдера – Мида (алгоритмом деформируемого многогранника).

В случае медленного испарения феромона (малой величины параметра b_θ) алгоритм *CIAC* может породить чрезмерно большое число активных феромонных точек, требующих значительного объема компьютерной памяти для их хранения и высоких вычислительных затрат для обработки. В алгоритме *HCIAC* привлекательность того или иного региона поиска основана не на числе феромонных точек, а на концентрации феромона в нем. С этой целью муравей, вместо того чтобы оставлять новую феромонную метку, может повысить концентрацию феромона в уже существующей «близкой» точке. Близость точки определяет принадлежность ее к так называемой *зоне видимости* (*visible zone*), радиус которой адаптивно уменьшается с ростом числа итераций.

Алгоритм Нелдера – Мида

Алгоритм локальной безусловной оптимизации *Нелдера – Мида* (*J. Nelder*, *R. Mead*) основан на использовании следующих операций над симплексами (вообще говоря, нерегулярными): отражение; редукция; сжатие; растяжение.

Отражение вершины симплекса относительно центра тяжести его остальных вершин трансформирует исходный симплекс $C = \{X_i, i \in [1:(|X|+1)]\}$ в новый симплекс

$$C' = \{X'_i = X_i,\ i \in [1:(|X|+1)],\ i \neq j;\ X'_j = X_c + \alpha\left(X_c - X_j\right)\},$$

где X_j – отражаемая вершина; α – коэффициент отражения;

$$X_c = \frac{1}{|X|}\sum_{i=1,\, i\neq j}^{|X|+1} X_i\ -$$

вектор координат центра тяжести остальных вершин симплекса C (за исключением вершины X_j).

Редукция симплекса означает пропорциональное уменьшение длин всех его ребер. В результате редукции вершин симплекса C к вершине X_j получаем симплекс

$$C' = \{X'_i = X_j + \beta(X_i - X_j),\ i \in [1:(|X|+1)],\ i \neq j;\ X'_j = X_j\},$$

где β – коэффициент редукции.

Сжатие симплекса C в направлении $(X_j - X_c)$ порождает симплекс

$$C' = \{X'_i = X_i,\ i \in [1:(|X|+1)],\ i \neq j,\ X'_j = X_c + \gamma(X_j - X_c)\},$$

где γ – коэффициент сжатия.

Растяжение симплекса C в направлении $(X_j - X_c)$ продуцирует симплекс

$$C' = \{X'_i = X_i,\ i \in [1:(|X|+1)],\ i \neq j,\ X'_j = X_c + \delta(X_j - X_c)\},$$

где δ – коэффициент растяжения.

Упрощенная схема алгоритма Нелдера – Мида имеет следующий вид.

1) Задаем начальную точку X_1^0, длину ребра симплекса ρ и полагаем счетчик числа итераций $t = 0$. Находим координаты X_i^0, $i \in [2:(|X|+1)]$ всех вершин регулярного симплекса C^0 с длиной ребра ρ.

2) Вычисляем значения фитнес-функции $\varphi(X_i^t)$ во всех вершинах симплекса C^t.

3) Среди вершин симплекса C^t находим вершины X_l^t, X_h^t, в которых функция $\varphi(X)$ принимает, соответственно, наименьшее и наибольшее значения, а также находим значения функции $\varphi(X)$ в этих точках

$$\varphi(X_l^t) = \min_{i \in [1:|X|+1]} \varphi(X_i^t) = \varphi_l^t, \ \varphi(X_h^t) = \max_{i \in [1:|X|+1]} \varphi(X_i^t) = \varphi_h^t.$$

4) *Отражение*. Отражаем вершину X_h^t симплекса C^t относительно центра тяжести остальных его вершин – получаем новый симплекс C^{t+1}. Вычисляем значение функции $\varphi(X)$ в вершине X_h^{t+1} этого симплекса $\varphi(X_h^{t+1})$.

5) Если условие окончания итераций выполнено, то в качестве приближенного значения точки минимума функции $\varphi(X)$ принимаем ту вершину симплекса C^{t+1}, в которой эта функция имеет минимальное значение, и заканчиваем вычисления.

6) Если $\varphi(X_h^{t+1}) \le \varphi(X_h^t)$ и $\varphi(X_h^{t+1}) \le \varphi(X_l^t)$, то переходим к шагу 7 (растяжению). Если $\varphi(X_h^{t+1}) \le \varphi(X_h^t)$, но $\varphi(X_h^{t+1}) \ge \varphi(X_l^t)$, то переходим к шагу 4 (отражение). Если $\varphi(X_h^{t+1}) > \varphi(X_h^t)$, то переходим к шагу 8 (сжатие).

7) *Растяжение* (ситуация $\varphi(X_h^{t+1}) \le \varphi(X_h^t)$ и $\varphi(X_h^{t+1}) \le \varphi(X_l^t)$). Выполняем растяжение симплекса C^{t+1} в направлении $(X_h^{t+1} - X_c^{t+1})$ – получаем симплекс C^{t+2}. Вычисляем значение функции $\varphi(X)$ в вершине X_h^{t+2} этого симплекса. Если $\varphi(X_h^{t+2}) \le \varphi(X_l^t)$, то полагаем $t = t + 2$ и переходим к шагу 4 (отражение) с симплексом C^{t+2}. Иначе полагаем $t = t + 1$ и, не используя результаты растяжения, переходим к шагу 4 (отражение) с симплексом C^t.

8) *Сжатие* (ситуация $\varphi(X_h^{t+1}) > \varphi(X_h^t)$). Выполняем сжатие симплекса C^{t+1} в направлении $(X_h^{t+1} - X_c^t)$ – получаем симплекс C^{t+2}. Вычисляем значение функции $\varphi(X)$ в вершине X_h^{t+2} этого симплекса. Если $\varphi(X_h^{t+2}) \le \varphi(X_h^t)$, то полагаем $t = t + 2$ и переходим к шагу 4 (отражение) с симплексом C^{t+2}, иначе переходим к шагу 9 (редукция) с симплексом C^t.

9) *Редукция* (ситуация, когда сжатие симплекса оказалось неудачным). Выполняем редукцию симплекса C^t к вершине X_l^t – получаем новый симплекс C^{t+1}. Вычисляем значение функции $\varphi(X)$ во всех новых вершинах симплекса C^{t+1}. Полагаем $t = t + 1$ и переходим к шагу 4 (отражение).

В качестве условия окончания итераций в алгоритме Нелдера – Мида можно использовать, например, условие

$$\max_{\substack{i,j \in [1:|X|+1] \\ i \ne j}} \left| \varphi(X_i^{t+1}) - \varphi(X_j^{t+1}) \right| \le \varepsilon_\varphi,$$

где ε_φ – требуемая точность решения по φ. Можно также завершить итерации, когда длина максимального из ребер текущего симплекса станет меньше или равна ε_X – требуемой точности решения по X.

Изложенная схема алгоритма Нелдера – Мида имеет тот недостаток, что для сильно овражных функций может происходить вырождение (*сплющивание*) симплекса. Поэтому к рассмотренной схеме алгоритма обычно добавляют этап периодического (через некоторое число итераций n) *восстановления симплекса*, который заключается в следующем: в текущем симплексе выбираем две лучшие вершины и определяем расстояние между ними $\tilde{\rho}$; исходя из лучшей вершины текущего симплекса строим новый симплекс, длину ребра которого принимаем равной $\tilde{\rho}$.

Алгоритм Нелдера – Мида следует классифицировать как детерминированный прямой итерационный одношаговый алгоритм последовательного поиска. Свободными параметрами алгоритма Нелдера – Мида являются $\alpha = 1$ – коэффициент отражения, $\beta \in (0,1)$ – коэффициент редукции (рекомендованное значение $\beta \approx 0,5$), $\gamma \in (0,1)$ – коэффициент сжатия (рекомендованное значение $\gamma \approx 0,4 \div 0,6$), δ – коэффициент растяжения (рекомендованное значение $\delta \approx 2,8 \div 3,0$).

Схема алгоритма HCIAC

Известны два варианта алгоритма *HCIAC*. Первый вариант предполагает, что локальный поиск с помощью алгоритма Нелдера – Мида запускается через фиксированное число итераций, независимо от текущего состояния каждого из муравьев. Назовем этот вариант гибридизации *простым алгоритмом HCIAC*. Второй вариант использует асинхронный запуск локального поиска в зависимости от состояния муравьев и называется *децентрализованным алгоритмом HCIAC*. Отметим, что на основе последнего алгоритма могут быть построены его эффективные параллельные реализации.

Концепция простого алгоритма *HCIAC* противоречит парадигме муравьиной оптимизации, поскольку предполагает некоторое централизованное управление популяцией. Децентрализованный алгоритм, напротив, не разрушает парадигму муравьиных алгоритмов. Ограничимся поэтому рассмотрением только этого алгоритма.

Текущую зону видимости A_i муравья s_i определяет набор

$$\left\langle X_i, r_i, \boldsymbol{X}_i, \Theta_i, X_i^*, \theta_i^*, \varphi_i^* \right\rangle, \quad i \in [1:|S|],$$

где X_i – вектор координат центра зоны видимости (текущего положения муравья s_i); r_i – радиус зоны видимости (положительная величина); $X_i = \{X_{i,1}, X_{i,2}, ..., X_{i,|\Phi_i|}\}$ – множество векторов координат активных феромонных точек Φ_i, принадлежащих данной зоне видимости; $\Theta_i = \{\theta_{i,1}, \theta_{i,2}, ..., \theta_{i,|\Phi_i|}\}$ – множество соответствующих количеств феромона; $X_i^* \in \boldsymbol{X}_i$ – вектор координат наиболее интенсивной феромонной точки зоны; θ_i^* – соответствующее количество феромона; φ_i^* – наилучшее значение фитнес-функции данной зоны. Таким образом, зоне A_i, $i \in [1:|S|]$, соответствует гипершар $D_i = \{X \mid \|X - X_i\| \le r_i)\}$.

Этап инициализации. На этапе инициализации популяции муравьев кроме генерации их начальных положений определяем начальные радиусы $r_i(0)$ зон видимости A_i, $i \in [1:|S|]$. Центры этих зон, полагается, совпадают с положениями соответствующих муравьев. Радиусы $r_i(0)$ подчинены нормальному зако-

ну распределения $N_1(m_r, \sigma_r)$ с математическим ожиданием m_r и средним квадратичным отклонением σ_r.

Этап глобального поиска. В первую очередь, на каждой итерации каждый из муравьев s_i индивидуально, ориентируясь только на доступную ему информацию, принимает решение об использовании одного из двух каналов – стигмергии или прямого канала.

Решение принимаем в соответствии с *функцией выбора* (*stimulus-response*) вида

$$\xi(e) = \frac{1}{1 + \exp(b_s(-e + b_h))}, \tag{3.13}$$

где $e \in [0;1]$ – *привлекательность* канала (в оригинале – *stimulus*); b_s – степень *гладкости* (*smoothness*); b_h – величина *порога чувствительности*. Значения параметров b_s, b_h обычно полагают одинаковыми для всех муравьев популяции.

Задача выбора канала стигмергии или прямого канала сводится к вычислению их привлекательности. Положим, к примеру, что для данного муравья привлекательность канала стигмергии (феромонного следа) $e = 0,5$ и порог чувствительности $b_h = 0,3$. Тогда при $b_s = 10$ вероятность выбора данного канала $\xi(e) = 0,88$.

Если выбран канал стигмергии, то в зоне видимости A_i, $i \in [1:|S|]$ муравья s_i находим все активные феромонные точки $\boldsymbol{X}_i$ и по схеме алгоритма *CIAC* перемещаем муравья в направлении *центра их тяжести*. Если такие точки отсутствуют, то переходим к следующему этапу.

В случае, если выбран канал прямого взаимодействия, из памяти муравья s_i выбираем сообщение и опять же по схеме алгоритма *CIAC* перемещаем муравья в направлении точки, указанной в сообщении. Если сообщения в памяти данного муравья отсутствуют, то также переходим к следующему этапу.

Этап локального поиска. Если в зоне видимости A_i, $i \in [1:|S|]$ муравья s_i нет активных феромонных точек ($|\Phi_i| = 0$) и/или в его памяти отсутствуют сообщения, то по формуле вида (3.13) принимаем одно из следующих решений: отправить сообщение другому муравью популяции; перейти к локальному поиску алгоритмом Нелдера – Мида.

Вместо привлекательности e в этом случае используется динамическая величина μ, называемая *мотивацией* (*motivation*). На данной итерации мотивация μ получает приращение, равное δ_μ, в следующих ситуациях: в зоне видимости муравья отсутствуют активные феромонные точки; в памяти муравья отсутствуют сообщения от других муравьев; муравей не находится в стадии локального поиска.

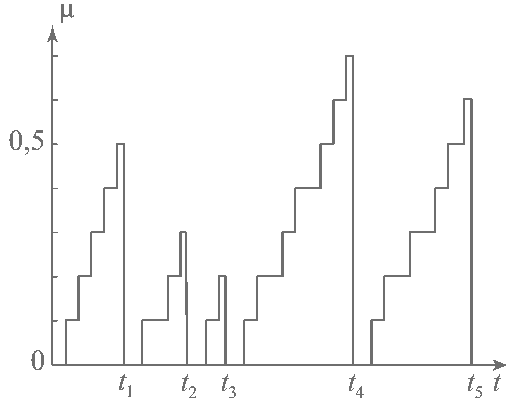


Рис. 3.25. Пример динамики уровня мотивации: $\delta_\mu = 0,1$

Динамику мотивации μ иллюстрирует рис. 3.25, где t_1, $t_2, \ldots, t_5$ – номера итераций, на которых имел место *сброс*

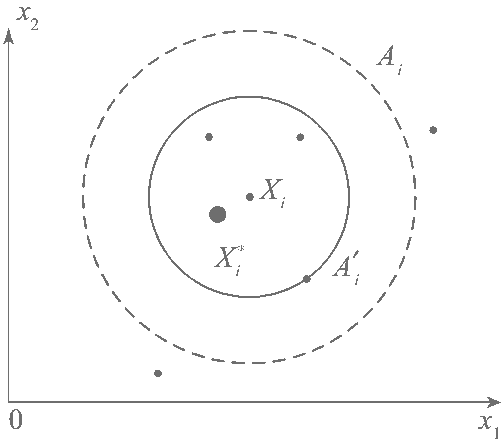


Рис. 3.26. К определению новой зоны видимости: выделена усиленная феромонная точка

мотивации μ в ноль. В соответствии с вероятностным характером принятия решения о начале локального поиска (и сбрасывания мотивации) значения величины μ, при котором начинается этот поиск, различны.

Если выбран прямой канал, то по схеме алгоритма *CIAC* отправляем сообщение случайному муравью популяции и мотивацию данного муравья s_i увеличиваем на величину δ_μ.

В случае если выбор был сделан в пользу локального поиска, то исходя из текущего положения муравья s_i запускаем поиск алгоритмом Нелдера – Мида. Длину ребра исходного симплекса $C = \{X_i, i \in [1:(|X|+1)]\}$ принимаем равной текущему радиусу зоны видимости муравья r_i. Итерации заканчиваем после $n_{\max}$ оценок фитнес-функции. В качестве конечного положения муравья принимаем положение, соответствующее лучшему достигнутому значению фитнес-функции.

После завершения итераций алгоритма Нелдера – Мида отыскиваем активные феромонные точки, находящиеся в зоне видимости A_i муравья s_i. Если хотя бы одна такая точка имеется, выполняем следующие действия: количество феромона θ_i в наиболее активной из найденных точек усиливаем по формуле $\theta'_i = \theta_i + \frac{\theta_i}{2}$; текущий радиус зоны видимости муравья r_i уменьшаем до величины, равной расстоянию муравья до наиболее дальней точки зоны видимости A_i (рис. 3.26).

В случае, когда таких точек в зоне видимости A_i нет, формируем феромонную метку в точке расположения муравья. Количество феромона в этой точке принимаем пропорциональным соответствующему значению фитнес-функции в ней.

И в том, и в другом случае финальным шагом является сброс мотивации μ_i муравья s_i в ноль, а также, возможно, случайное перемещение его в пределах зоны видимости.

Этап завершения итераций предполагает, как всегда, проверку выполнения условия окончания итераций. В случае невыполнения этого условия осуществляем возврат к этапу глобального поиска, а в случае выполнения – завершаем вычисления. В качестве указанного условия в оригинальном варианте алгоритма используют невозможность улучшить значение фитнес-функции после заданного числа испытаний.

Дополнительно к свободным параметрам алгоритма *CIAC* свободными являются следующие параметры алгоритма *HCIAC*:

α, β, γ, δ – коэффициенты отражения, редукции, сжатия и растяжения симплекса соответственно (рекомендуемые значения приведены выше);

m_r, σ_r – математическое ожидание и среднее квадратичное отклонение радиусов зон видимости соответственно;

b_s, b_h – степень гладкости и величина порога чувствительности соответственно;

δ_μ – приращение мотивации μ;

$n_{\max}$ – максимально допустимое число оценок фитнес-функции (число испытаний).

Выполненное авторами алгоритма *HCIAC* исследование его эффективности показывает, что на широком классе тестовых функций по сравнению с алгоритмом *CIAC* алгоритм *HCIAC* имеет значительные преимущества по скорости сходимости и точности локализации глобального экстремума.

Вариантом алгоритма *HCIAC* является *динамический гибридный алгоритм непрерывно взаимодействующей муравьиной колонии* (*Dynamic Hybrid Continuous Interacting Ant Colony*, *DHCIAC*). В отличие от всех рассмотренных ранее популяционных алгоритмов оптимизации, алгоритм *DHCIAC* предназначен для решения задачи *непрерывной динамической* (*dynamic continuous*) *оптимизации*, когда кроме вектора варьируемых параметров X непосредственно или косвенно аргументом целевой функции является еще и время τ. В общем виде *динамическая целевая функция* (*dynamic objective function*) имеет вид

$$f(X,\tau) = f_s(X, f_\tau(\tau)), \tag{3.14}$$

где f_s – *структурная* (*structure*), а f_τ – *временная* (*time*) *функции*.

Известны две версии алгоритма *DHCIAC* – *DHCIACfind* и *DHCIACtrack*. Наиболее существенное отличие этих версий друг от друга заключается в том, что первая из них, подобно базовому алгоритму *HCIAC*, реализует локальный поиск с помощью алгоритма Нелдера – Мида, в то время как вторая версия – с помощью модифицированного варианта симплекс-алгоритма, названного авторами *алгоритмом динамического симплекса* (*dynamic simplex*).

Алгоритм динамического симплекса использует только операцию отражения симплекса, тем самым сохраняя его размер фиксированным. Схема алгоритма для задачи максимизации имеет следующий вид.

1) По схеме базового алгоритма *HCIAC* определяем симплекс $C_i^0 = \{X_{i,j}^0, j \in [1:(|X|+1)]\}$.

2) Путем отражения вершины с минимальным значением фитнес-функции последовательно получаем симплексы C_i^k, $k \in [0:n]$, где n – натуральный положительный свободный параметр алгоритма.

3) Для каждого из $(n+1)$ указанных симплексов определяем среднее значение фитнес-функции

$$\overline{\varphi}_i^k = \frac{1}{|X|+1} \sum_{j=1}^{|X|+1} \varphi(X_{i,j}^k).$$

4) В качестве начального симплекса для следующего шага локального поиска выбираем симплекс C_i^l, соответствующий максимальному значению величины $\overline{\varphi}_i^k$:

$$\max_{k \in [0:n]} \overline{\varphi}_i^k = \overline{\varphi}_i^l.$$

5) Если максимально допустимое число испытаний не достигнуто, переходим к шагу 2 с симплексом C_i^l в качестве начального симплекса. В противном случае заканчиваем локальный поиск.

3.2.6. Пример применения модифицированного алгоритма *CIAC*

Рассмотрим силовую часть электрического привода типа *генератор-двигатель*, предназначенного для управления некоторым рабочим механизмом (рис. 3.27).

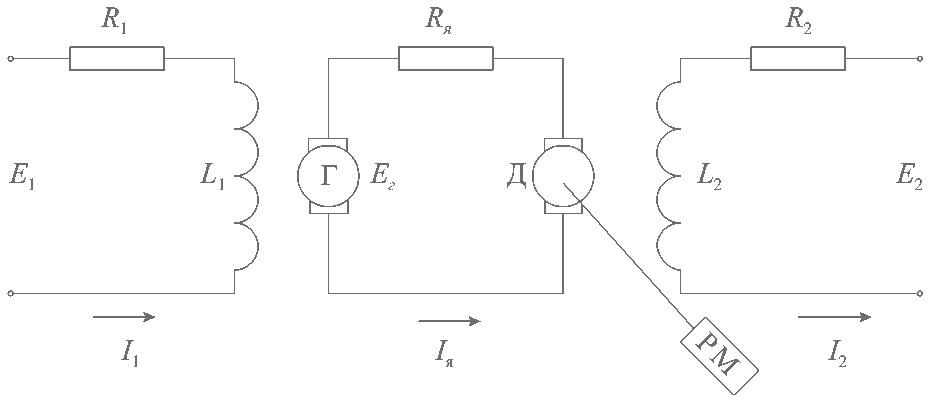


Рис. 3.27. Схема силовой части электрического привода типа генератор-двигатель: Г – генератор; Д – двигатель; РМ – рабочий механизм

Динамику системы описываем системой алгебро-дифференциальных уравнений:

$$\begin{cases} J\,\ddot{\psi} = M_д - M_c, & \psi(0) = \psi^0, \quad \dot{\psi}(0) = \dot{\psi}^0, \\ L_1\,\dot{I}_1 = E_1 - I_1\,R_1, & I_1(0) = I_1^0, \\ L_2\,\dot{I}_2 = E_2 - I_2\,R_2, & I_2(0) = I_2^0, \end{cases} \tag{3.15}$$

$$I_я = \frac{\Phi_г(I_1) - cI_2\dot{\psi}}{R_я}, \tag{3.16}$$

где приняты следующие обозначения:

J – момент инерции якоря двигателя и приводимого им в движение рабочего механизма;

$R_я$ – сопротивление якоря двигателя;

ψ – угол поворота вала двигателя;

$M_д$ – момент, развиваемый двигателем;

M_c – момент нагрузки;

L_1, R_1, I_1, E_1 – индуктивность обмотки возбуждения генератора, его омическое сопротивление, ток и электродвижущая сила (ЭДС) соответственно;

L_2, R_2, I_2, E_2 – аналогичные величины обмотки возбуждения двигателя;

$E_г$ – ЭДС в контуре якоря двигателя, связанная с током в его обмотке возбуждения выражением $E_г = \Phi_г(I_1)$;

$E_д = cI_2\dot{\psi}$ – ЭДС в контуре якоря двигателя, где $c = const$ – коэффициент пропорциональности.

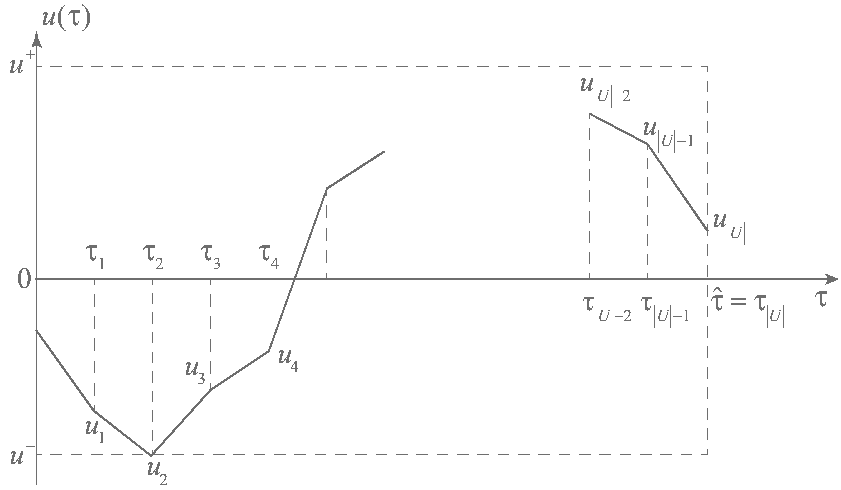


Рис. 3.28. Кусочно-линейная аппроксимация управления $E_1(\tau) = u(\tau)$

Во-первых, при заданных начальных условиях требуется найти закон изменения во времени ЭДС $E_1(\tau) = E_1^*(\tau)$, который минимизирует значение функционала

$$F_1(E_1) = \int_0^{\hat{\tau}} \left(I_я^2(E_1,\tau)R_я + I_1^2(E_1,\tau)R_1 + I_2^2(E_1,\tau)R_2 \right) d\tau\,, \qquad (3.17)$$

имеющего смысл суммарных затрат энергии на перемещение якоря двигателя за период времени $[0;\hat{\tau}]$.

Во-вторых, функция $E_1^*(\tau)$ должна обеспечивать поворот якоря двигателя на требуемый угол $\psi_{\max}$ за время $[0;\hat{\tau}]$. Другими словами, эта функция должна обеспечивать минимальное значение функционала

$$F_2(E_1) = abs\left(\psi(E_1) - \psi_{\max}\right). \qquad (3.18)$$

В-третьих, функция $E_1^*(\tau)$ должна обеспечивать нулевую угловую скорость якоря двигателя в момент времени $\hat{\tau}$, т. е. минимизировать функционал

$$F_3(E_1) = abs\left(\dot{\psi}(E_1)\right). \qquad (3.19)$$

Таким образом, имеем трехкритериальную задачу оптимального управления: найти допустимое управление $E_1^*(\tau) \in D_E$, которое на решениях системы уравнений (3.15), (3.16) обеспечивает минимум критериев оптимальности (3.17) – (3.19).

Подобно примеру 2.13, сведем эту задачу к задаче нелинейного программирования. Обозначим для простоты записи $E_1(\tau) = u(\tau)$ и положим, что множество допустимых управлений D_u определяют двухсторонние ограничения вида $u^- \le u(\tau) \le u^+$. Покроем интервал $[0;\hat{\tau}]$ равномерной сеткой, узлы которой обозначим τ_i, $i \in [0:|U|]$. Будем искать оптимальное управление $u^*(\tau)$ в классе кусочно-линейных функций (рис. 3.28).

Обозначив $U=(u_1,u_2,...,u_{|U|})$ – $(|U|\times 1)$-вектор, где $u_i=u(\tau_i)$, $i\in[1:|U|]$, получим трехкритериальную задачу вида

$$\min_{U\in D_U} F_j(U)=F_j(U^*),\ j=1,2,3, \tag{3.20}$$

где $D_U=\{u^-\le u_i\le u^+, i\in[1:|U|]\}$.

Переведем критерии оптимальности $F_1(U)$, $F_2(U)$ в ограничения с помощью алгоритма штрафных функций, то есть перейдем от задачи (3.20) к одноцелевой задаче глобальной безусловной оптимизации

$$\min_{U\in R^{|U|}} \varphi(U)=\min_{U\in D_U}\left(F_1(U)+\alpha_1F_2(U)+\alpha_2F_3(U)\right), \tag{3.21}$$

где α_1,α_2 – коэффициенты штрафа.

Примем следующие значения параметров задачи: J = 0,425 кг·м^2; $R_я$ = 0,224 Ом; $M_д=k_дI_2I_я$ Н·м, где $k_д=10$ – масштабный коэффициент; M_c=150 Н·м; L_1=8 Гн; R_1=43 Ом; L_2=8 Гн; R_2=43 Ом; E_2 = 40 В; $\Phi_г(I_1)=k_гI_1$, где $k_г=10$ – коэффициент кривой намагничивания; c = 1; $u^-=0$; u^+ = 100 В. Начальные условия примем нулевыми: $\psi^0=\dot\psi^0=0$; $I_1^0=I_2^0=0$. Конечное состояние угла поворота якоря двигателя полагаем равным $\psi_{\max}$ = 210 град. Движение системы рассматриваем на интервале $[0;20]$, т. е. полагаем $\hat\tau=20$ с.

Число узлов сетки, покрывающей интервал $[0;\hat\tau]$, принимаем равным 21, так что размерность вектора варьируемых параметров $|U|=20$. Потребуем удовлетворения ограничений

$$F_2(E_1)=abs\left(\psi(E_1)-\psi_{\max}\right)=0, \tag{3.22}$$

$$F_3(E_1)=abs\left(\dot\psi(E_1)\right)=0 \tag{3.23}$$

с точностями $\varepsilon_1=\varepsilon_2=1,0$. Начальные значения коэффициентов штрафа α_1,α_2 принимаем равными 2,0; 5,0.

Используем для решения задачи глобальной безусловной оптимизации (3.21) модифицированный алгоритм непрерывно взаимодействующей колонии муравьев *CIAC*, включающий в себя процедуру встряски популяции.

Принимаем следующие значения основных свободных параметров алгоритма: размер популяции равен $|S|=300$; максимальное число поколений $\hat i=100$; коэффициент устойчивости феромонных точек b_θ=0,1; минимальное количество феромона в точке $\theta_{\min}$= 0,0001; максимальное число сообщений $m_{\max}=30$; для каждого из муравьев делитель размерного параметра определяет формула

$$b_r=abs\left(N_1(0;\sigma)\right),$$

где среднее квадратичное отклонение $\sigma=13,5$.

Пример начального приближения к решению иллюстрирует рис. 3.29. Указанному приближению соответствуют угол $\psi(\hat\tau)=223,14^\circ$, угловая скорость $\dot\psi(\hat\tau)=2,46$ град/с и расход энергии $F_1(E_1^0)=5636$ Вт.

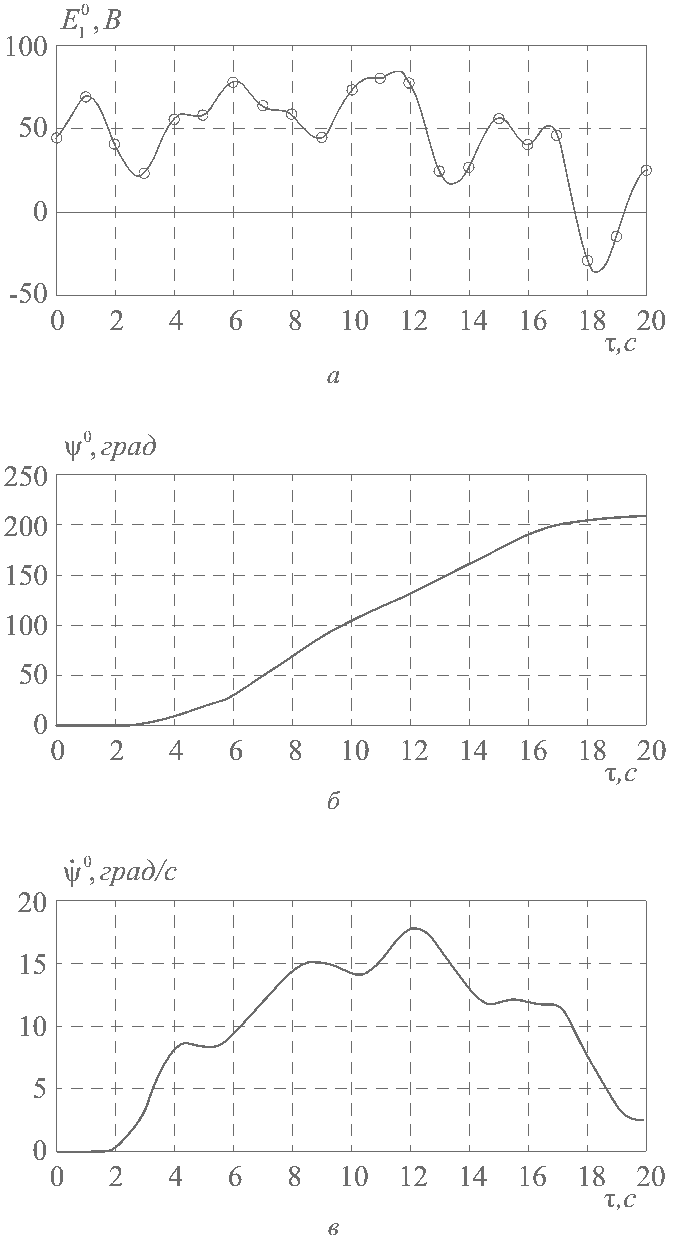


Рис. 3.29. Пример начального приближения к решению $E_1^0(\tau)$ и соответствующие функции $\psi^0(\tau)$, $\dot{\psi}^0(\tau)$: *а* – $E_1^0(\tau)$; *б* – $\psi^0(\tau)$; *в* – $\dot{\psi}^0(\tau)$

Лучшие (с точки зрения достигнутого значения функционала (3.21)) результаты вычислительного эксперимента иллюстрируют рис. 3.30, *а* – *в*: первый рис. 3.30, *а,* показывает найденное оптимальное управление $E_1^*(\tau)$, а два других – соответствующие законы изменения во времени угла поворота якоря двигателя $\psi^*(\tau)$ и его скорости $\dot{\psi}^*(\tau)$.

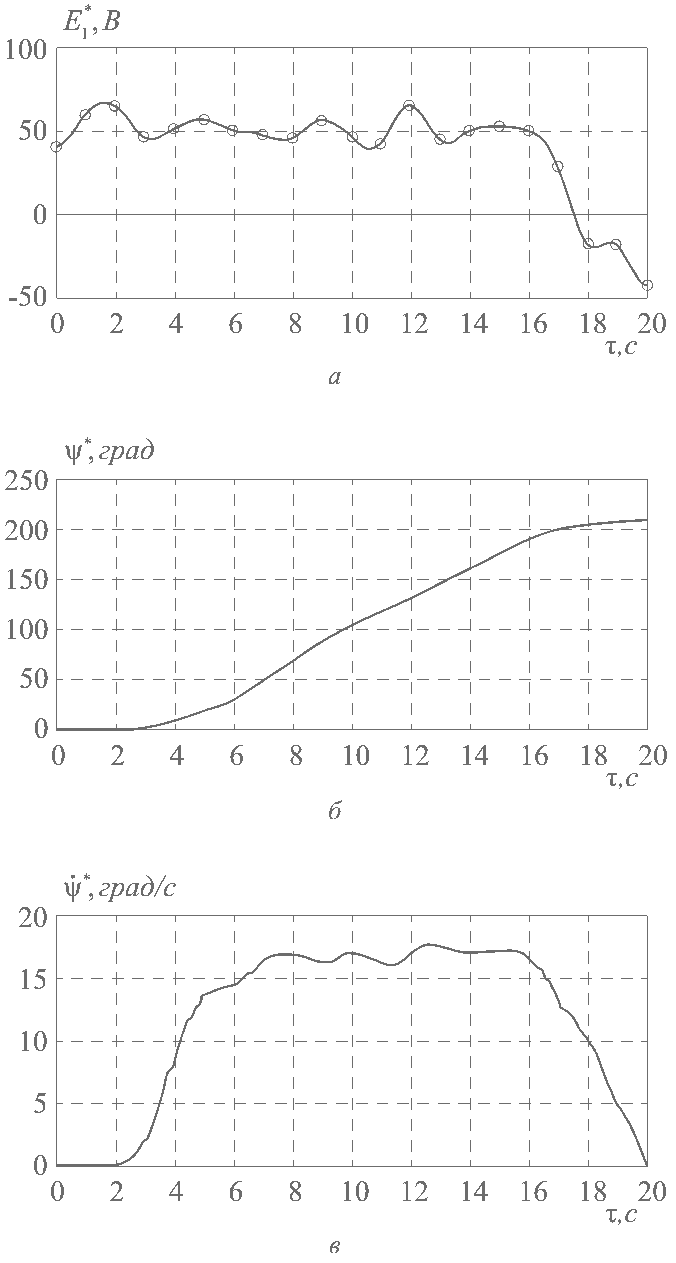


Рис. 3.30. Результаты решения задачи – оптимальная функция $E_1^*(\tau)$ и соответствующие функции $\psi^*(\tau)$, $\dot{\psi}^*(\tau)$: *а* – $E_1^*(\tau)$; *б* – $\psi^*(\tau)$; *в* – $\dot{\psi}^*(\tau)$

Полученному решению задачи соответствуют угол $\psi^*(\hat{\tau}) = 209,7^\circ$, угловая скорость $\dot{\psi}^*(\hat{\tau}) = 0,02$ град/с и расход энергии $F_1(E_1^*) = 3352$ Вт. Таким образом, требование (3.22) выполнено с точностью $0,3^\circ$, требование (3.23) – с точностью 0,02 град/с, а расход энергии уменьшился по сравнению с начальным приближением более чем на 40 %.

3.3. ОПТИМИЗАЦИЯ ПЧЕЛИНЫМ РОЕМ

Алгоритмы пчелиной оптимизации являются одними из новейших многоагентных алгоритмов, основанных на моделировании *роевого интеллекта* (*swarm intelligence*). Первые статьи, в которых излагается алгоритм данного класса, были опубликованы в 2005 г. Основная идея алгоритмов состоит в моделировании поведения пчел при поиске нектара.

Алгоритмы пчелиной оптимизации успешно применяют при решении различных задач дискретной оптимизации – задача календарного планирования, задача коммивояжера, транспортная задача и др. Разработаны также высокоэффективные варианты алгоритма, предназначенные для решения задач непрерывной оптимизации, наиболее известными из которых являются *пчелиный алгоритм* (*Bees algorithm*) и *алгоритм колонии искусственных пчел* (*Artificial Bee Colony*, *ABC*).

Среди недостатков пчелиных алгоритмов упоминания заслуживает большое число свободных параметров.

В этом параграфе представлены бионические предпосылки пчелиных алгоритмов, базовый алгоритм роя пчел, алгоритм *ABC*, а также один из гибридных алгоритмов, построенный на основе алгоритма роя пчел и алгоритма оптимизации путем имитации отжига.

3.3.1. Бионические предпосылки

Радиус области распространения роя *медоносных пчел* (*honey bee swarm*) достигает 10 и более километров, что обеспечивает высокую вероятность выявления большего числа источников нектара. «Стратегию» роя можно определить следующим образом: непрерывный поиск в указанной области цветочных участков, содержащих достаточное для добычи *количество нектара* (*nectar amounts*); посещение цветочных участков, изобилующих нектаром или пыльцой, которые могут быть собраны с меньшими усилиями, большим числом пчел по сравнению с участками, содержащими меньшее количество нектара или пыльцы.

Для описания поведения пчел в природе используют такие основные понятия, как источник нектара, рабочие пчелы, пчелы-разведчики, пчелы-наблюдатели.

Источник нектара, источник пищи (*food sources*) характеризуется своей *полезностью*, которую определяют такие факторы, как удаленность от улья, концентрация нектара, удобство его добычи.

Рабочие пчелы, *занятые фуражиры* (*worker bees*) – пчелы, которые *связаны* с одним из источников нектара, то есть добывают на нем нектар. Занятые фуражиры владеют следующей информацией о «своем» источнике нектара: направление от улья на источник, полезность источника.

Пчелы-разведчики, *незанятые фуражиры*, *пчелы-скауты* (*scout bees*, *scouts*) осуществляют поиск источников нектара для их использования.

Пчелы-наблюдатели (*onlooker bees*) в данное время выполняют некоторые работы в улье.

Каждая незанятая пчела может полететь к источнику нектара, следуя за пчелой-разведчиком, которая нашла путь к такому источнику. Пчела-раз-

ведчик выполняет *вербовку* незанятых пчел с помощью *танца (waggle dance)* на специальной площадке улья – области танцев. *Завербованная* (*recruit*) пчела следует за соответствующей пчелой-разведчиком к области с нектаром и становится, таким образом, занятым фуражиром. Вербовку осуществляют те из пчел-разведчиков, которые нашли участки с полезностью выше некоторого определенного порога.

Занятый фуражир после добычи нектара возвращается в улей и оставляет его там. После этого данный фуражир может выполнить одно из следующих действий: оставить «свой» источник нектара и стать незанятым фуражиром; продолжить заготовку нектара из прежнего источника, не вербуя других пчел; выполнить вербовку.

Пчела выбирает одно из указанных действий по некоторому вероятностному закону.

Одновременно в пределах области танцев разные пчелы могут «рекламировать» различные источники нектара. Механизмы принятия решений, в соответствии с которыми пчела решает следовать за той или иной пчелой-вербовщиком, исследованы недостаточно хорошо. Логично предположить, что вероятность вербовки тем или иным образом определяется полезностью соответствующего источника нектара.

Таким образом, самоорганизация пчелиного роя основывается на четырёх следующих основных механизмах.

• Положительная обратная связь – на основе информации, полученной от других пчел, пчела летит к одному из источников нектара.

• Отрицательная обратная связь – основываясь на информации, полученной от других пчел, данная пчела может решить, что «ее» источник нектара значительно хуже других найденных источников, и оставить этот источник.

• Случайность – вероятностный поиск пчёлами-разведчиками новых источников нектара.

• Множественность взаимодействия – информация об источнике нектара, найденном одной пчелой, передается многим другим пчелам улья.

3.3.2. Пчелиный алгоритм

Пчелиным алгоритмом называют (*B*-алгоритм) *Bees algorithm*, предложенный в работах Фама (*D. T. Pham*) с соавторами в 2005 г. Концептуально, схема *B*-алгоритма имеет следующий вид. Сначала из улья вылетает в случайном направлении некоторое число пчел-разведчиков, которые пытаются отыскать участки, где есть нектар. Через какое-то время пчелы возвращаются в улей и сообщают другим пчелам, где и сколько они нашли нектара. После этого на найденные участки отправляются рекрутированные (рабочие) пчелы, причем чем больше на данном участке предполагается найти нектара, тем больше пчел летит к этому участку. Пчелы-разведчики опять улетают искать другие участки, после чего процесс повторяется.

Положим, что речь идет о задаче максимизации фитнес-функции. Количество нектара в некоторой точке пространства поиска полагаем пропорциональным значению фитнес-функции в этой точке. Одна итерация алгоритма включает в себя следующие основные шаги.

1. В случайные точки пространства поиска отправляем пчел-разведчиков.

2. На основании значений фитнес-функции, вычисленных в указанных точках, выделяем некоторое число *элитных участков* – подобластей пространства поиска, соответствующих максимальным значениям этой функции. Аналогично определяем некоторое число *перспективных участков*, которые соответствуют значениям фитнес-функции, близким к максимальным.

3. На каждый из элитных и перспективных участков посылаем определенное число рабочих пчел. Используя соответствующие значения фитнес-функции, находим новые элитные и перспективные участки. При этом выборе учитываем результаты, полученные как пчелами-разведчиками, так и рабочими пчелами.

В качестве текущего приближения к решению задачи принимаем точку с максимальным значением фитнес-функции. Размеры элитных и перспективных участков уменьшаем с ростом числа итераций, так что на завершающих итерациях поиск ведется только в окрестностях максимумов целевой функции.

Формализуем представленную схему *B*-алгоритма. Рассмотрим задачу глобальной условной максимизации в гиперпараллелепипеде *П*. Среди пчел $S = \{s_i, i \in [1:|S|]\}$ выделяем $|S^o| < |S|$ пчел-разведчиков. Положим для простоты записи, что эти пчелы являются первыми в рое S:

$$S^o = \{s_i^o, i \in [1:|S^o|]\}.$$

Остальные пчелы роя

$$S^w = \left\{s_i^w, i \in \left[1:|S^w|\right]\right\} = \left\{s_i, i \in \left[|S^o|+1:|S|\right]\right\}$$

являются рабочими пчелами; $|S^w| = |S| - |S^o|$. Значения фитнес-функции $\varphi(X)$, соответствующие текущему положению пчел s_i^o, s_i^w, обозначаем $\varphi(X_i^o) = \varphi_i^o$, $\varphi(X_i^w) = \varphi_i^w$.

Множество *элитных* (*top-rated*, *elite*) участков обозначаем $A^b = \{a_j^b, j \in [1:|A^b|]\}$. Аналогично, множество перспективных участков обозначаем $A^p = \{a_k^p, k \in [1:|A^p|]\}$. В оригинальной работе, в которой предложен *B*-алгоритм, совокупность элитных и перспективных участков называют *отобранными* (*selected*).

Полагаем, что участки a_j^b, a_k^p представляют собой гиперпараллелепипеды в пространстве $R^{|X|}$, грани которых параллельны координатным плоскостям, центры находятся в точках X_j^b, X_k^p и длины сторон равны $2r_l^b$, $2r_l^p$ соответственно (рис. 3.31). Здесь $r_l^b, r_l^p, l \in [1:|X|]$ – радиусы участков по соответствующим измерениям пространства поиска.

При формировании элитных участков может оказаться, что евклидово расстояние $\left\|X_{j_1}^b - X_{j_2}^b\right\|_E$ между некоторыми такими участками $a_{j_1}^b$, $a_{j_2}^b$ не превышает некоторой фиксированной свободной величины $\varepsilon_b > 0$:

$$\left\|X_{j_1}^b - X_{j_2}^b\right\|_E \le \varepsilon_b, \; j_1, j_2 \in [1:|A^b|], \; j_1 \ne j_2.$$

В таком случае возможны два следующих варианта действия: ставим в соответствие точкам $X_{j_1}^b$, $X_{j_2}^b$ два различных пересекающихся участка $a_{j_1}^b$, $a_{j_2}^b$; ставим в соответствие этим точкам один участок a_j^b, центр которого находится в точке $X_{j_1}^b$ или точке $X_{j_2}^b$, соответствующей большему значению фитнес-функции, то есть полагаем $\varphi(X_j) = \varphi_j = \max(\varphi_{j_1}, \varphi_{j_2})$.

Аналогичным образом разрешаем ситуацию, когда при формировании участков a_k^p имеет место неравенство

$$\left\| X_{k_1}^p - X_{k_2}^p \right\|_E \le \varepsilon_p, \; k_1, k_2 \in [1 : |A^p|], \; k_1 \ne k_2,$$

где $\varepsilon_p > 0$ – свободный параметр.

Наконец, возможна ситуация, когда малым оказывается евклидово расстояние между некоторыми точками X_j^b, X_k^p. В этом случае также могут быть использованы различные варианты действий, например, точка X_k^p может быть просто исключена из рассмотрения.

Во введенных обозначениях схема *B*-алгоритма имеет следующий вид.

1) Генерируем случайные точки X_i, $i \in [1 : |S^o|]$, равномерно распределенные во множестве *П*, и отправляем в эти точки $|S^o|$ пчел-разведчиков s_i^o. Вычисляем в точках X_i значения фитнес-функции $\varphi(X_i) = \varphi_i^o$, сортируем величины φ_i^o по убыванию и представляем в виде линейного списка.

2) Точки X_j, соответствующие первым $|A^b|$ элементам списка, объявляем центрами X_j^b элитных участков a_j^b, $j \in [1 : |A^b|]$. Аналогично точки X_k, $k \in [1 : |A^p|]$, соответствующие последующим $|A^p|$ элементам списка, объявляем центрами перспективных участков a_k^p.

3) В каждый из a_j^b элитных и a_k^p перспективных участков посылаем соответственно по n_b и n_p пчел-фуражиров $s_i^w \in S^w$, так что

$$n_b |A^b| + n_p |A^p| = |S^w|.$$

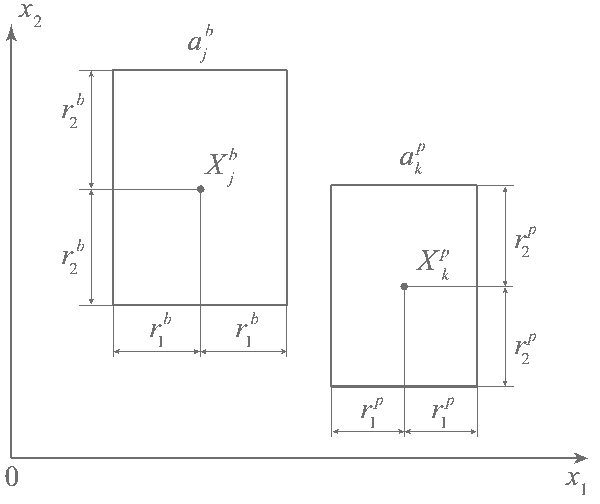


Рис. 3.31. К определению элитных и перспективных участков: $|X| = 2$

Координаты точек, в которые посылаются эти пчелы, полагаем случайными величинами, равномерно распределенными в соответствующих гиперпараллелепипедах a_j^b, a_k^p. Во всех точках, в которые посланы пчелы, вычисляем значения фитнес-функции.

4) Проверяем выполнение условия окончания итераций. Если это условие выполнено, то в качестве решения задачи принимаем точку, соответствующую максимальному достигнутому значению фитнес-функции, и завершаем вычисления. В противном случае, приспособленности всех точек, найденных пчелами-разведчиками и пчелами-фуражирами, сортируем по убыванию и представляем в виде линейного списка. Переходим к шагу 2.

По общему правилу, в качестве условия окончания итераций может быть использовано, например, достижение заданного числа итераций $\hat{t}$ или стагнация вычислительного процесса в течение δ_t итераций.

Кроме общих для популяционных алгоритмов, свободными являются следующие основные параметры алгоритма:

$|S^o|$ – число пчел-разведчиков;

$|A^b|$ – число элитных участков;

$|A^p|$ – число перспективных участков;

n_b – число пчел, отправляемых на элитные участки;

n_p – число пчел, отправляемых на перспективные участки;

r_i^b, r_i^p, $i \in [1:|X|]$ – радиусы участков.

Увеличение всех указанных величин диверсифицирует поиск, но замедляется сходимость алгоритма.

В качестве варианта, в рассмотренной схеме *B*-алгоритма на участки локального поиска a_j^b, a_k^p можно посылать не фиксированные числа пчел, а их числа, пропорциональные значениям фитнес-функции в центрах этих подобластей X_j^b, X_k^p.

Еще один вариант *B*-алгоритма представляет *улучшенный пчелиный алгоритм* (*Improved Bee algorithm*, *IB algorithm*). Основное отличие данного алгоритма от *B*-алгоритма заключается в том, что здесь на каждой итерации текущие радиусы участков r_l^b, r_l^p уменьшают по правилу

$$r_l^b = \rho\, r_l^b,\ r_l^p = \rho\, r_l^p,\ l \in [1:|X|],$$

где $\rho \in (0;1]$ – свободный адаптивный *параметр редукции*. Величина параметра ρ увеличивается (поиск диверсифицируется), если на данной итерации улучшить решение не удалось, и уменьшается в противном случае.

Рассмотренная схема *B*-алгоритма учитывает то обстоятельство, что решается задача условной оптимизации только при размещении пчел-разведчиков. Если эти пчелы оказываются вблизи границы множества *П*, то некоторые фрагменты соответствующих элитных либо перспективных участков могут выходить за его границы. Можно предложить несколько вариантов решения данной проблемы, из которых чаще всего используют два следующих: применение в качестве фитнес-функции аддитивной свертки целевой функции и функции штрафа; использование «невидимых стен», когда участки могут беспрепят-

ственно выходить за границы области Π, но значения фитнес-функции пчел, находящихся вне этой области, игнорируются.

В терминах генетического алгоритма можно сказать, что, как и алгоритм муравьиной колонии, *B*-алгоритм и его модификации используют вещественное кодирование «бессмертных» особей. Механизм изменения координат пчел в процессе итераций можно интерпретировать как своеобразную мутацию их координат.

Пример 3.5. Рассмотрим задачу минимизации *B*-алгоритмом двумерных функций Розенброка, Химмельблау и Растригина (Приложение Б) в следующих параллелепипедах допустимых значений Π соответственно:

$$\Pi=\left\{X \middle| x_1, x_2 \in [-5,0;5,0]\right\},$$
$$\Pi=\left\{X \middle| x_1, x_2 \in [-10,0;10,0]\right\},$$
$$\Pi=\left\{X \middle| x_1, x_2 \in [-2,0;2,0]\right\}.$$

Полагаем, что соответствующие фитнес-функции обратны указанным функциям.

Если не оговорено противное, имеем в виду следующие значения свободных параметров алгоритма: число агентов-разведчиков $\left|S^o\right| = 16$; радиусы элитных и перспективных участков постоянны и равны $r_i^b = r_i^p = r = 0,2$; $i \in [1:|X|]$; число элитных участков $\left|A^b\right| = 2$; число перспективных участков $\left|A^p\right| = 3$; число агентов, отправляемых на каждый элитный участок, $n_b = 7$; число агентов, отправляемых на каждый перспективный участок, $n_p = 4$; максимальное число итераций $\hat{t} = 500$; условие останова – стагнация вычислительного процесса в течение $\delta_t = 20$ итераций. Таким образом, общее число рабочих пчел равно

$$\left|S^w\right| = n_b \left|A^b\right| + n_p \left|A^p\right| = 7 \cdot 2 + 4 \cdot 3 = 26.$$

Величины, характеризующие эффективность *B*-алгоритма, как и всякого стохастического алгоритма поисковой оптимизации, являются случайными. Поэтому используем мультистарт с числом стартовых точек $\hat{n} = 30$. Эффективность алгоритма оцениваем следующими величинами: среднее по числу стартов число итераций $\overline{t}_{end}$; аналогичная средняя абсолютная погрешность решения $\overline{\varepsilon}$; оценка вероятности локализации глобального максимума фитнес-функции $\tilde{\xi}$.

Рассмотрим зависимость указанных критериев эффективности алгоритма от значений следующих свободных параметров: число пчел-разведчиков $\left|S^o\right|$ – рис. 3.32; радиусы элитных и перспективных участков r – рис. 3.33; величина δ_t, определяющая начало стагнации вычислительного процесса, – рис. 3.34.

Из рис. 3.32 становится ясно, что с увеличением числа пчел-разведчиков $\left|S^o\right|$ для одноэкстремальной функции Розенброка имеет место уменьшение среднего числа итераций $\overline{t}_{end}$, а для многоэкстремальных функций Химмельблау и Растригина – практическое его постоянство. Показательным является быстрое (на один-два порядка) уменьшение средней погрешности решения $\overline{\varepsilon}$ для всех рассматриваемых функций с ростом величины $\left|S^o\right|$. Начиная с числа пчел-разведчиков, равного $\left|S^o\right| \approx 20$, наблюдается стабилизация величин $\overline{t}_{end}$, $\overline{\varepsilon}$.

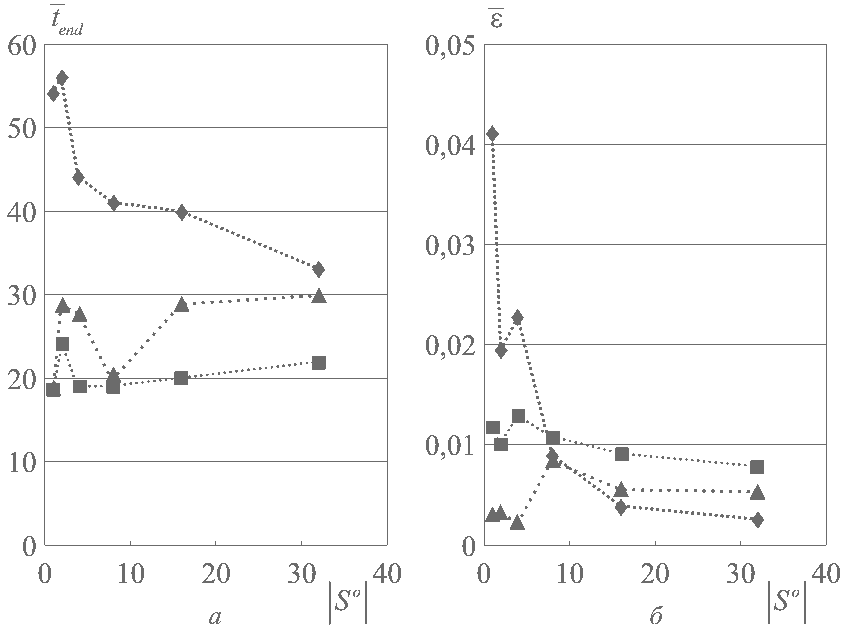


Рис. 3.32. Число итераций $\overline{t}_{end}$ (*а*) и погрешность решения $\overline{\varepsilon}$ (*б*) в функции числа пчел-разведчиков $|S^o|$: ♦– функция Розенброка; ▲ –функция Химмельблау; ■ – функция Растригина

Данные, приведенные на рис. 3.33 позволяют сделать следующие выводы. Для всех рассматриваемых целевых функций с ростом радиусов элитных и перспективных участков r число итераций $\overline{t}_{end}$ уменьшается, а погрешность решения $\overline{\varepsilon}$ возрастает.

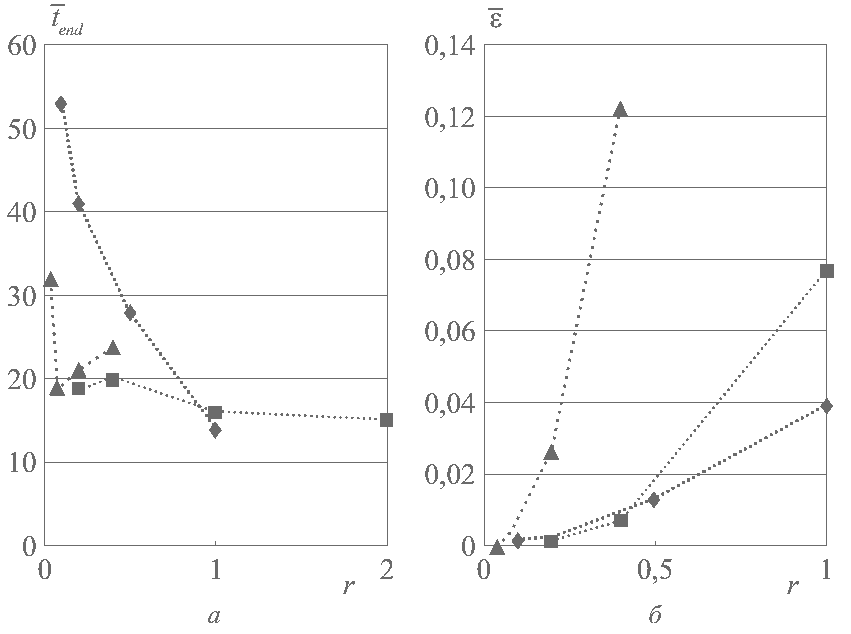


Рис. 3.33. Число итераций $\overline{t}_{end}$ (*а*) и погрешность решения $\overline{\varepsilon}$ (*б*) в функции радиусов элитных и перспективных участков r: ♦– функция Розенброка; ▲ –функция Химмельблау; ■ – функция Растригина

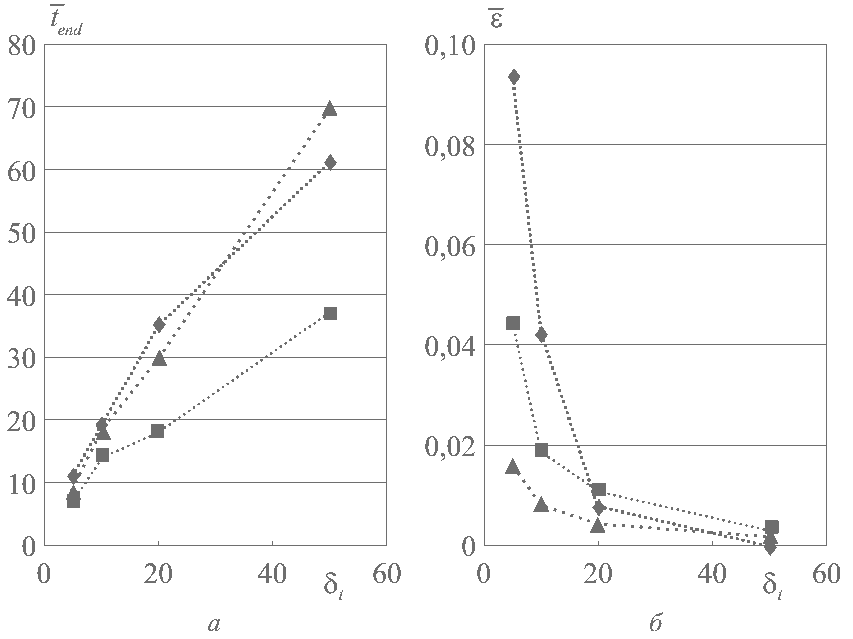


Рис. 3.34. Число итераций $\overline{t}_{end}$ (*а*) и погрешность решения $\overline{\varepsilon}$ (*б*) в функции величины δ_t: ♦– функция Розенброка; ▲–функция Химмельблау; ■– функция Растригина

Скорость этих изменений пропорциональна сложности топологии целевой функции. Эффект обусловлен диверсификацией поиска с ростом величины *r*.

Из данных, представленных на рис. 3.34, следует, что с ростом величины δ_t число итераций $\overline{t}_{end}$ возрастает для всех трех функций с примерно одинаковой скоростью. В то же время погрешности $\overline{\varepsilon}$ быстро убывают и при $\delta_t \approx 40$ становятся практически неразличимыми.

Рассмотренный пример показывает также, что *B*-алгоритм обеспечивает оценку вероятности локализации глобального минимума функций Розенброка и Химмельблау, равную $\tilde{\xi} = 1{,}0$. Для функции Растригина, которая является многоэкстремальной и имеет сложную топологию, та же оценка зависит от значений свободных параметров алгоритма и изменяется от 0,20 до 0,97●

Известны результаты сравнения эффективности *B*-алгоритма с эффективностью канонического генетического алгоритма, а также с одной из модификаций алгоритма колонии муравьев. Приведем некоторые из этих результатов.

При максимизации двумерной функции Де Янга (Приложение Б) в области $D = \{X \mid x_1, x_2 \in [-2{,}048; 2{,}048]\}$ *B*-алгоритм нашел решение в 120 раз быстрее муравьиного и в 207 раз быстрее генетического алгоритма. При этом в 100 запусках *B*-алгоритм 100 раз нашел оптимальное решение, то есть обеспечил оценку вероятности локализации глобального экстремума $\overline{\xi} = 1$.

В случае двумерной задачи Голдштейна и Прайса (Приложение Б) *B*-алгоритм оказался почти в пять раз быстрее алгоритма муравьев и генетического алгоритма (опять же со 100% успехом).

При минимизации двумерной функции Розенброка в области $D=\{X \mid x_1, x_2 \in [-1,2;1,2]\}$ (см. Приложение Б) *B*-алгоритм также достиг 100 % успеха и потребовал по крайней мере в два раза меньшего числа итераций по сравнению с указанными алгоритмами. В случае четырехмерной функции Розенброка в той же области поиска *B*-алгоритму потребовалось большее число испытаний для достижения минимума со 100 % успехом.

3.3.3 Алгоритм колонии искусственных пчел

Алгоритм колонии искусственных пчел (*Artificial Bee Colony, ABC*) предложил Карабога (*D. Karaboga*) в 2005 г. для задачи непрерывной безусловной оптимизации. Впоследствии алгоритм модифицирован в работах того же автора и Бастурка (*B. Basturk*) и распространен на задачу условной оптимизации. *ABC*-алгоритм хорошо зарекомендовал себя при решении как унимодальных, так и многоэкстремальных задач оптимизации. Суть алгоритма состоит в комбинировании алгоритмов локального и глобального поиска с целью достижения оптимального баланса между ними.

Рассмотрим задачу глобальной условной минимизации в гиперкубе

$$D=\left\{X \mid x^- \le x_i \le x^+, i \in [1:|X|]\right\} \subset R^{|X|}.$$

Напомним, что произвольный гиперпараллелепипед *П* с помощью линейных преобразований компонент вектора *X* легко трансформировать в указанный гиперкуб.

В *ABC*-алгоритме популяция содержит три группы пчел: рабочие пчелы, пчелы-исследователи и пчелы-разведчики. На разных этапах алгоритма рабочие пчелы превращаются в пчел-разведчиков и наоборот, так что в каждый данный момент времени популяция состоит из множеств рабочих пчел и пчел-исследователей, имеющих равные мощности. Каждому разрабатываемому участку (источнику нектара) ставится в соответствие одна рабочая пчела. Если некоторый участок перестает представлять интерес для популяции, то этот участок объявляется *заброшенным* и соответствующая рабочая пчела становится пчелой-разведчиком.

Введем следующие обозначения: $s_i^w, i \in [1:|S^w|]$ – рабочие пчелы; $s_j^o, j \in [1:|S^o|]$ – пчелы-исследователи; $|S|=|S^w|+|S^o|$ – общее число пчел в рое. Текущие положения рабочих пчел и пчел-исследователей определяют векторы их координат X_i^w, X_j^o соответственно, которым придаем смысл положений источников нектара. Количество нектара в его источнике полагаем равным значению фитнес-функции в соответствующей точке. Заброшенным объявляем участок, который не изменяется в течение заданного числа итераций.

Схема *ABC*-алгоритма включает в себя этап инициализации популяции и повторяющиеся итерации поискового процесса, каждая из которых состоит из следующих стадий: разработка источников рабочими пчелами; разработка источников пчелами-исследователями; поиск новых источников пчелами-разведчиками; завершение итераций. Рассмотрим указанные шаги алгоритма.

Инициализация популяции заключается в генерации в гиперкубе D начальных решений (координат участков) по формуле

$$X_i^w(0) = U_{|X|}(x^-; x^+),\ i \in [1:|S^w|] \tag{3.24}$$

и вычислении количеств нектара в них

$$\varphi(X_i^w(0)) = \varphi_i^w(0) = \begin{cases} \dfrac{1}{1+f(X_i^w(0))}, & f(X_i^w(0)) \geq 0, \\ 1+abs\left(f(X_i^w(0))\right), & f(X_i^w(0)) < 0. \end{cases} \tag{3.25}$$

Заметим, что формула (3.25) сводит задачу минимизации целевой функции $f(X)$ к задаче максимизации фитнес-функции.

Решение $X_i^w(0)$ интерпретируем как источник, подлежащий разработке рабочей пчелой s_i^w.

Стадия разработки участков. На этой стадии каждая рабочая пчела $s_i^w, i \in [1:|S^w|]$ пытается отыскать в окрестности своего текущего участка X_i^w новый участок X_i' с большим количеством нектара по следующей схеме:

1) генерируем точку X_i',

2) вычисляем количество нектара в этой точке и сравниваем его с количеством нектара в источнике X_i^w,

3) если количество нектара в новой точке больше, то перемещаем рабочую пчелу в эту точку.

Генерацию точки X_i' выполняем по формуле

$$x_{i,j}' = \begin{cases} x_{i,j}^w + U_1(-1;1)\left(x_{i,j}^w - x_{k,j}^w\right), & j = l, \\ x_{i,j}^w, & j \neq l, \end{cases} \tag{3.26}$$

где $l = U_1[1:|X|]$, $k = U_1[1:|S^w|]$, $k \neq i$. Другими словами, новый источник находим на прямой, параллельной случайно выбранной координатной оси $0x_l$. Источник на этой прямой располагаем на случайном расстоянии от источника X_i^w, пропорциональном расстоянию по оси $0x_l$ до случайно выбранного другого источника X_k^w (рис. 3.35).

Отметим следующее обстоятельство. Из формулы (3.26) следует, что чем меньше расстояние между источниками X_i^w, X_k^w, тем меньше расстояние между X_i^w и новой точкой X_i'. Поскольку на завершающих итерациях алгоритма пчелы s_i^w, $i \in [1:|S^w|]$ концентрируются вблизи глобального максимума фитнес-функции, указанное расстояние, а тем самым и шаг поиска с ростом числа итераций уменьшаются. В результате уменьшается широта обзора и повышается точность локализации искомого максимума.

Рассмотренная схема генерации точки X_i' может привести к ситуации, когда эта точка оказывается за пределами области поиска D. В соответствии с алгоритмом *ABC* в этом случае в качестве результирующей точки используем проекцию точки X_i' на ближайшую границу этой области.

Количество нектара φ_i' в найденных точках X_i', $i \in [1:|S^w|]$ вычисляем по формуле, аналогичной формуле (3.25). Если имеет место ситуация

$\varphi_i' > \varphi_i^w$, то полагаем $X_i^w = X_i'$, иначе оставляем вектор X_i^w без изменений.

Стадия разработки источников пчелами-исследователями. На этой стадии рабочие пчелы делятся информацией о полезности своих источников с пчелами-исследователями, которые на основе полученной информации принимают решение о том, к какому источнику им целесообразно отправиться для его дальнейшей разработки. Чем выше качество источника, тем больше пчел-исследователей к нему будет направлено. Пчелы-исследователи разрабатывают окрестности выбранного источника аналогично рабочим пчелам. Полученную информацию о качестве источника пчелы-исследователи передают рабочим пчелам, разрабатывающим данный источник. На основе этой информации те принимают решение, следует ли им переключаться на разработку нового источника или нет. После этого пчелы-исследователи возвращаются обратно в улей.

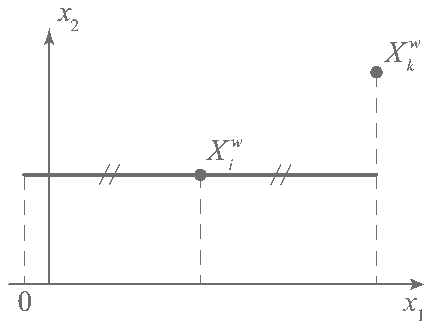


Рис. 3.35. К схеме локального поиска: участок X_i' с равной вероятностью находится в любой из точек выделенного отрезка

Выбор источника, к которому должна направиться пчела-исследователь, осуществляем алгоритмом рулетки. Так что вероятность ξ_i^o того, что пчела-исследователь s_i^o выберет источник X_i^w, определяет формула

$$\xi_i^o = \frac{\varphi_i^w}{\sum\limits_{j \in [1:|S^w|]} \varphi_j^w}, \; i \in [1:|S^o|]. \tag{3.27}$$

Пчела-исследователь осуществляет локальный поиск в окрестности выбранного источника $X_i^w, i \in [1:|S^w|]$ по рассмотренной схеме стадии разработки источников. Каждая из пчел-исследователей, разрабатывающая источник X_i^w, передает рабочей пчеле s_i^w, «ответственной» за этот источник, соответствующую полезность, вычисленную по формуле (3.25). На основании информации от всех пчел-исследователей, разрабатывающих источник X_i^w, рабочая пчела s_i^w принимает решение о переносе источника в новую точку.

Стадия поиска новых источников пчелами-разведчиками. Если в течение b_t итераций полезность источника $X_i^w, i \in [1:|S^w|]$ не удается улучшить, то источник переводим в разряд заброшенных. Соответствующая этому источнику рабочая пчела становится пчелой-разведчиком. Автор алгоритма *ABC* предлагает использовать значение параметра b_t, равное $|X| \cdot |S^w|$.

По схеме инициализации популяции для всех пчел-разведчиков генерируем новые источники и вновь переводим всех этих пчел в ранг рабочих пчел.

Стадия завершения итераций. На данной стадии проверяем выполнение условия завершения итераций, и если оно выполнено, заканчиваем вычисления. В противном случае возвращаемся к стадии разработки источников рабочими пчелами. Как всегда, в качестве условия окончания итераций может быть использовано, например, условие достижения максимального числа итераций или условие наступления стагнации.

Общая схема алгоритма *ABC* имеет следующий вид.

1) По формулам (3.24), (3.25) инициализируем популяцию s_i^w рабочих пчёл в источниках $X_i^w, i \in [1:|S^w|]$.

2) Вычисляем исходное приближение к решению

$$\varphi(\tilde{X}^*) = \max_{i \in [1:|S^w|]} \varphi(X_i) = \tilde{\varphi}^*.$$

3) Для каждой рабочей пчелы s_i^w по формуле (3.26) генерируем точку X_i' и по формуле, аналогичной формуле (3.25), вычисляем ее полезность φ_i'. Если $\varphi_i' > \varphi_i^w$, то источник нектара из точки X_i^w переносим в точку X_i'.

4) По формуле (3.27) для каждого из источников нектара X_i^w вычисляем вероятность ξ_i^o того, что пчела-исследователь выберет его для разработки. С помощью алгоритма рулетки назначаем каждой пчеле-исследователю источник, который ей следует разрабатывать, и отправляем ее к этому источнику.

5) Для каждой пчелы-исследователя по формуле (3.26) генерируем точку X_i' и по формуле, аналогичной формуле (3.25), вычисляем ее полезность φ_i'.

6) На основании информации о результатах локального поиска, выполненного пчелами-исследователями, разрабатывающими каждый данный источник нектара, принимаем решение о перемещении или не перемещении соответствующей рабочей пчелы в новый источник.

7) Если решение в источнике X_i^w не улучшается в течение b_t итераций, то данный источник исключаем из процесса разработки и соответствующую ему рабочую пчелу объявляем пчелой-разведчиком.

8) Для каждой пчелы-разведчика по формуле (3.24) генерируем новый источник нектара и объявляем ее рабочей пчелой, разрабатывающей этот источник.

9) В каждом текущем источнике нектара $X_i^w, i \in [1:|S^w|]$ вычисляем значение фитнес-функции $\varphi(X_i)$ и определяем максимальное из этих значений $\varphi(X_{i_j}^w) = \max_{i \in [1:|S^w|]} \varphi(X_i^w)$. Если $\tilde{\varphi}^* < \varphi(X_{i_j}^w)$, то полагаем $\tilde{\varphi}^* = \varphi(X_{i_j}^w)$.

10) Проверяем условие окончания итераций. Если это условие выполнено, то принимаем $X^* = \tilde{X}^*$ и завершаем вычисления. Иначе переходим к шагу 3.

Кроме общих для всех популяционных алгоритмов, свободным параметром алгоритма является параметр b_t – максимальное число неудачных итераций.

Пример 3.6. Рассмотрим задачи минимизации сферической функции, а также функций Розенброка, Растригина и Экли (Приложение Б). Множеством допустимых значений вектора варьируемых параметров во всех случаях является гиперкуб

$$D = \{X \mid -20 \le x_i \le 20, i \in [1:|X|]\}.$$

Полагаем, что фитнес-функция обратна целевой функции.

Используем число рабочих пчел (а значит, и источников нектара), равное $|S^w| = 20$, так что общее число пчел в рое $|S| = 40$. Максимальное число неудачных итераций в источнике принимаем равным $b_t = |S^w| \cdot |X|$. В качестве критерия окончания итераций используем стагнацию вычислительного процесса в течение $\delta_t = 20$ итераций. Размерность вектора X варьируем в пределах от двух до 16.

Скорость сходимости алгоритма *ABC* при решении указанных задач иллюстрируют рис. 3.36, 3.37. Рисунки показывают высокую скорость сходимости алгоритма даже в случае высокой размерности пространства поиска и даже для таких сложных многоэкстремальных функций, как функции Растригина и Экли.

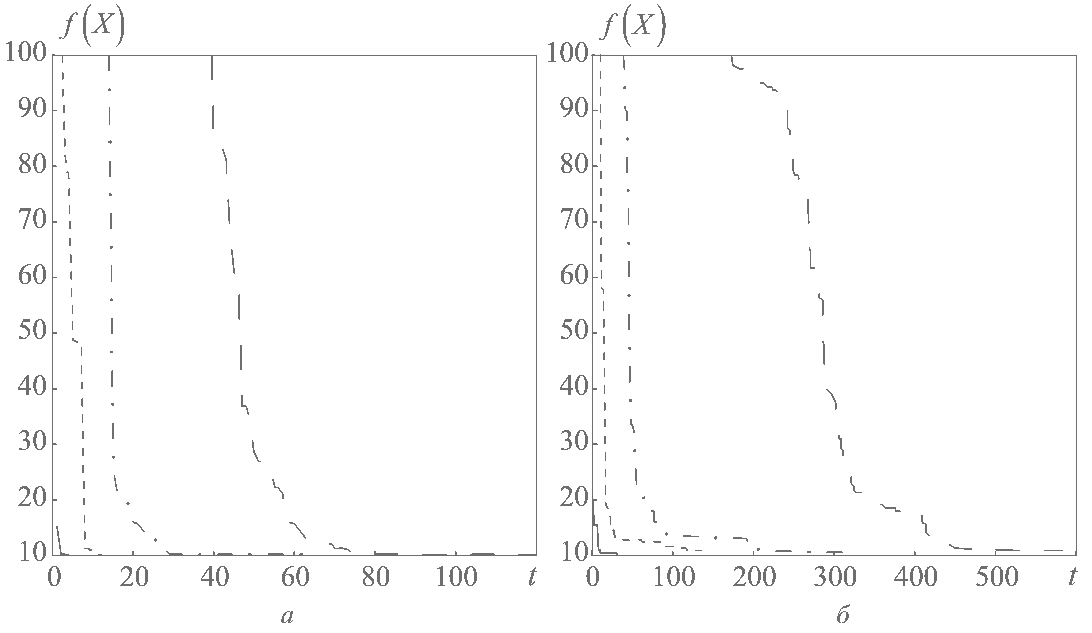


Рис. 3.36. Сходимость *ABC*-алгоритма: *а* – сферическая функция: — $|X| = 2$, ····· $X = 4$, -·- $|X| = 8$, – – $|X| = 16$; *б* – функция Розенброка: — $|X = 2$, ···· $X| = 4$, -·- $|X| = 8$, – – $|X| = 16$

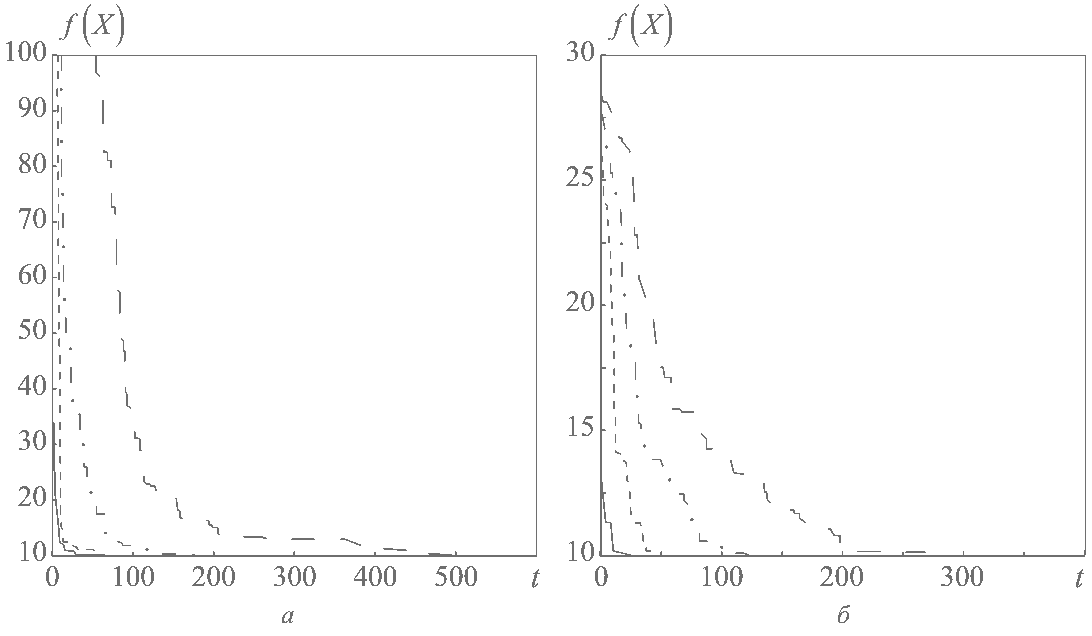


Рис. 3.37. Сходимость *ABC*-алгоритма: *а* – функция Растригина: — $|X = 2$, ···· $|X = 4$, -·- $X| = 8$, – – $|X| = 16$; *б* – функция Экли: — $X| = 2$, ···· $X| = 4$, -·- $|X| = 8$, – – $|X| = 16$

Для исследования зависимости скорости сходимости алгоритма от размерности задачи воспользуемся мультистартом с числом запусков алгоритма $\hat{n} = 30$. В качестве меры скорости сходимости используем усредненную величину t_{end}, то есть усредненное число итераций до начала стагнации вычислений. Результаты исследования иллюстрируют табл. 3.3 и рис. 3.38.

Таблица 3.3 **Среднее число итераций $\overline{t}_{end}$**

X	Среднее число итераций $\overline{t}_{end}$			
	Сферическая функция	Функция Розенброка	Функция Растригина	Функция Экли
2	82	33	116	154
4	155	92	239	287
8	312	254	463	574
16	626	589	942	1137

Табл. 3.3 и рис. 3.38 показывают, что в широком диапазоне варьирования размерности задачи имеет место практически линейная зависимость величины $\overline{t}_{end}$ от этой размерности.

Результаты исследования показывают, что *ABC*-алгоритм обеспечивает 100% локализацию глобального экстремума таких топологически сложных функций, как функции Растригина и Экли, даже при их высокой размерности, равной 16. В то же время алгоритм показывает гораздо худшие результаты при оптимизации овражной функции Розенброка, особенно при ее высокой размерности. Это обстоятельство связано с тем, что дно оврага функции Розенброка не ориентировано вдоль какой-либо оси системы координат $0x_1x_2...x_{|X|}$, а локальный поиск в алгоритме *ABC* осуществляется вдоль этих координатных осей. В результате пчела, расположенная вблизи дна оврага, постоянно «наталкивается» на крутые склоны оврага и не может улучшить свою позицию. В такой ситуации найти глобальный экстремум могут только пчелы-разведчики, осуществляющие глобальный поиск. Однако точность их способа поиска намного ниже точности локального поиска рабочих пчел. Исправить ситуацию мог бы другой алгоритм локального поиска, например, известный алгоритм Хука – Дживса или алгоритм Розенброка.

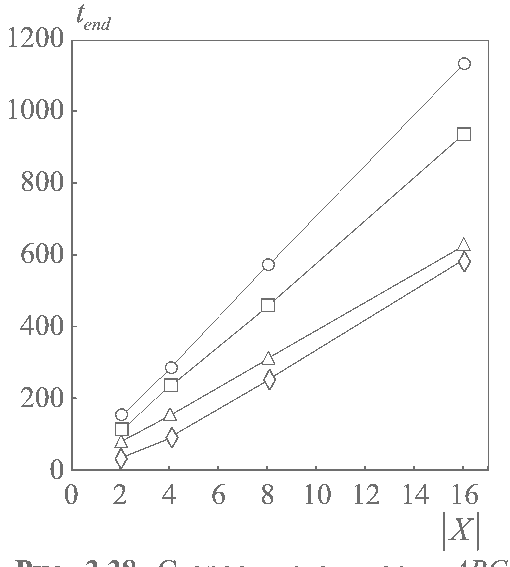


Рис. 3.38. Скорость сходимости *ABC*-алгоритма $\overline{t}_{end}$ в функции размерности задачи $|X|$: △ – сферическая функция; ◊ – функция Розенброка; □ – функция Растригина; ○ – функция Экли

Значительный интерес представляет зависимость вероятности локализации глобального экстремума ξ в функции размера популяции $|S|$. В качестве оценки $\tilde{\xi}$ этой вероятности используем отношение числа успешных запусков алгоритма (когда глобальный максимум был локализован) к общему числу запусков $\hat{n} = 30$.

Результаты исследования для функции Растригина размерности $|X| = 8$ представлены в табл. 3.4. Исследование выполнено при следующих значениях свободных параметров алгоритма: $b_t = |S^w| \cdot |X|$; $\delta_t = 10$.

Таблица 3.4 **Оценка вероятности локализации глобального экстремума $\tilde{\xi}$ функции Растригина в функции размера популяции $|S|$**

$\|S$	6	10	14	20	30	40
$\tilde{\xi}$	0,07	0,60	0,83	0,90	0,97	1,00

Табл. 3.4 показывает, что вероятность локализации глобального экстремума сложной многоэкстремальной функции Растригина достаточно высокой размерности ($|X| = 8$) быстро растет с ростом размера популяции и уже при общем числе пчел в популяции, равном 30, достигает величины 0,97●

Известно значительное число модификаций *ABC*-алгоритма, имеющих целью повышение его эффективности. Например, в работе Ал. А. Олейника, С. А. Субботина (2008 г.) предложена модификация, основные положения которой заключаются в следующем.

Если общее число пчел в данном источнике X_i достигло своего порогового значения $n_{\max}$, то одна из них (пусть это будет рабочая пчела s_i^w) изменяет свои координаты по правилу, близкому к правилу (3.26),

$$x'_{i,j} = \begin{cases} x_{i,j}^w + u_{sign}^{\pm 1} b_x, & j = l, \\ x_{i,j}^w, & j \neq l, \end{cases} \quad i \in [1:|S^w|], \; j \in [1:|X|],$$

где $l = U_1(1:|X|)$ – случайная натуральная величина, определяющая координатную ось, вдоль которой перемещается пчела s_j^w; $u_{sign}^{\pm 1}$ – случайная дискретная величина, которая с равной вероятностью принимает значения ± 1 и определяет направление смещения пчелы s_i^w; b_x – свободный параметр, шаг смещения.

Полезность φ_i^w, $i \in [1:|S^w|]$ источника нектара X_i^w определяет формула

$$\varphi_i^w = \begin{cases} 1, & \tilde{\varphi}_i^w > 1, \\ \tilde{\varphi}_i^w, & \tilde{\varphi}_i^w \in [\delta_\varphi; 1], \\ 0, & \tilde{\varphi}_i^w < \delta_\varphi, \end{cases} \tag{3.28}$$

где

$$\tilde{\varphi}_i^w = 1 - \overline{\varphi}_i^w + U_1(-b_n; b_n). \tag{3.29}$$

Здесь $\delta_\varphi > 0$ – порог полезности источника (рекомендованное значение равно 0,1); $b_n > 0$ – уровень случайных возмущений (рекомендованное значение равно 0,1); $\overline{\varphi}_i^w$ – нормированная полезность источника нектара X_i^w, рассчитанная на основе его минимальной и максимальной полезностей на протяжении всех итераций $[1: t]$.

На текущей итерации вероятность ξ_i^w выполнения пчелой s_i^w процедуры вербовки других пчел оцениваем в соответствии с выражением

$$\xi_i^w = \frac{a_i}{b_\xi},\ i \in [1:|S^w|],$$

где $a_i = \max\left(\varphi_i^w - b_a\overline{\varphi}^w,\ 0\right)$ – привлекательность источника X_i^w, $b_\xi > 0$ – параметр, управляющий вероятностью ξ_i^w; $\overline{\varphi}^w$ – средняя текущая полезность всех $|S^w|$ участков; $b_\alpha > 0$ свободный коэффициент, регулирующий влияние величины $\overline{\varphi}^w$.

Вероятность того, что незанятый фуражир станет разведчиком, определяет формула

$$\xi^o = \exp\left(-\frac{a^2}{2b_o}\right),$$

где $a = \sum\limits_{i\in[1:|S^w|]} a_i$ – суммарная текущая привлекательность всех источников, $b_o > 0$ – свободный параметр.

Авторами рассматриваемой модификации выполнено сравнение ее эффективности с каноническим алгоритмом роя частиц. Исследование показало, что при оптимизации сложных многомерных многоэкстремальных функций данная модификация обеспечивает более высокую точность локализации глобального экстремума.

3.3.4. Гибридный алгоритм

Алгоритм предложен в работах С. А. Субботина, Ан. А. Олейника и Ал. А. Олейника (2008 г.) и существенно использует алгоритм оптимизации путем имитации отжига.

Рассмотрим задачу глобальной условной максимизации в гиперпараллелепипеде *П*. Схема алгоритма включает в себя следующие основные шаги.

1) Инициализация алгоритма.

2) Выбор пчел, на основе которых с помощью процедуры скрещивания будут получены новые пчелы.

3) Скрещивание отобранных агентов.

4) Моделирование процесса вербовки пчел.

5) Локальная оптимизация найденных лучших решений с помощью алгоритма имитации отжигом.

6) Выбор лучшей пчелы.

7) Выбор пчел-разведчиков.

8) Обновление динамических параметров алгоритма.

9) Завершение итераций.

Последовательно рассмотрим перечисленные шаги алгоритма.

Инициализация алгоритма. Этап инициализации включает в себя следующие действия. Задаем начальные координаты $X_i(0)$ пчел-разведчиков (источников нектара)

$$x_{i,j}(0) = x_j^- + U_1(0;1)\ \left(x_j^+ - x_j^-\right),\ i \in [1:|S^o|],\ j \in [1:|X|]. \qquad (3.30)$$

Вычисляем полезность $\varphi_i(0)$ каждого из этих источников $X_i(0)$. Аналогично вводим решения $X_j^w(t)$, соответствующие рабочим пчелам $s_j^w(t)$, $j \in [1:|S^w|]$. Устанавливаем начальное число итераций $t=1$ и начальную «температуру», равную $\overline{\varepsilon}_0$.

Выбор агентов для скрещивания. Выбираем лучшую пчелу $s_*(t)$ за все предыдущие итерации $[1:t]$ такую, что

$$\max_{\substack{i\in[1:|S|]\\ \tau\in[1:t]}} \varphi_i(\tau) = \tilde{\varphi}^* = \varphi(s_*(t)), \tag{3.31}$$

а также $|S^w|$ рабочих пчел, удовлетворяющих условию

$$\exp\left(-\frac{\tilde{\varphi}^* - \varphi_i}{\overline{\varepsilon}}\right) > U_1(0;1),\ i \in [1:|S|], \tag{3.32}$$

где $\overline{\varepsilon} = \overline{\varepsilon}(t)$ – текущая «температура».

Решение, соответствующее лучшей пчеле $s_*(t)$, обозначаем $\tilde{X}^*(t)$. Отметим, что формула (3.32) реализует отбор пчел для скрещивания по схеме алгоритма имитации отжига.

Скрещивание рабочих пчел требует выполнения следующих шагов.

1) Создаем новых агентов на основе рабочих пчел по схеме

$$x_{j,k}^w(t+1) = x_{j,k}^w(t) + u_{sign}^{\pm 1}\left(x_{j,k}^w(t) - \tilde{x}_k^*(t)\right),\ j \in [1:|S^w|],\ k \in [1:|X|].$$

2) По аналогичной схеме создаем такое же число новых пчел $|S^w|$ на основе лучшей пчелы:

$$x_{|S^w|+l,\,k}^w(t+1) = \tilde{x}_k^*(t) \pm U_1(0;1)\left(x_{l,k}^w(t) - \tilde{x}_k^*(t)\right),\ l \in [1:|S^w|],\ k \in [1:|X|].$$

3) Производим коррекцию полученных $2|S^w|$ решений путем проецирования на границы области D тех из них, которые выходят за границы этой области. За скорректированными решениями сохраняем прежние обозначения.

4) Вычисляем полезность полученных решений $\varphi(X_l^w(t+1))$, $l \in [1:2\cdot|S^w|]$ и выбираем новую лучшую особь $s_*(t+1)$, а также соответствующее решение $\tilde{X}^*(t+1)$ с полезностью $\tilde{\varphi}^*(t+1)$.

Моделирование вербовки. К выполнению процедуры вербовки допускается множество пчел $S^r = \{s_i^r(t),\ i \in [1:|S^r|]\}$, включающее в себя лучшую пчелу $s_*(t+1)$, всех рабочих пчел $s_i^w(t)$, $i \in [1:|S^w|]$, а также всех новых рабочих пчел $s_j^w(t+1)$, $j \in [1:2\cdot|S^w|]$. Таким образом, мощность множества S^r равна $|S^r| = 3\cdot|S^w| + 1$. Вербовка включает в себя следующие шаги.

1) Выполняем нормирование полезностей пчел, допущенных к вербовке, то есть полагаем

$$\varphi_i^r = \frac{\varphi_i^r}{\tilde{\varphi}^*(t+1)},\ i \in [1:|S^r|].$$

2) Добавляем к нормированным полезностям случайные возмущения (шумы) и корректируем полученные полезности по формуле

$$\varphi_i^r = \begin{cases} 1, & \tilde{\varphi}_i^r > 1, \\ \tilde{\varphi}_i^r, & \tilde{\varphi}_i^r \in [\delta_\varphi; 1], \quad i \in [1:|S^r|], \\ 0, & \tilde{\varphi}_i^r < \delta_\varphi, \end{cases}$$

близкой к формуле (3.28). Здесь

$$\tilde{\varphi}_i^r = \varphi_i^r + U_1(-b_n; b_n).$$

3) Вычисляем привлекательности пчел множества S^r

$$\phi_i^r = \max_{i \in [1:|S^r|]} \left(\varphi_i^r - b_\phi \overline{\varphi}^r, 0\right),$$

где $\overline{\varphi}^r = \dfrac{1}{|S^r|} \sum_{i=1}^{|S^r|} \varphi_i^r$ – средняя нормированная полезность пчел множества S^r; b_ϕ – положительный параметр, определяющий вес компоненты $\overline{\varphi}^r$ в привлекательности пчел.

4) Полагаем, что пчела s_i выполнила вербовку агента, если для этого агента справедливо неравенство

$$\phi_i^r > b_r \overline{\varphi}^r, \; i \in [1:|S^r|], \tag{3.33}$$

где $b_r \in (0;1]$ – положительный параметр, определяющий насколько выше средней полезности должна быть привлекательность данной пчелы, чтобы она смогла выполнить вербовку агента.

Обозначаем $\tilde{S}^r = \{\tilde{s}_i^r, i \in [1:|\tilde{S}^r|]\}$ – множество пчел, выполнивших вербовку, то есть удовлетворяющих неравенству (3.33). Соответствующее множество завербованных агентов (ставших занятыми фуражирами) обозначаем $S^f = \{s_i^f, i \in [1:|S^f|]\}$, где полагается, что агента s_i^f завербовала пчела $\tilde{s}_i^r$; $|S^f| = |\tilde{S}^r|$.

5) Определяем координаты завербованных агентов по формуле

$$x_{i,j}^f = \tilde{x}_{i,j}^r(t) + U_1(-0{,}5; 0{,}5)\, b_\delta, \; i \in [1:|S^f|], \; j \in [1:|X|],$$

где b_δ – положительная величина, определяющая максимально допустимое отклонение координат завербовавших и завербованных пчел.

Локальная оптимизация. В различных вариантах рассматриваемого гибридного алгоритма локальная оптимизация может выполняться, исходя только из лучшего решения $\tilde{X}^*(t+1)$, либо исходя из всех решений, отобранных на предыдущем шаге. В качестве алгоритма локальной оптимизации может быть использован любой из детерминированных алгоритмов, например алгоритм Нелдера – Мида, или один из стохастических алгоритмов, рассмотренных в 1.3.

Выбор лучшей пчелы осуществляем по формуле, аналогичной формуле (3.31).

Выбор пчел-разведчиков. К пчелам, которые на следующей итерации будут использованы в качестве разведчиков, относим следующих агентов: лучшая

пчела $s_*(t+1)$; пчелы множества $\tilde{S}^r = \{\tilde{s}_i^r, i \in [1:|\tilde{S}^r|]\}$, выполнившие вербовку; завербованные пчелы множества $S^f = \{s_i^f, i \in [1:|S^f|]\}$; пчелы числом $|S^o|$, случайные координаты которых получены по формуле вида (3.30). Вычисляем для всех указанных пчел полезность соответствующих решений.

Обновление динамических параметров алгоритма включает в себя следующие действия: увеличение значения счетчика числа итераций $t = t+1$; понижение текущей «температуры» $\overline{\varepsilon}(t) = \alpha\,\overline{\varepsilon}(t)$, где $\alpha \in (0;1]$ – коэффициент понижения «температуры»; уменьшение максимального отклонения

$$b_\delta = b_\delta \frac{\hat{t} - t}{\hat{t}}.$$

Завершение итераций предполагает проверку условий окончания итераций

$$t \geq \hat{t}\,,\ \overline{\varepsilon}(t) < \overline{\varepsilon}^{\min},$$

где $\overline{\varepsilon}^{\min}$ – минимально допустимая «температура». Если хотя бы одно из указанных условий выполнено, в качестве решения задачи принимаем решение $\tilde{X}^*(t)$ и завершаем вычисления, в противном случае переходим к шагу 2 рассмотренной схемы алгоритма.

Свободными являются следующие параметры алгоритма:

$\overline{\varepsilon}_0$ – начальная «температура»;

δ_φ – минимальный порог полезности пчелы (рекомендованное значение равно 0,1);

b_n – уровень возмущений (рекомендованное значение равно 0,1);

b_ϕ – вес компоненты $\overline{\varphi}^r$ в привлекательности пчел;

b_r – порог привлекательности пчелы;

b_δ – максимально допустимое отклонение координат вербовавших и завербованных пчел;

α – коэффициент понижения текущей «температуры»;

$\overline{\varepsilon}^{\min}$ – минимально допустимая «температура».

Вопросы для самопроверки

1. Представьте схему канонического алгоритма роя частиц. Определите понятия *собственный путь* частицы и ее *локальный путь*.

2. Сформулируйте основные отличительные признаки следующих модификаций алгоритма роя частиц: притягательно-отталкивающий алгоритм, алгоритм роя частиц с управлением скоростью частиц, алгоритм роя частиц с полной информацией (*FIPS*), алгоритм *FIPS* с весами, ранжированный алгоритм *FIPS*, *FDR* алгоритм роя частиц.

3. Дайте определение понятия *топология соседства частиц*. Проиллюстрируйте топологии соседства типа клика, кольцо, двумерный тор, кластерная топология.

4. Определите понятие *динамическая топология соседства частиц*. Назовите основные особенности следующих алгоритмов роя частиц, использующих динамические топологии соседства: клубный алгоритм роя частиц, алгоритм роя частиц со стереотипированием, алгоритм с дополнением графа соседства частиц.

5. Каковы бионические предпосылки муравьиных алгоритмов оптимизации?

6. Приведите схему алгоритма непрерывной оптимизации колонией муравьев *CACO*. Детализируйте содержание основных этапов этого алгоритма.

7. Назовите основные отличительные признаки алгоритма непрерывно взаимодействующей муравьиной колонии *CIAC*.

8. Изложите схему непрерывного ортогонального алгоритма муравьиной колонии *COAC*.

9. Представьте схему гибридного алгоритма непрерывно взаимодействующей муравьиной колонии *HCIAC*.

10. Каковы бионические предпосылки алгоритмов оптимизации пчелиным роем?

11. Приведите схему пчелиного алгоритма (*Bees algorithm*).

12. Представьте схему алгоритма колонии искусственных пчел *ABC*. Детализируйте основные операторы этого алгоритма.

13. Дайте схему гибридного пчелиного алгоритма, использующего локальную оптимизацию путем имитации отжига.

Глава 4

ДРУГИЕ ПОПУЛЯЦИОННЫЕ АЛГОРИТМЫ, ВДОХНОВЛЕННЫЕ ЖИВОЙ ПРИРОДОЙ

Представляем алгоритмы, вдохновленные живой природой и менее разработанные по сравнению с аналогичными алгоритмами, рассмотренными в предыдущей главе. Алгоритмы упорядочены по убыванию степени их разработанности и широте применения.

4.1. ИСКУССТВЕННЫЕ ИММУННЫЕ СИСТЕМЫ

Искусственные иммунные системы (*Artificial Immune Systems*, *AIS*) представляют собой относительно новую область исследований. Первые попытки разработки *AIS* относятся к 70-м гг. XX века. Однако представляющие практический интерес результаты были получены только в 1990-х гг. В настоящее время *AIS* интенсивно развиваются и широко применяются в вычислительной практике. С 2002 г. ежегодно проводится международная конференция *International Conference on Artificial Immune Systems*, полностью посвященная данной области исследований.

AIS можно определить как информационную технологию, использующую понятия, аппарат и некоторые результаты теоретической иммунологии для решения прикладных задач. Кроме решения задач оптимизации, модели *AIS* широко используют для построения многоагентных и самоорганизующихся систем, моделей коллективного интеллекта и автономных распределенных систем, моделей искусственной жизни и обучающихся систем и тому подобное.

4.1.1. Биологические основы

Биологическим прототипом *AIS* является иммунная система человека и обработка информации в ней молекулами белков (пептидов). Эта система представляет собой сложную адаптивную структуру, использующую комбинацию различных механизмов защиты от внешних патогенов – любых микроорганизмов (включая вирусы и бактерии), способных вызывать патологическое состояние (болезнь) человека.

Защитная система человека имеет иерархическую многоуровневую структуру, при этом каждый следующий уровень повышает специфичность защиты. Первый эшелон защиты от патогенов реализует кожный покров, который создает физический барьер на пути бактерий и вирусов. Второй уровень защиты – биохимический. На этом уровне защиту обеспечивают пот, слюна и слезы. Третий и четвертый эшелоны защиты представляют врожденная и приобретенная иммунные системы. Если патоген проникает через два предыдущих барьера, то *неспецифическую реакцию* (*non-specific*

defense reactions) на него осуществляет врожденная иммунная система. В случае, когда патоген преодолевает и эту систему, он попадает в сферу деятельности приобретенной иммунной системы, которая реализует *специфическую реакцию* (*specific defense reactions*) организма на вторжение. Важно, что ответ этой системы сохраняется после уничтожения возбудителя (в виде иммунологической памяти). В результате при следующем появлении данного патогена приобретенная иммунная система обеспечивает более быструю и сильную ответную реакцию по сравнению с первым появлением этого возбудителя.

Основная задача иммунной системы заключается в распознавании клеток, классификации их на своих и чужих, а также в уничтожении чужеродных клеток. С более общей точки зрения, задачей иммунной системы является уничтожение любого чужеродного агента, которым может быть болезнетворный микроорганизм, инородное тело, ядовитое вещество или переродившаяся клетка самого организма. Агенты, воспринимаемые иммунной системой как чужеродные, называют *антигенами* (*antigens*). Можно сказать, что антигены – это агенты, в ответ на появление которых в организме иммунная система образует специальные, реагирующие с ними *антитела* (*antibodies*).

При попадании антигенов в организм формируется *иммунный ответ организма*. Различают иммунный ответ на новые антигены – еще неизвестные иммунной системе (*первичный иммунный ответ*), и ответ на антигены, уже известные системе (*вторичный иммунный ответ*). Первичный иммунный ответ реализуется по следующей схеме: создание антител в ответ на вторжение в организм новых антигенов; адаптация (дозревание) антител; клонирование наилучших антител. Вторичный иммунный ответ использует иммунную память системы, то есть наилучшие антитела, выработанные ранее при первом контакте с данными патогенами.

В иммунном ответе организма участвуют два типа клеток – фагоциты и лимфоциты.

Фагоциты (*phagocytes*) – это клетки иммунной системы, основной функцией которых является поглощение антител (а также мертвых или погибающих клеток организма). С помощью фагоцитов иммунная система реализует функции врожденной иммунной системы.

Для использования в информационных технологиях наибольший интерес представляют функции приобретенной иммунной системы, которые реализуют, прежде всего, лимфоциты. *Лимфоциты* (*lymphocytes*) – главные клетки иммунной системы, вырабатывающие антитела в ответ на вторжение антигенов. По функциональным признакам различают несколько типов лимфоцитов, главными из которых являются *B*-лимфоциты, осуществляющие распознавание антигенов и выработку антител. Среди *B*-лимфоцитов выделяют клетки «памяти», живущие относительно долго и хранящие в себе информацию о встреченных ранее организмом чужеродных белках.

Распознавание антигенов *B*-клетками лейкоцитов (*иммунное распознавание*) основано на комплементарности между рецепторами этих клеток и активными участками антигена, образующими его область сцепления (*эпитопами*).

Комплементарностью называют пространственную взаимодополняемость молекул или их частей, приводящую к образованию связей между ними. Заметим, что именно комплементарность обеспечивает соединение двух полинуклеотидных цепей в двуспиральную молекулу ДНК. У антител может быть только один рецептор, в то время как антигены могут иметь большое число эпитопов. Поэтому один и тот же антиген может быть распознан как чужеродный агент многими антителами.

При распознавании *B*-лимфоцитами антигенов совпадение образа антигена в рецепторах и встреченного антигена может быть неполным. Мера близости антигена (точнее, его образа в рецепторах *B*-лимфоцита) и антитела носит наименование *аффинности* (*affinity*).

Известно несколько теорий, объясняющих механизм производства антител иммунной системой. Наиболее известной из этих теорий является клонально-селекционная теория Бернета (*F.M. Burnet*), в соответствии с которой при распознавании *B*-клетками антигена они стимулируются и начинают синтезировать антитела с той же специфичностью (то есть предназначенные для борьбы с данным антигеном) путем клонирования. При этом число клонов, произведенных *B*-лимфоцитом, пропорционально уровню его стимуляции. Процесс, который вызывает клонирование только тех *B*-клеток, которые синтезируют нужный тип антител, называют *клональным отбором* (*clonal selection*). Можно сказать, что клональный отбор создает подпопуляцию *B*-клеток, предназначенную для борьбы с соответствующим антигеном. После подавления проникновения в организм данного антигена большая часть клонированных *B*-клеток разрушается. Оставшиеся клетки реализуют функцию иммунной памяти, так что последующее воздействие похожего антигена приводит к более быстрой иммунной реакции.

При проектировании *AIS* широко используют также элементы теории *идиотипической иммунной сети*, созданной Ерне (*N.K. Jerne*) и развитой в работах Перельсона (*A. S. Perelson*). Согласно этой теории иммунный ответ основывается не только на вышерассмотренном взаимодействии *B*-клеток и антигенов, но также на взаимодействии *B*-клеток друг с другом. Этим взаимодействием в значительной степени объясняется эффект стимуляции *B*-клеток и память иммунной системы. С точки зрения информационных технологий, первичный иммунный ответ соответствует обучению иммунной системы распознаванию соответствующего антигена, а вторичный ответ – собственно распознаванию образов, то есть отнесению новых антигенов к одной из известных групп.

Оптимизационные методы, использующие идеи иммунной системы, рассматривают антиген как подлежащую решению задачу, а антитела – как векторы, соответствующие решениям этой задачи.

Известно значительное число математических моделей иммунной системы в целом и ее различных компонентов, построенных на основе систем детерминированных и стохастических дифференциальных уравнений, клеточных автоматов и других подходов. Для решения прикладных задач с помощью парадигмы *AIS* используют модели клонального отбора, иммунной сети, отрицательного отбора, библиотеки генов и другие модели. При решении задач

оптимизации чаще всего применяют модели клонального отбора и иммунной сети. Последняя модель включает в себя первую модель. Ограничимся поэтому рассмотрением только модели иммунной сети.

4.1.2. Оптимизация с помощью модели иммунной сети

Иммунная сеть представляет собой математическую структуру, имитирующую некоторые функции иммунной системы человека. Многие внешние черты сближают иммунную сеть с такой современной информационной технологией, как нейронные сети. Подобно нейронным сетям, иммунные сети обладают способностью к обучению, прогнозированию и принятию решений в незнакомой ситуации. Как и нейронные сети, иммунные сети не нуждаются в заранее известной модели задачи, а строят эту модель на основе полученной информации. Наконец, как нейронные, так и иммунные сети наиболее эффективны при решении плохо формализованных (слабо структурированных) задач, таких как прогнозирование, классификация и управление.

Популяцию антител (детекторов) обозначим $S^b = (s_i^b, i \in [1:|S^b|])$, а популяцию антигенов (данных) – $S^g = (s_j^g, j \in [1:|S^g|])$. Положим, что текущие координаты антитела s_i^b и антигена s_j^g определяют, соответственно, векторы $X_i^b = (x_{i,k},\ k \in [1:|X|])$, $X_j^g = (x_{j,l},\ l \in [1:|X|])$.

Аффинность связей антител s_i^b, s_j^b друг с другом (*BB*-аффинность) определяем как расстояние $\rho(X_i^b, X_j^b) = \rho_{i,j}^{bb}$ между соответствующими векторами. Аналогично определяем *BG*-аффинность связей антитела s_i^b и антигена s_j^g в виде расстояния $\rho(X_i^b, X_j^g) = \rho_{i,j}^{bg}$. Полагаем, что меньшим значениям расстояния $\rho(\cdot,\cdot)$ соответствует большая аффинность связей между агентами. Совокупность всех расстояний $\rho_{i,j}^{bb}$ обозначаем

$$R^{bb} = \left\{\rho_{i,j}^{bb}, i, j \in [1:|S^b|]\right\},$$

а расстояний $\rho_{i,j}^{bg}$ –

$$R^{bg} = \left\{\rho_{i,j}^{bg}, i \in [1:|S^b|], j \in [1:|S^g|]\right\}.$$

При бинарном кодировании координат агентов в качестве меры расстояния $\rho(\cdot,\cdot)$ обычно используют расстояние Хемминга $\|\cdot\|_H$, а в случае целочисленного или вещественного кодирования – евклидову норму $\|\cdot\|_E$ или манхэттенское расстояние $\|\cdot\|_M$.

Таким образом, иммунная сеть может быть представлена в виде графа (не обязательно связного), узлы которого соответствуют антителам и антигенам, а дуги – аффинным связям между ними. Формально иммунную сеть определяет набор

$$\left\langle S^b, S^g, R^{bb}, R^{bg}, S^m, n_b, n_c, b_s, b_b, b_r, b_n \right\rangle,$$

где кроме введенных выше приняты следующие обозначения: S^m – подмножество популяции антител S^b, состоящее из *клеток памяти* иммунной сети;

n_b – число лучших антител, отбираемых для клонирования и мутации из множества антител S^b; n_c – число клонов, создаваемых каждым из отобранных антител; b_s – *степень селекции* (*selection rate*), то есть относительное число улучшенных антител, отбираемых из множества клонированных клеток; b_b – пороговый коэффициент гибели или стимуляции антител в зависимости от значений их ρ^{bg}-аффинности; b_r – пороговый коэффициент сжатия иммунной сети; b_n – коэффициент обновления иммунной сети.

В общем случае схему функционирования иммунной сети можно представить в следующем виде.

1) Инициализируем популяции антител и антигенов S^b, S^g.

2) Для каждого из антигенов $s_i^g, i \in [1:|S^g|]$ выполняем следующие действия.

2.1) Вычисляем *BG*-аффинность антигена s_i^g со всеми антителами сети $s_j^b, j \in [1:|S^b|]$ и отбираем среди указанных антител n_b агентов с максимальной *BG*-аффинностью.

2.2) Клонируем отобранные антитела с коэффициентом клонирования, пропорциональным их *BG*-аффинности, так чтобы общее число клонов было равно $n_b n_c$. Все полученные клоны подвергаем мутации.

2.3) Вычисляем для полученных антител их *BG*-аффинность с антигеном s_i^g. Среди всех полученных антител выбираем $n_d = b_s n_b n_c$ антител, имеющих максимальную *BG*-аффинность, и помещаем в популяцию клеток памяти S^m. Удаляем из популяции S^m те клетки, *BG*-аффинность которых ниже величины b_b.

2.4) Вычисляем *BB*-аффинность всех клеток скорректированной популяции S^m. Удаляем из популяции S^m тех агентов, *BB*-аффинность которых ниже величины b_r (*клональное сжатие*). Расширяем популяцию S^b агентами популяции S^m.

3) Вычисляем *BB*-аффинность агентов расширенной популяции S^b и удаляем из этой сети тех агентов, аффинность которых ниже величины b_r (*сетевое сжатие*).

4) Заменяем b_n процентов худших (с точки зрения *BB*-аффинности) агентов популяции S^b новыми, случайно сгенерированными клетками.

5) Проверяем выполнение условий окончания итераций и в зависимости от результата проверки либо переходим к шагу 2, либо завершаем вычисления.

Модель иммунной сети можно использовать для решения задачи распознавания образов. В этом случае в качестве популяции антигенов выступает набор данных (векторов), которые нужно распознать, а антитела играют роль искомого решения задачи. При этом элементами векторов, кодирующих антигены и антитела, могут являться не только переменные (как в рассмотренной схеме иммунной сети), но любые атрибуты или характеристики распознаваемого объекта.

На основе рассмотренной модели иммунной сети легко построить также иммунный алгоритм решения задачи непрерывной глобальной оптимизации. Базовый алгоритм иммунной оптимизации является частным случаем алгоритма функционирования иммунной сети в предположении, что в сети имеется один антиген и *BG*-аффинность каждого из антител $s_i^b, i \in [1:|S^b|]$ к этому антигену равна соответствующему значению фитнес-функции $\varphi(s_i^b) = \varphi_i$.

4.1.3. Алгоритм на основе искусственной микроиммунной системы

Алгоритм оптимизации на основе искусственной микроиммунной системы (*Micro Artificial Immune System, Micro-AIS*) концептуально близок генетическому микроалгоритму. Отличительной особенностью алгоритма *Micro-AIS* является использование небольших популяций.

Рассмотрим задачу глобальной условной оптимизации в случае, когда множество допустимых значений вектора варьируемых параметров D представляет собой гиперпараллелепипед

$$\Pi = \{X \mid x_k^- \le x_k \le x_k^+, k \in [1:|X|]\}.$$

Схема алгоритма *Micro-AIS* имеет следующий вид.

1) Случайным образом генерируем популяцию $S^b = (s_i^b, i \in [1:|S^b|])$ из $|S^b| = 5$ антител.

2) Определяем аффинность $\varphi(s_i^b) = \varphi_i$ каждого из антител, ранжируем эти антитела в порядке убывания их аффинности и представляем в виде списка $s_{i_1}^b, s_{i_2}^b, s_{i_3}^b, s_{i_4}^b, s_{i_5}^b$.

3) Выполняем клонирование антител $s_{i_1}^b, s_{i_2}^b, s_{i_3}^b, s_{i_4}^b, s_{i_5}^b$ так, что антитело $s_{i_1}^b$ образует $n_1 = 5$ клонов, антитело $s_{i_2}^b$ – $n_2 = 4$ клона и так далее до антитела $s_{i_5}^b$, образующего $n_5 = 1$ клон.

4) Производим мутацию полученных 15 клонов таким образом, чтобы вероятность мутации группы из пяти клонов, полученных от антитела $s_{i_1}^b$, была ниже вероятности мутации группы из четырех клонов, полученных от антитела $s_{i_2}^b$, и так далее до клона антитела $s_{i_5}^b$, имеющего максимальную вероятность мутации.

5) Формируем новую популяцию из пяти агентов по следующему правилу: вычисляем аффинность 15 мутированных клонов; ранжируем эти клоны по их аффинности; создаем новую популяцию из двух лучших клонов и трех других клонов, выбранных случайным образом из 15 указанных клонов.

6) После развития популяции по рассмотренной схеме в течение 10 поколений (так называемый *интервал номинальной сходимости*) новую популяцию формируем по иной схеме – включаем в эту популяцию два лучших клона текущей популяции и добавляем в нее три антитела, сгенерированных случайным образом.

7) Продолжаем итерации до выполнения условий окончания итераций.

Обозначим $S' = (s'_j, j \in [1:|S'|])$ популяцию из $|S'| = 15$ клонов (шаг 4 рассмотренной схемы). Мутацию агентов этой популяции выполняем по следующим правилам.

Вычисляем вероятности

$$\xi_j = \frac{\varphi(s'_j)}{t \sum_{k=1}^{15} \varphi(s'_k)}, \; j \in [1:|S'|], \; t \in [1:10],$$

убывающие, заметим, с ростом номера поколения t в пределах интервала номинальной сходимости.

Мутацию клона s'_j производим в случае выполнения неравенства $U_1(0;1) \geq \xi_j$ с 50 % вероятностью по одной из формул

$$x''_{j,k} = x'_{j,k} + \frac{tU_1(0;1)(x_k^+ - x_k^-)}{15}, \; x''_{j,k} = x'_{j,k} + \frac{U_1(0;1)(x_k^+ - x_k^-)}{15t},$$

где $k \in [1:|X|]$.

Исследования эффективности алгоритма *Micro-AIS* показывают, что этот алгоритм имеет существенные преимущества по сравнению с известной стандартной версией алгоритма клонального отбора *CLONALG* (*CLONal selection ALGorithm*) – алгоритм *Micro-AIS* сходится быстрее, чем *CLONALG* и использует меньший объем оперативной памяти компьютера.

Повысить эффективность алгоритма *Micro-AIS* можно, например, путем использования более быстрых операторов мутации.

4.1.4. Пример алгоритма искусственной иммунной сети

Рассмотрим несколько упрощенный, по сравнению с алгоритмом из 4.1.2, алгоритм искусственной иммунной сети, схема одной итерации которого имеет следующий вид.

1) Из текущей популяции антител S^b выбираем n_b антител с максимальной *BG*-аффинностью.

2) Для каждого антитела из числа отобранных n_b антител создаем n_c клонов и помещаем их во множество S^m.

3) Выполняем мутацию каждого антитела из множества S^m по формуле

$$X_i^m(t+1) = X_i^m(t) + \alpha U_{|X|}(-0,5;0,5), \; i \in [1:|S^m|],$$

где α – коэффициент мутации.

4) Определяем *BG*-аффинность всех антител модифицированного множества S^m и осуществляем его сжатие путем исключения всех антител, кроме n_d лучших.

5) Объединяем популяцию S^b с полученной указанным образом популяцией S^m и выполняем сжатие этой объединенной популяции с сохранением наилучших решений.

Используем данный алгоритм для решения задачи глобальной безусловной минимизации тестовых функций Розенброка, Химмельблау и Растригина (см. Приложение Б). Примем следующие значения свободных параметров алгоритма: размер популяции $|S^b| = 50$; число лучших антител, отбираемых для клонирования и мутации, $n_b = 10$; число лучших клонов, оставляемых после мутации, $n_d = 5$.

Исследование эффективности алгоритма выполнено в режиме мультистарта (число стартов равно $\hat{n} = 30$) для размерностей указанных функций $|X| = 2, 4, 6, 8, 10$. В качестве условия окончания итераций используем стагнацию вычислительного процесса в течение $\delta_t = 20$ итераций. Эффективность алгоритма оцениваем средним числом итераций $\overline{t}_{end}$ по указанному числу стартов, аналогичным средним числом испытаний $\overline{n}_f$, а также лучшим достиг-

Таблица 4.1 **Результаты исследования точности алгоритма для двумерной задачи**[1]

Функция	n_c	α	b_r	$\frac{f^*}{\tilde{f}^*}$	$\frac{X^* = (x_1^*; x_2^*)}{\tilde{X}^* = (\tilde{x}_1^*; \tilde{x}_2^*)}$
Розенброка	50	0,02	0,05	$\frac{0}{0,0000000214}$	$\frac{(1;1)}{(1,00004;1,00006)}$
Химмельблау	50	0,02	0,05	$\frac{0}{0,0000000573}$	$\frac{(3,584;-1,848)}{(3,58440;-1,84815)}$
			0,1	$\frac{0}{0,0000000777}$	$\frac{(-2,805;3,131)}{(-2,80513;3,13127)}$
			0,2	$\frac{0}{0,0000000191}$	$\frac{(3;2)}{(2,99998;2,00003)}$
	100		0,2	$\frac{0}{0,0000000400}$	$\frac{(-3,779;-3,283)}{(-3,77931;-3,28315)}$
Растригина	50	0,02	0,05	$\frac{0}{0,000000122}$	$\frac{(0;0)}{(-0,00002;\ 0,00001)}$

[1] Числитель – точное решение, знаменатель – полученное приближенное решение.

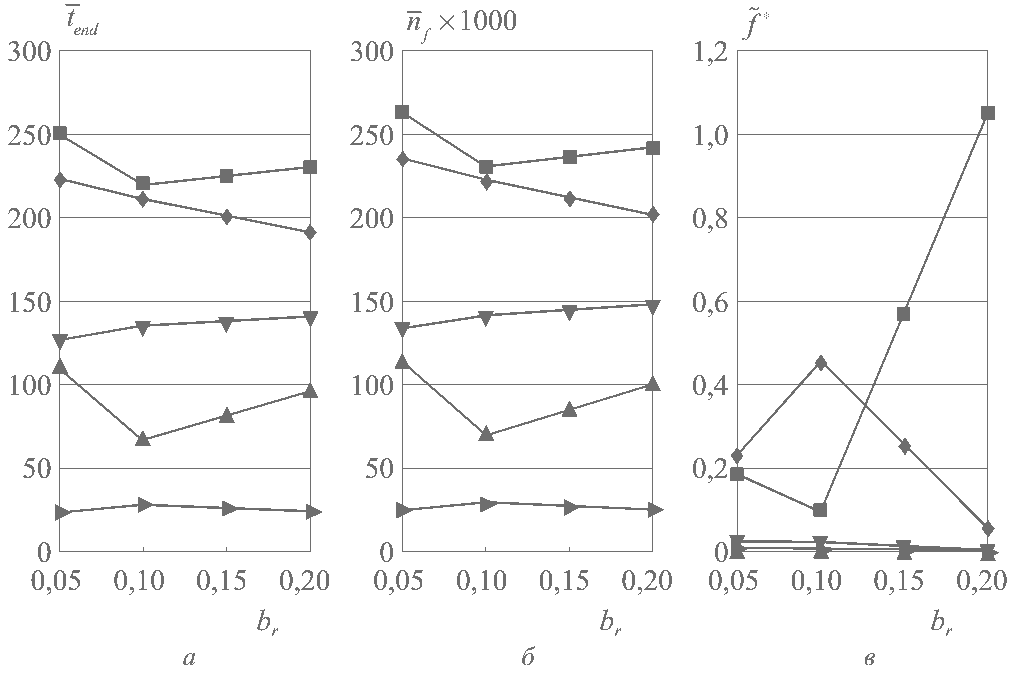


Рис. 4.1. Эффективность алгоритма искусственной иммунной сети, варьируемый параметр – коэффициент сжатия b_r: *а* – по среднему числу итераций $\overline{t}_{end}$; *б* – по среднему числу испытаний $\overline{n}_f$; *в* – по лучшему достигнутому значению целевой функции $\tilde{f}^*$: ► – $|X| = 2$, ▲ – $|X| = 4$, ▼ – $|X| = 6$, ♦ – $|X| = 8$, ■ – $|X| = 10$

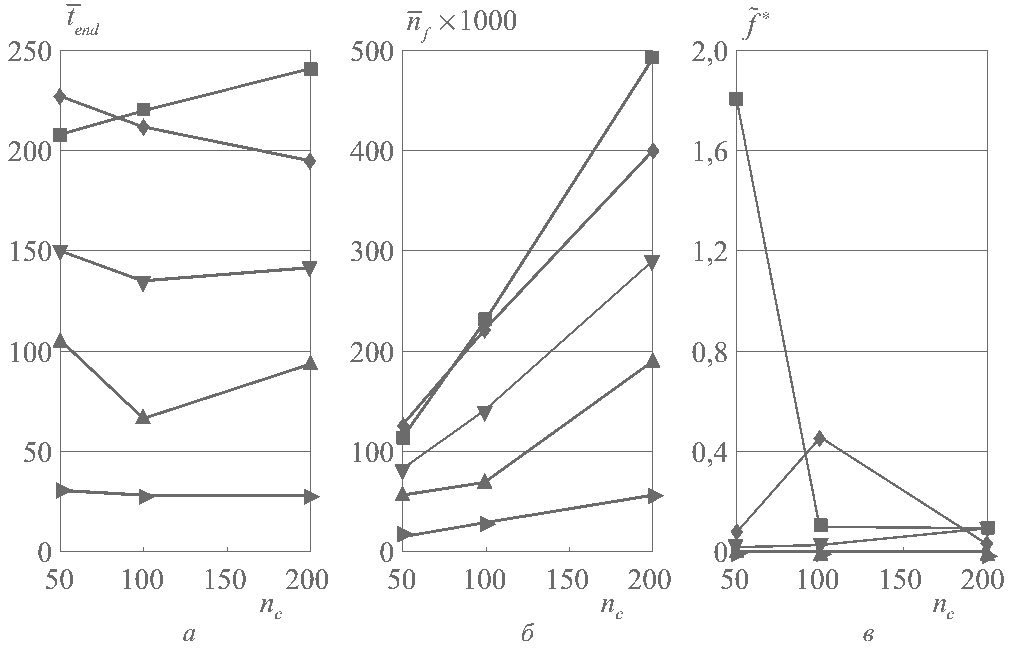


Рис. 4.2. Эффективность алгоритма искусственной иммунной сети, варьируемый параметр – число клонов n_c: ► – $|X| = 2$, ▲ – $|X| = 4$, ▼ – $|X| = 6$, ♦ – $|X| = 8$, ■ – $X = 10$

нутым значением целевой функции $\tilde{f}^*$. Некоторые результаты исследования для случая $|X| = 2$ иллюстрирует табл. 4.1.

Результаты исследования для функции Розенброка иллюстрируют рис. 4.1–4.3, показывающие характер зависимости указанных критериев опти-

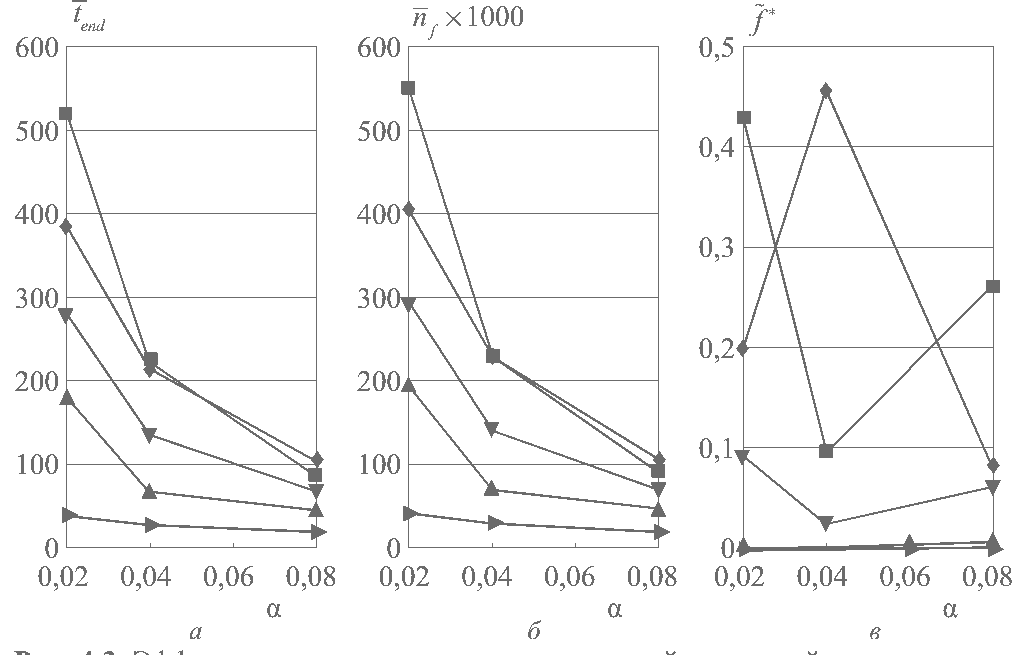


Рис. 4.3. Эффективность алгоритма искусственной иммунной сети, варьируемый параметр – коэффициент мутации α:

► – $|X| = 2$, ▲ – $|X| = 4$, ▼ – $|X| = 6$, ♦ – $|X| = 8$, ■ – $X| = 10$

мальности алгоритма от коэффициента сжатия b_r, числа клонов n_c и коэффициента мутации α.

Рисунки показывают, во-первых, ожидаемую сильную зависимость всех рассматриваемых критериев оптимальности от размерности целевой функции. Так, с увеличением $|X|$ с 2 до 10 среднее число итераций $\overline{t}_{end}$ возрастает примерно в 10 раз, среднее число испытаний $\overline{n}_f$ – в 5 раз, лучшее достигнутое значение целевой функции $\tilde{f}^*$ ухудшается более чем в 10 раз. Рис. 4.1, 4.2 демонстрируют слабую зависимость величины $\overline{t}_{end}$ от значений коэффициента сжатия b_r и числа клонов n_c. В то же время рис. 4.3 показывает сильную зависимость этой величины от коэффициента мутации α. Наиболее ярко проявляется зависимость величины $\overline{n}_f$ от значений параметров n_c, α: с увеличением числа клонов n_c с 50 до 200 эта величина может возрастать в 2–5 раз (рис. 4.2), а с увеличением коэффициента α – во столько же раз убывать (рис. 4.3).

4.2. БАКТЕРИАЛЬНАЯ ОПТИМИЗАЦИЯ

В последние годы поведение бактерий как источник новых подходов к синтезу популяционных алгоритмов поисковой оптимизации привлекает все большее внимание. К настоящему времени разработано несколько алгоритмов оптимизации, объединяемых общим названием *бактериальная оптимизация* (*bacterial optimization*). Известны примеры использования таких алгоритмов для решения прикладных задач из различных предметных областей, например, задачи оптимального управления динамической системой, задачи обучения нейронной сети и т. д. Принято считать, что метод бактериальной оптимизации предложен в работе Пассино (*K. M. Passino*) в 2002 г.

В нижеследующих пунктах представляем бионические предпосылки бактериальных алгоритмов, канонический алгоритм бактериальной оптимизации, кооперативную бактериальную оптимизацию, а также алгоритм, использующий эффект роения бактерий, и некоторые гибридные бактериальные алгоритмы.

4.2.1. Бионические предпосылки

Подвижные бактерии, такие как *кишечная палочка* или *сальмонелла* (лат. *Salmonella*), продвигают себя при помощи вращающихся жгутиков. Чтобы двигаться вперед, все жгутики вращаются в одном направлении. При этом бактерия совершает движение, которое называют *плавание* (*swims*). При вращении жгутиков в разные стороны бактерия разворачивается – совершает движение *кувырок* (*tumble*).

Поведение бактерий обусловлено механизмом, который называется *бактериальным хемотáксисом* (*bacterial chemotaxis*) и представляет собой двигательную реакцию микроорганизмов на химический раздражитель. Данный механизм позволяет бактерии двигаться по направлениям к аттрактантам (чаще всего, питательным веществам) и от репеллентов (потенциально вредных для бактерии веществ).

Рецепторы, детектирующие аттрактанты и репелленты, расположены на полюсах бактерии. В силу малого размера бактерии она не способна уло-

вить разницу концентраций полезных и вредных веществ между полюсами. Градиенты указанных веществ бактерии определяют путем измерения изменений их концентраций при движении. Скорость этого движения может достигать нескольких десятков длин бактерии в секунду. Так, бактерия кишечной палочки (лат. *Escherichia colii*) обычно перемещается со скоростью 10–20 своих длин в секунду.

Если выбранное бактерией направление движения соответствует увеличению концентрации аттрактанта (снижению концентрации репеллента), то время до следующего кувырка увеличивается. Из-за малого, опять же, размера бактерии сильное влияние на ее перемещения оказывает броуновское движение. В результате бактерия только в среднем движется в направлениях к полезным веществам и от вредных веществ.

Рассмотренный механизм движения бактерии не является единственным. Так, у некоторых бактерий имеется один жгутик. Варианты движения бактерии в этом случае обеспечивают различные режимы его вращения и остановки. Однако во всех случаях, если бактерия движется в «нужном» направлении, то продолжительность такого движения увеличивается.

Таким образом, в целом бактериальный хемотаксис можно определить как сложное сочетание плавания и кувырков, позволяющее бактерии удерживаться в местах высокой концентрации питательных веществ и избегать неприемлемой концентрации вредных веществ. В контексте задачи поисковой оптимизации, бактериальный хемотаксис можно также интерпретировать как механизм оптимизации использования бактерией известных пищевых ресурсов и поиска новых, потенциально более ценных областей.

Популяция бактерий достаточной численности может формировать сложные пространственно-временные структуры – эффект структурообразования в популяциях бактерий. Этот эффект может быть вызван как хемотаксисом, так и многими другими причинами. Для некоторых бактерий образование таких структур объясняется регулирующими свойствами их продуктов метаболизма. Аналогичный эффект возможен на основе явлений магнитотаксиса (чувствительности к магнитному полю), биоконвекции, отрицательного геотаксиса (предпочтительное движение микроорганизмов против направления силы тяжести) и других явлений.

4.2.2. Канонический алгоритм бактериальной оптимизации

Рассмотрим задачу многомерной глобальной безусловной оптимизации. Полагаем, что значения фитнес-функции $\varphi(X)$ должны быть максимизированы.

Канонический *алгоритм бактериальной оптимизации* (*Bacterial Foraging Optimization, BFO*) основан на использовании трех следующих основных механизмов: хемотаксис, репродукция, ликвидация и рассеивание.

Пусть $X_{i,r,l} = X_{i,r,l}(t)$ – $(|X| \times 1)$-вектор текущего положения бактерии $s_i \in S$ на итерации t (на t-м шаге хемотаксиса), r-м шаге репродукции и l-м шаге ликвидации и рассеивания. Здесь $i \in [1:|S|]$, $t \in [1:\hat{t}]$, $r \in [1:\hat{t}^r]$, $l \in [1:\hat{t}^l]$, где $|S|$ – четное число агентов в колонии бактерий; $\hat{t}, \hat{t}^r, \hat{t}^l$ – общие числа ите-

раций (шагов хемотаксиса), шагов репродукции, а также шагов ликвидации и рассеивания. Соответствующее значение фитнес-функции обозначим $\varphi_{i,r,l} = \varphi_{i,r,l}(t)$.

Хемотаксис. Процедура хемотаксиса реализует в алгоритме *BFO* локальную оптимизацию. Следующее положение $X'_{i,r,l}$ бактерии s_i (то есть положение на $(t+1)$-м шаге хемотаксиса) определяет формула

$$X'_{i,r,l} = X_{i,r,l} + \lambda_i \frac{V_i}{\|V_i\|_E}, \tag{4.1}$$

где V_i – текущий направляющий $(|X| \times 1)$-вектор шага хемотаксиса бактерии, λ_i – текущая величина этого шага. При плавании бактерии на следующей итерации вектор V_i остается неизменным, то есть верно равенство $V'_i = V_i$. При кувырке бактерии вектор V'_i представляет собой случайный вектор, значения компонент которого находятся в интервале [–1; 1]. Другими словами, при кувыркании имеет место равенство $V'_i = U_{|X|}(-1;1)$. Плавание каждой из бактерий продолжается до тех пор, пока происходит увеличение значений фитнес-функции.

Величина шага хемотаксиса в формуле (4.1) может меняться в процессе поиска, уменьшаясь по некоторому закону с ростом числа итераций t.

Репродукция (*reproduction*). Механизм репродукции имеет своей целью ускорение сходимости алгоритма, то есть его интенсификацию. Назовем текущим *состоянием здоровья* (*health status*) h_i бактерии s_i сумму значений фитнес-функции во всех точках ее траектории от первой до текущей итерации t:

$$h_i = \sum_{\tau=1}^{t} \varphi_{i,r,l}(\tau), \; i \in [1:|S|].$$

Вычислим значения всех величин h_i, отсортируем бактерии в порядке убывания состояний их здоровья и представим результат сортировки в виде линейного списка. Механизм репродукции состоит в том, что на $(r+1)$-м шаге репродукции вторая половина агентов (наиболее слабых) исключается из указанного списка (погибает), а каждый из агентов первой (выжившей) половины списка расщепляется на два одинаковых агента с одинаковыми координатами, равными координатам расщепленного агента.

Пусть, например, s_j, $j \in [1:|S|]$ – один из выживших агентов, положение которого определяет вектор $X_{j,r,l}$. После репродукции этого агента получаем агентов s_j, s_k, $k = |S|/2 + j$, положения которых равны $X_{j,r+1,l} = X_{j,r,l}$, $X_{k,r+1,l} = X_{j,r,l}$.

В результате выполнения процедуры репродукции общее число бактерий в популяции остается низменным и равным $|S|$. Требование четности числа бактерий в популяции обусловлено именно рассмотренной схемой репликации.

Ликвидация и рассеивание (*elimination and dispersal*). Рассмотренных процедур хемотаксиса и репродукции в общем случае недостаточно для отыскания глобального максимума многоэкстремальной фитнес-функции, поскольку

эти процедуры не позволяют бактериям покидать найденные ими локальные минимумы указанной функции. Процедура ликвидации и рассеивания особей призвана преодолеть этот недостаток.

Механизм ликвидации и рассеивания включается после выполнения определенного числа процедур репродукции и состоит в следующем. С заданной вероятностью ξ_e случайным образом выбираем $n < |S|$ бактерий $s_{i_1}, s_{i_2}, \ldots, s_{i_n}$ и уничтожаем их. Вместо каждой из уничтоженных бактерий в случайно выбранной точке пространства поиска создаем нового агента с тем же номером. Число бактерий в колонии в результате выполнения операции ликвидации и рассеивания остается постоянным и равным $|S|$.

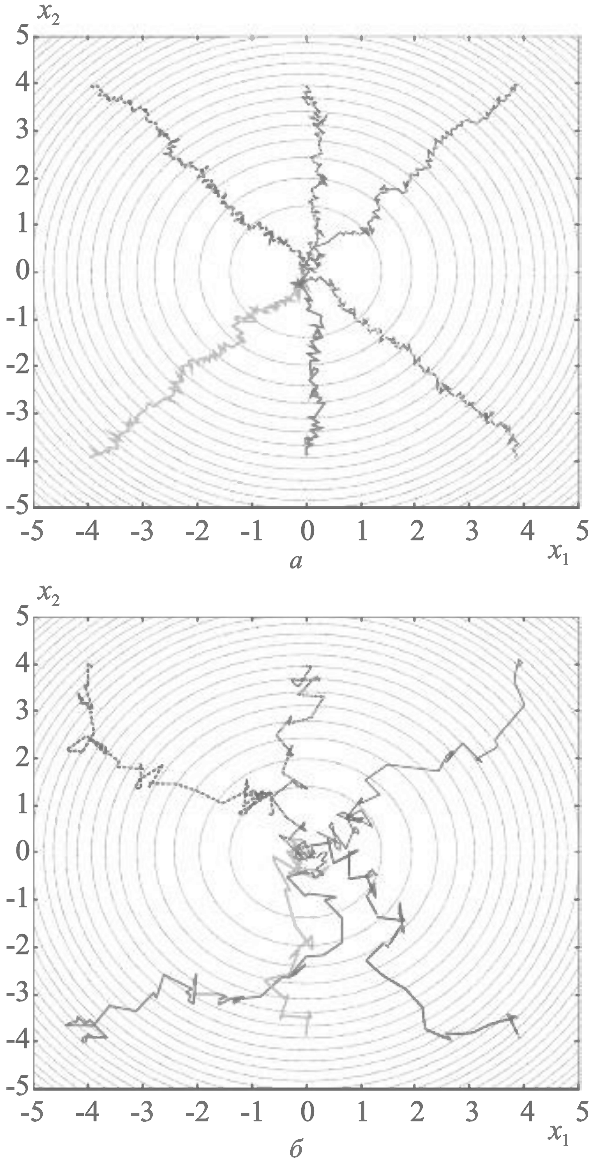


Рис. 4.4. Траектории бактерий: обратная сферическая фитнес-функция; $|X| = 2$: *а* – $\lambda = 0{,}1$; $\hat{t} = 50$; *б* – $\lambda = 0{,}01$; $\hat{t} = 500$

Более строго, процедуру ликвидации и рассеивания определяет следующая последовательность шагов.

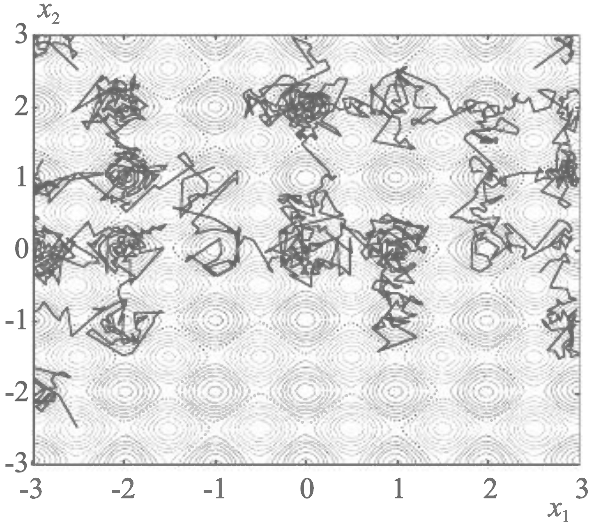
1) Полагаем $j = 1$.

2) Генерируем натуральное случайное число $i_j = U_1(1:|S|)$ и вещественное случайное число $u_j = U_1(0;1)$.

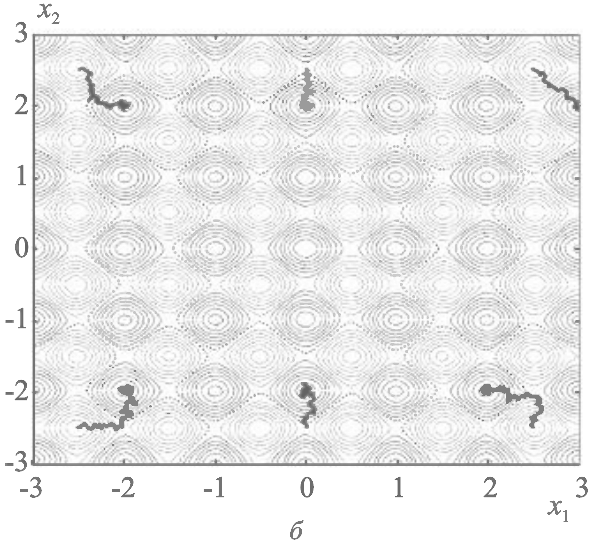
3) Если $u_j > \xi_e$, то новые координаты бактерии s_j определяем по формуле

$$x_{i_j,r,l,k} = U_1(x_k^-; x_k^+),\ k \in [1:|X|], \tag{4.2}$$

где x_k^-, x_k^+ – константы, определяющие диапазон возможных начальных координат бактерий.



а



б

Рис. 4.5. Траектории бактерий: обратная функция Растригина; $|X| = 2$: *а* – $\lambda = 0{,}1$; $\hat{t} = 200$; *б* – $\lambda = 0{,}01$; $\hat{t} = 200$

4) Полагаем $j = j + 1$ и продолжаем итерационный процесс до тех пор, пока не будет уничтожено n бактерий.

Инициализация популяции, то есть определение векторов начальных положений бактерий $X_{i,0,0}(0)$, $i \in [1:|S|]$, производится по формуле, аналогичной формуле (4.2).

Свободными являются следующие параметры алгоритма *BFO*: $\hat{t}, \hat{t}^r, \hat{t}^l$ – общие числа шагов хемотаксиса, репродукции, а также ликвидации и рассеивания соответственно; λ_i, $i \in [1:|S|]$ – величины шагов хемотаксиса; ξ_e – вероятность ликвидации; n – число бактерий, уничтожаемых при выполнении процедуры ликвидации и рассеивания; x_k^-, x_k^+, $k \in [1:|X|]$ – константы, определяющие диапазон возможных начальных координат бактерий, а также координат бактерий, создаваемых в процессе выполнения процедуры ликвидации и рассеивания.

Поведение бактерий при отыскании минимума двумерной унимодальной функции – обратной сферической функции (см. Приложение Б) – с помощью алгоритма *BFO*, не использующего процедуру репродукции, а также процедуру ликвидации и рассеивания, иллюстрирует рис. 4.4, *а*, *б*. На рисунке представлены траектории шести бактерий на протяжении 50 и 500 итераций хемотаксиса соответственно. Начальные положения бактерий в обоих случаях одинаковы и равны (–4; –4), (0; –4), (4; –4), (4; 4), (0;4) и (–4; 4).

Из рисунка ясно, что, как и следовало ожидать, бактерии перемещаются, в среднем, в направлении антиградиента фитнес-функции до ее максимума. Уменьшение шага λ, опять же ожидаемо, увеличивает число итераций, необходимых для достижения окрестности максимума функции, и повышает точность локализации этого максимума.

Приведем также аналогичный пример для двухмерной многоэкстремальной функции – обратной функции Растригина (см. Приложение Б), глобальный максимум которой расположен в точке с координатами (0; 0). На рис. 4.5 для этого случая приведены траектории шести бактерий, стартовавших из точек с координатами (–2,5; –2,5), (0; –2,5), (2,5; –2,5), (2,5; 2,5), (0; 2,5) и (–2,5; 2,5).

На рис. 4.5 проиллюстрировано следующее важное свойство алгоритма *BFO*. При больших значениях шага (рис. 4.5, *а*) алгоритм обеспечивает широкое исследование пространства поиска (диверсификацию поиска). Бактерии при этом могут некоторое число итераций оставаться в окрестностях найденных локальных максимумов фитнес-функции, а затем вырываться из областей их «притяжения» и отыскивать другие локальные либо глобальный максимумы. С другой стороны, в ситуации, когда величина шага мала (рис. 4.5, *б*), бактерии с высокой точностью, в конце концов, концентрируются в окрестностях локальных максимумов и не могут их покинуть.

Рис. 4.4, 4.5 заимствованы из работы *Chen H., Zhu Y., Hu K. Cooperative Bacterial Foraging Optimization // Discrete Dynamics in Nature and Society, 2009, Vol. 2009, pp. 1–17.*

4.2.3. Кооперативная бактериальная оптимизация

Как показано выше, бактерии, перемещающиеся с большим шагом λ, обеспечивают широкое исследование пространства поиска, в то время как бактерии с относительно малым значением этого шага – точную локализацию максимумов фитнес-функции. Это обстоятельство порождает идею разделить процедуру оптимизации колонией бактерий на несколько этапов, каждый из которых занимает некоторую часть общего числа поколений и характеризуется различными значениями параметра λ.

Данную идею реализует *кооперативный бактериальный алгоритм* (*Cooperative BFO algorithm*, *CBFO*). Известны два варианта этого алгоритма – *CBFO-S* и *CBFO-H*. Рассмотрим прежде первый из указанных алгоритмов.

Последовательный (*sequence*) *алгоритм* (*CBFO-S*) представляет собой пример конвейерной гибридизации (гл. 6). Алгоритм использует несколько алгоритмов *BFO*, отличающихся значениями параметра λ и выполняющихся последовательно друг за другом. Выход предыдущего алгоритма *BFO* (лучшие позиции, найденные каждой из бактерий) поступают на вход следующего алгоритма *BFO*. На начальном этапе используем алгоритм *BFO* с большим значением параметра λ, обеспечивающим поиск по всему пространству поиска. В процессе поиска каждая из бактерий сохраняет в своей памяти координаты посещенных точек, а также соответствующие значения фитнес-функции. Позиция с наибольшим значением фитнес-функции объявляется *многообещающим участком* (*promising regions*).

При переходе к следующему этапу текущее значение параметра λ по некоторому правилу уменьшаем и начинаем поиск из найденных на предыдущем этапе многообещающих позиций. Поиск с данным значением λ продолжаем до выполнения условия перехода к очередному этапу, который характеризуется меньшим значением параметра λ. И так далее до выполнения условия окончания итераций.

Число этапов алгоритма является свободным параметром. Числа шагов хемотаксиса в пределах каждого из этапов обычно полагают одинаковыми. Новое значение шага λ определяют, например, по правилу

$$\lambda = \nu\lambda,$$

где $\nu \in (0;1)$ – свободный коэффициент уменьшения шага алгоритма.

Развитием алгоритма *CBFO-S* можно считать *адаптивный* (*adaptive*) *алгоритм бактериальной оптимизации ABFO*, в котором величина шага λ меняется, вообще говоря, на каждой итерации по правилу

$$\lambda(t) = \sum_{\tau=1}^{m} b_\tau \lambda(t-\tau),$$

где m – число предыдущих учитываемых шагов, b_τ – назначаемые исследователем весовые коэффициенты.

Гибридный (*hybrid*) *алгоритм CBFO-H* являет собой пример низкоуровневой гибридизации вложением (гл. 6). Алгоритм относится к классу многопопуляционных алгоритмов и выполняется в два этапа. На первом этапе используем алгоритм *CBFO* с большим значением параметра λ, который после

заданного числа итераций обнаруживает аналогично алгоритму *CBFO-S* многообещающие участки и передает их на следующий этап.

Основной особенностью второго этапа алгоритма *CBFO-H* является разложение пространства поиска $R^{|X|}$ на $|X|/2$ подпространств, каждое из которых имеет размерность, равную двум. Поиск в каждом из указанных двухмерных подпространств выполняет своя колония бактерий, совокупность которых образует *мегаколонию* $\boldsymbol{S} = \{S_i, i \in [1:|X|/2]\}$, где $|X|/2 = |\boldsymbol{S}|$ – число колоний в мегаколонии. Вектор решения задачи получаем путем комбинирования $|X|/2$ лучших двумерных решений, найденных каждой из колоний. Колонии $S_i, i \in [1:|\boldsymbol{S}|]$ выполняют поиск последовательно друг за другом, начиная с колонии S_1, обрабатывающей первые две компоненты вектора X. При работе колонии S_i в качестве значений компонент x_j, $j \in [1:2(i-1)]$ вектора X используем лучшие решения, полученные колониями S_k, $k \in [1:(i-1)]$, а в качестве значений компонент x_j, $j \in [(2i+1):|X|]$ – лучшие решения, полученные на первом этапе алгоритма.

4.2.4. Алгоритм, использующий эффект роения бактерий

Эффект роения бактерий наблюдается для нескольких разновидностей бактерий, включая бактерию сальмонеллы, в полутвердой питательной среде. Эффект обусловлен локальным уменьшением концентрации аттрактанта вследствие его поглощения бактериями. В результате возникает градиент этой концентрации и в хаотическом перемещении бактерий появляется составляющая, направленная по градиенту. Кроме того, бактерии продолжают размножаться. Можно считать, что данный эффект объясняется наличием неявного канала связи между бактериями. В рассматриваемом алгоритме этот канал формализуют следующим образом.

Введем в рассмотрение величины d_{att}, w_{att}, определяющие размеры области наличия аттрактанта, а также аналогичные величины d_{rep}, w_{rep}, характеризующие размеры области репеллентов (имеется в виду двумерная модель популяции бактерий). Указанную выше связь между бактериями задаем функцией

$$\tilde{\varphi}(X) = \sum_{i=1}^{|S|} \tilde{\varphi}_i(X) = \sum_{i=1}^{|S|} \left[-d_{att} \exp\left(-w_{att} \|X - X_i\|_E^2\right) + d_{rep} \exp\left(-w_{rep} \|X - X_i\|_E^2\right) \right], \quad (4.3)$$

где $X_i, i \in [1:|S|]$ – текущие координаты бактерий.

Легко видеть, что алгебраическое значение первого слагаемого в формуле (4.3) возрастает с увеличением евклидова расстояния от точки X до точки X_i. Наоборот, второе слагаемое в этой формуле убывает с ростом указанного расстояния. Таким образом, первая сумма в формуле (4.3) формализует «притяжение» бактерий областью, в которой имеются запасы аттрактанта (а значит, и сосредоточены агенты популяции). Аналогично вторая сумма в этой формуле формализует «отталкивание» бактерий этой областью. Отметим, что значение функции $\tilde{\varphi}(X)$ не зависит от значения целевой функции в точке X.

В качестве фитнес-функции используем функцию

$$\varphi(X) = \varphi(X) + \tilde{\varphi}(X). \quad (4.4)$$

Варьируя значения параметров d_{att}, w_{att}, d_{rep}, w_{rep}, можно управлять поведением колонии бактерий. Так, уменьшение значений параметра d_{att} и увеличение значений параметра w_{rep}, а также увеличение значений величины d_{rep} и уменьшение значений величины w_{rep} увеличивают значения модифицированной фитнес-функции (4.4) и диверсифицируют поиск. Противоположные изменения указанных величин локализуют популяцию в небольших областях, обеспечивая высокую точность поиска.

4.2.5. Гибридные бактериальные алгоритмы

Известно большое число алгоритмов оптимизации, построенных на основе гибридизации бактериального алгоритма с другими популяционными алгоритмами. Рассмотрим в качестве примеров низкоуровневую гибридизацию (гл. 6) бактериального алгоритма с генетическим алгоритмом и алгоритмом роя частиц.

Гибридизация бактериального алгоритма с генетическим алгоритмом

Данный гибридный алгоритм (*BFO-GA*) предложен в одной из работ Кима (*D. H. Kim*), Абрахама (*A. Abraham*) и Чоу (*J. H. Cho*) в 2007 г. Алгоритм представляет собой расширенный генетическими операторами отбора, скрещивания и мутации алгоритм, использующий роение бактерий. В качестве оператора отбора *BFO-GA* использует алгоритм на основе метода рулетки. Скрещивание реализовано с помощью оператора арифметического кроссовера (см. 2.4.2), а мутация – с помощью неравномерного мутатора Михалевича (см. 2.3.2).

Указанные модифицирующие операторы выполняются после завершения процедур хемотаксиса и репродукции бактериального алгоритма перед процедурами ликвидации и рассеивания этого алгоритма. Модификация включает в себя следующую последовательность шагов.

1) Из числа $|S|$ бактерий, полученных после завершения процедур хемотаксиса и репродукции, методом рулетки отбираем $\lceil |S|/2 \rceil$ пар родительских агентов. Рекомендуется использовать вероятность отбора, лежащую в интервале $[0,6; 0,99]$.

2) С помощью арифметического кроссовера выполняем скрещивание каждой из отобранных $\lceil |S|/2 \rceil$ родительских пар и получаем популяцию S' агентов-потомков числом $|S'| = |S|$.

3) Путем применения к каждому из агентов популяции S' оператора неравномерной мутации Михалевича получаем новую популяцию S'', $|S''| = |S|$.

Широкий численный эксперимент, выполненный авторами данного модифицированного бактериального алгоритма, показывает его значительные преимущества по сравнению с исходным алгоритмом как в плане скорости сходимости, так и в плане вероятности локализации глобального экстремума сложных многоэкстремальных функций.

Гибридизация бактериального алгоритма с алгоритмом роя частиц

Данный гибридный алгоритм (*BFO-PSO*) предложен Корани (*W. Korani*) в 2007 г. Основной целью гибридизации было расширение возможностей агентов канонического бактериального алгоритма средствами обмена информацией между ними, позаимствованными в алгоритме *PSO*. Таким образом, алгоритм *BFO-PSO* моделирует процессы хемотаксиса, репродукции, ликвидации и рассеивания колонии бактерий, а также процесс роения агентов.

После завершения шага хемотаксиса в алгоритме *BFO-PSO* выполняются следующие действия.

1) В текущей популяции бактерий S находим лучшее локальное положение каждой бактерии $X_i^*, i \in [1:|S|]$, а также лучшее глобальное положение X_i^{**} бактерии, являющейся соседкой данной бактерии.

2) В соответствии с каноническим алгоритмом роя частиц определяем вектор V_i текущего приращения координат каждой из бактерий $s_i, i \in [1:|S|]$ и вектор ее нового положения X_i'.

Затем по схеме алгоритма *BFO* выполняем операции репродукции, ликвидации и рассеивания бактерий.

4.3. АЛГОРИТМЫ, ВДОХНОВЛЕННЫЕ РОЕМ СВЕТЛЯЧКОВ

В мире насчитывается около двух тысяч видов светлячков, большинство из которых обладают способностью светиться, производя короткие и ритмичные вспышки. Вспышки света производятся с помощью процесса биолюминесценции. Считается, что основными функциями таких вспышек являются привлечение особей противоположного пола и потенциальных жертв. Кроме того, сигнальные вспышки могут служить защитным механизмом предупреждения потенциальных хищников о том, что светлячок горек на вкус. Некоторые тропические светлячки могут синхронизировать свои мерцания, демонстрируя тем самым пример биологической самоорганизации.

Интенсивность света, как функция расстояния от его источника, подчиняется закону обратных квадратов. Кроме того, интенсивность света падает с увеличением расстояния r вследствие поглощения света воздухом. Комбинация этих двух факторов определяет расстояние, на котором светлячки видят друг друга, равное ночью нескольким сотням метров.

Известны два варианта популяционных алгоритмов оптимизации, инспирированных поведением светлячков, – *алгоритм светлячков* (*Firefly algorithm*) и *алгоритм оптимизации роем светлячков* (*Glowworm Swarm Optimization*, *GSO*). Основное различие между *firefly*- и *glowworm*-светлячками состоит в том, что вторые являются бескрылыми.

4.3.1. Алгоритм светлячков

Алгоритм светлячков (*F*-алгоритм) предложен в Кембриджском университете (Великобритания) в 2007 г. Янгом (*X.-Sh. Yang*). Алгоритм использует

следующую модель поведения светлячков: все светлячки могут привлекать друг друга, независимо от своего пола; привлекательность светлячка для других особей пропорциональна его яркости; менее привлекательные светлячки перемещаются в направлении более привлекательного светлячка; яркость излучения данного светлячка, видимая другим светлячком, уменьшается с увеличением расстояния между светлячками; если светлячок не видит возле себя светлячка ярче, чем он сам, то он перемещается случайным образом.

Яркость излучения светлячка $s_i \in S$, $i \in [1:|S|]$ принимаем равной значению фитнес-функции в его текущем положении.

Привлекательность светлячка s_j для светлячка s_i полагаем равной величине

$$\beta_{i,j} = \beta_0 \exp(-\gamma r_{i,j}^2),\ i, j \in [1:|S|], i \neq j, \tag{4.5}$$

где $r_{i,j}$ – расстояние между светлячками s_i, s_j; β_0 – взаимная *привлекательность* (*attractiveness*) светлячков при нулевом расстоянии между ними; γ – вещественная величина, имеющая смысл *коэффициента поглощения света* средой (*light absorption coefficient*).

Для ускорения и упрощения вычислений экспоненциальную функцию в выражении (4.5) иногда заменяют функцией $\dfrac{1}{1+\gamma r_{i,j}^2}$. При этом выражение (4.5) приобретает вид

$$\beta_{i,j} = \frac{\beta_0}{1+\gamma r_{i,j}^2},\ i, j \in [1:|S|], i \neq j. \tag{4.6}$$

Отметим, что формулы (4.5), (4.6) определяют характерное расстояние $r_c = 1/\sqrt{\gamma}$, на котором привлекательность изменяется от β_0 до $\beta_0 e^{-1}$ для случая (4.5) и от β_0 до $\beta_0/2$ для случая (4.6).

Вместо экспоненциального и обратного квадратичного убывания привлекательности с ростом расстояния между светлячками, в качестве функции $\beta(r_{i,j})$ могут быть использованы и другие монотонно убывающие функции, например функция вида

$$\beta(r_{i,j}) = \beta_0 \exp\left(-\gamma r_{i,j}^n\right),\ n \geq 1.$$

Движение светлячка s_i, который притягивается более привлекательным светлячком s_j, определяет формула

$$X_i = X_j + \beta(r_{i,j})\,(X_i - X_j) + \alpha\, U_{|X|}(-1;1),\ i, j \in [1:|S|],\ i \neq j, \tag{4.7}$$

где α – свободный параметр рандомизации. Для обеспечения приемлемого баланса между диверсификацией и интенсификацией поиска значение этого параметра рекомендуют уменьшать с ростом номера поколения t, например, по формуле

$$\alpha = \alpha_\infty + (\alpha_0 - \alpha_\infty)e^{-t},$$

где α_0 – начальное, а α_∞ – его конечное значение.

Для улучшения сходимости алгоритма в формуле (4.7) может быть использован еще один член вида

$$\lambda U_{|X|}(-1;1) \otimes (X_i - X^{**}),$$

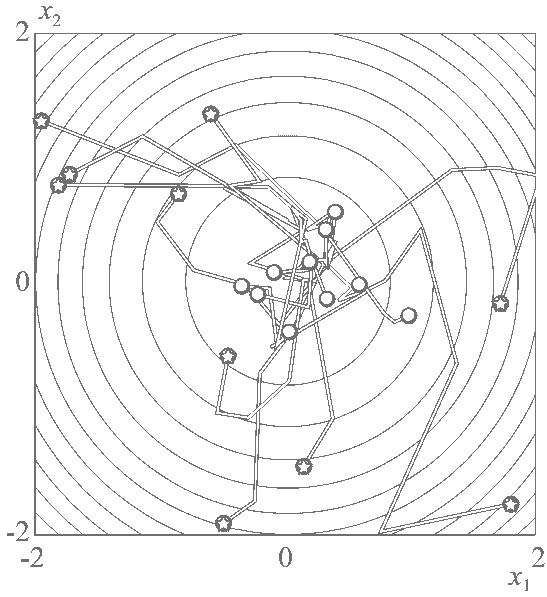


Рис. 4.6. К характеру сходимости алгоритма светлячков: двумерная сферическая фитнес-функция; $|S| = 10$; $\hat{t} = 5$

где λ – параметр, подобный параметру α, а X^{**} – текущее положение глобально лучшего светлячка.

Схема *F*-алгоритма для задачи глобальной безусловной минимизации фитнес-функции имеет следующий вид.

1) Инициализируем начальную популяцию светлячков S и вычисляем значения $\varphi(X_i^0)$ фитнес-функции в начальных точках X_i^0; $i \in [1:|S|]$.

2) Если $\varphi(X_i) < \varphi(X_j)$, то по формуле вида (4.7) перемещаем светлячка s_i в направлении светлячка s_j; $i, j \in [1:|S|], i \neq j$.

3) Вычисляем значения фитнес-функции в полученных точках X_i', $i \in [1:|S|]$.

4) Если условие окончания итераций выполнено, то полагаем лучшее из текущих положений светлячков приближенным решением задачи, иначе переходим к шагу 2.

В большинстве случаев можно использовать следующие значения основных свободных параметров алгоритма: $\beta_0 = 1$; $\alpha \in [0;1]$; $\gamma = 1$.

Рассмотрим два важных частных случая *F*-алгоритма, когда коэффициент поглощения света средой $\gamma \to 0$ и когда $\gamma \to \infty$.

Случай $\gamma \to 0$ означает идеальную прозрачность среды распространения света, так что каждый светлячок видит всех остальных светлячков, независимо от расстояния до них. Легко заметить, что при этом $\beta_{i,j} \to \beta_0$ для всех $i, j \in [1:|S|], i \neq j$ и *F*-алгоритм совпадает с алгоритмом роя частиц, у которого присутствует только социальная компонента.

Случаю $\gamma \to \infty$ соответствует привлекательность $\beta_{i,j}$ в виде *дельта-функции Дирака.* Другими словами, взаимная привлекательность светлячков в данном случае близка к нулю, светлячки перемещаются случайным образом, не видя других светлячков, и *F*-алгоритм вырождается в алгоритм случайного поиска.

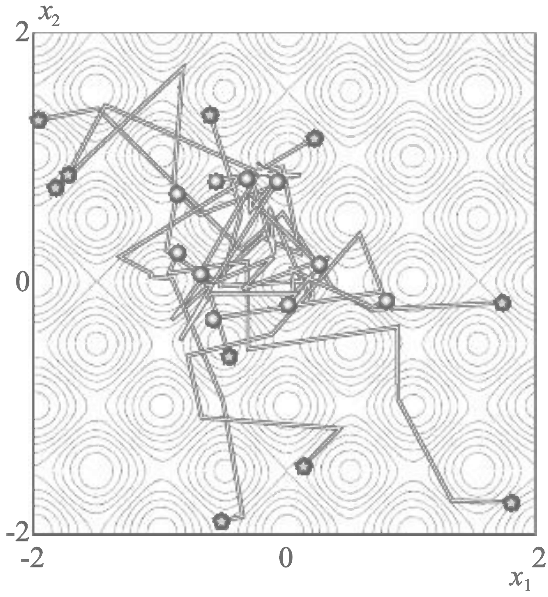


Рис. 4.7. К характеру сходимости алгоритма светлячков: двумерная функция Растригина; $|S| = 10$; $\hat{t} = 7$

F-алгоритм представляет собой некоторый промежуточный вариант между двумя указанными крайностями. Алгоритм использует такие значения своих свободных параметров, которые обеспечивают превосходство как над алгоритмом случайного поиска, так и над алгоритмом *PSO*.

В общем случае значение величины γ зависит от размерности пространства поиска и ландшафта фитнес-функции. Известна рекомендация, по которой эта величина должна принимать одно из значений

$$\gamma = \frac{\gamma_0}{r_{\max}}, \ \gamma = \frac{\gamma_0}{r_{\max}^2},$$

где $\gamma_0 \in [0;1]$, $r_{\max} = \max\limits_{i,\, j \in [1:|S|]} \left\| X_i, X_j \right\|_E$.

Характер сходимости алгоритма светлячков при поиске минимума одноэкстремальной сферической функции иллюстрирует рис. 4.6; аналогичный рис. 4.7 иллюстрирует сходимость алгоритма для двумерной функции Растригина (см. Приложение Б). Звездочками на рисунках обозначены начальные положения агентов, а кружками – их положения после указанного числа итераций.

4.3.2. Алгоритм оптимизации роем светлячков

Алгоритм *GSO* представили индийские ученые Кришнананд (*K. N. Krishnanand*) и Гхош (*D. Ghose*) в 2005 г. Состояние светлячка $s_i, i \in [1:|S|]$ в этом алгоритме определяют следующие переменные: X_i – текущее положение светлячка в пространстве поиска; l_i – *уровень светимости* (*luciferin level*);

r_i – *радиус окрестности* (*neighborhood range*). Основным содержанием каждой итерации является обновление значений указанных переменных.

Уровень светимости светлячка s_i обновляем по формуле

$$l_i' = (1-\rho)\, l_i + \gamma\varphi(X_i),\ i \in [1:|S|],$$

где ρ – положительный свободный параметр алгоритма, моделирующий распад флуоресцирующего вещества и снижение привлекательности светлячка. Отличные от нуля значения параметра ρ обеспечивают алгоритм памятью. Параметр γ определяет относительные веса текущей светимости агента и значения его фитнес-функции.

Светлячок s_j считается соседом светлячка s_i, $i, j \in [1:|S|]$, $i \neq j$ при выполнении следующих двух условий: евклидово расстояние между светлячками s_i, s_j не превышает текущий радиус окрестности r_i; текущий уровень светимости светлячка s_j превышает этот же уровень светлячка s_i, то есть $l_j > l_i$. Если светлячок имеет несколько соседей, то случайным образом выбираем одного из них с вероятностью, пропорциональной уровням их светимости (правило рулетки).

Положим, что по рассмотренной схеме светлячком s_i выбран светлячок s_j. Тогда новое положение светлячка s_i определяет формула

$$X_i' = X_i + \lambda \frac{X_j - X_i}{\left\| X_j - X_i \right\|},\ i, j \in [1:|S|],\ i \neq j,$$

где λ – постоянная величина шага (свободный параметр алгоритма).

Новый радиус окрестности светлячка s_i определяем в соответствии с выражением

$$r_i' = \min\left(r_{\min}, \max\left(0,\ r_i + b\,(n_n - |N_i|)\right)\right),\ i \in [1:|S|], \tag{4.8}$$

где N_i – текущее множество соседей светлячка s_i; $r_{\min}$ – минимально допустимый радиус окрестности; n_n – *желательное число соседей*; b – положительная константа. Последние три величины – свободные параметры алгоритма.

Рассмотрим некоторые модификации представленного канонического алгоритма *GSO*.

Для диверсификации поиска в случае, когда число соседей светлячка s_i в окрестности N_i менее желательного числа n_n, радиус окрестности $r_{\min}$ может быть по тому или иному правилу расширен (в простейшем случае, положим, на 5 %).

В той же ситуации возможно иное решение – случайное перемещение светлячка в пределах гиперкуба с центром в точке X_i и длиной ребра, равной величине шага λ,

$$x_{i,j}' = x_{i,j} + \lambda U_1(-0{,}5;\, 0{,}5),\ j \in [1:|X|].$$

Если значение фитнес-функции в новой точке X_i' лучше ее значения в точке X_i, то оставляем светлячка в точке X_i', в противном случае – возвращаем в прежнюю точку X_i.

Новый радиус r_i' окрестности светлячка s_i может вычисляться не по формуле (4.8), а по формуле вида

$$r_i' = \frac{r_{\max}}{1 + \varepsilon\, d_i},$$

где $r_{\max}$ – максимально допустимое значение этой величины, $\varepsilon > 0$ – заданная константа, d_i – средняя текущая плотность агентов в окрестности N_i:

$$d_i = \frac{|N_i|}{\pi\, r_i^2}.$$

Величины $r_{\max}$, ε представляют собой свободные параметры алгоритма.

Наиболее сложной является модификация алгоритма *GSO*, использующая специальный механизм организации агентов в группы. В каждой из групп в этом случае выделяют одного из агентов в качестве ее *владельца* (*master*), который определяет миграцию всех *подчиненных* (*slave*) агентов группы. В исходном состоянии подчиненные агенты равномерно распределены случайным образом в гиперсфере радиуса r_d с центром, совпадающим с начальным положением владельца группы. В процессе поиска величина r_d может варьироваться, уменьшаясь по тому или иному закону с ростом числа итераций t.

4.4. СОРНЯКОВЫЙ АЛГОРИТМ

Алгоритм сорняковой оптимизации (*Invasive Weed Optimization*, *IWO*) вдохновлен таким общераспространенным явлением, как колонизация сельскохозяйственных угодий сорняками. Алгоритм предложен в 2006 г. иранскими учеными Мехрабианом (*A. R. Mehrabian*) и Лукасом (*C. Lucas*) и основан на моделировании таких свойств, как посев, рост и конкуренция в колонии сорняка.

4.4.1. Биологические основы

Говоря простым языком, сорняк – это любое растение, растущее там, где оно нежелательно. В общем случае, любое растение может быть классифицировано как сорняк. Однако обычно термин используют по отношению к тем растениям, чьи экспансионистские свойства представляют серьезную угрозу для культивируемых растений. В силу высокой актуальности борьбы с сорняками в мире издается значительное число журналов, полностью посвященных систематике сорняков, их экологии и физиологии, методам контроля и так далее.

Наиболее интересной особенностью проблемы сорняков является всеобщее убеждение, что они всегда побеждают, и чем больше люди стараются, тем сильнее они становятся. Пространственные возможности для распространения сорняков создают системы земледелия человека. Сорняки вторгаются в эти пространства с помощью сначала рассеивания семян, затем колонизации и, наконец, оккупации полей. Биоразнообразие сорняков, а также их высокая адаптируемость к местным условиям обеспечивают высокую эффективность указанных процессов.

В зависимости от вида растения сорняк может воспроизводиться с использованием или без использования половых клеток. Половое размножение осуществляется с помощью семян, которые формируются в материнском растении

после того, как яйцеклетка оплодотворяется пыльцой. Затем эти семена распространяются с помощью ветра, воды, животных и так далее (пространственное рассеивание). Если семена попадают в подходящие для жизни условия, они прорастают, вырастают во взрослые растения, затем цветут и производят семена.

Основным механизмом, определяющим динамику сообщества любых растений, является естественный отбор, из которого выделяют два крайних типа: *r*-отбор и *K*-отбор. Реальные стратегии отбора лежат между этими предельными типами.

Можно сказать, что девизом *r*-отбора является *живи быстро, размножайся быстро, умирай молодым* (*live fast, reproduce quick, die young*). Данный тип отбора необходим для успеха в нестабильной, непредсказуемой окружающей среде. При *r*-отборе предпочтительны такие качества, как высокая плодовитость, маленький размер семян и приспособленность к рассеиванию их на большое расстояние.

K-отбор использует принцип *живи медленно, размножайся медленно, умирай в старости* (*live slow, reproduce slow, die old*). Этот тип отбора необходим для успеха в стабильной, предсказуемой окружающей среде, когда вероятно тяжелое соперничество за ограниченные ресурсы между конкурентоспособными индивидуумами. Ситуация имеет место, если размер популяции в ареале обитания близок к максимуму, который он способен вместить. При *K*-отборе предпочтительны такие качества индивидов, как большой размер семян, длинная жизнь, небольшое потомство, за которым требуется интенсивный уход.

4.4.2. Схема алгоритма

Рассмотрим задачу глобальной безусловной максимизации фитнес-функции. В алгоритме *IWO* модель поведения сорняков при колонизации учитывает следующие базовые свойства процесса.

1) Распределение конечного числа семян по всей области поиска (инициализация популяции).

2) Производство выросшими растениями семян в зависимости от приспособленности растений (воспроизводство).

3) Распределение произведенных семян в случайном порядке по области поиска (пространственное распределение).

4) Повторение шагов 2, 3 до тех пор, пока не будет достигнут заданный максимум числа растений.

5) Отбор растений с более высокой приспособленностью, их воспроизводство и пространственное распределение (конкурентное исключение).

6) Повторение шага 5 до выполнения условия окончания процесса.

Инициализация популяции. Начальные положения сорняков принимаем случайными, равномерно распределенными в некотором гиперпараллелепипеде $\Pi^0 = (X^- \le X \le X^+)$:

$$X_i^0 = X^- + U_{|X|}(0;1) \otimes \left(X^+ - X^-\right),\ i \in [1 : \left|S^0\right|].$$

Здесь X^-, X^+ – векторы свободных параметров алгоритма, $\left|S^0\right|$ – начальный размер популяции.

Воспроизводство. В оригинальном алгоритме *IWO* число семян n_i, произведенных сорняком s_i, линейно зависит от его текущей приспособленности $\varphi_i = \varphi(X_i)$ и определяется формулой (рис. 4.8)

$$n_i = \frac{n_{\max} - n_{\min}}{\varphi_{\max} - \varphi_{\min}} \varphi_i + \frac{\varphi_{\max}\, n_{\min} - \varphi_{\min}\, n_{\max}}{\varphi_{\max} - \varphi_{\min}}, \; i \in [1:|S|],$$

где $n_{\min}, n_{\max}$ – заданные константы, представляющие собой минимальное и максимальное числа семян, которые могут быть произведены каждым из сорняков на текущей итерации t;

$$\varphi_{\min} = \min_{i \in [1:|S|]} \varphi(X_i), \; \varphi_{\max} = \max_{i \in [1:|S|]} \varphi(X_i).$$

Отметим следующее обстоятельство. Во многих популяционных алгоритмах агентам, имеющим низкую приспособленность, не позволяют размножаться, хотя существует вероятность того, что некоторые из них несут более полезную информацию, чем их конкуренты. Схема воспроизводства, используемая алгоритмом *IWO*, дает шанс менее приспособленным агентам выжить и размножиться подобно тому, как это имеет место в природе.

Пространственное распределение. Произведенные сорняком s_i семена распределяем в окрестности родительского растения в соответствии с нормальным законом распределения, имеющим нулевое математическое ожидание и стандартное отклонение σ:

$$X_{i,j} = X_i + N_{|X|}(0, \sigma), \; i \in [1:|S|], \; j \in [1:n_i].$$

Стандартное отклонение σ в этом выражении зависит от текущего номера поколения t, уменьшаясь с ростом этого номера по формуле

$$\sigma(t) = \left(\frac{\hat{t} - t}{\hat{t}}\right)^m \left(\sigma_b - \sigma_e\right) + \sigma_e, \tag{4.9}$$

где σ_b, σ_e – начальное и конечное значения стандартного отклонения σ, $m > 1$ – свободный *параметр модуляции*, определяющий характер функции $\sigma(t)$.

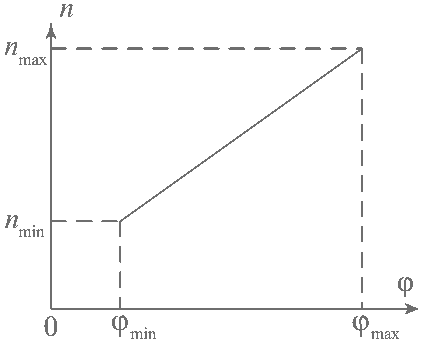


Рис. 4.8. К схеме воспроизводства сорняков

Формула (4.9) обеспечивает уменьшение вероятности распространения семян вдали от родительского растения с ростом номера итераций, уменьшая тем самым диверсификационные и повышая локализационные свойства алгоритма. Можно сказать, что данная схема изменения стандартного отклонения σ реализует механизм перехода от *r*-отбора к *K*-отбору в процессе эволюции популяции.

Конкурентное исключение. Обозначим через S_i субпопуляцию сорняков, являющихся потомками растения s_i. Тогда до достижения всей популяцией сорняков своего максималь-

ного размера, равного $|S|^{\max}$, новую популяцию S' формируем путем объединения текущей популяции S со всеми субпопуляциями S_i:

$$S' = S \bigcup \left(\bigcup_{i=1}^{|S|} S_i \right).$$

Конкурентное исключение начинает функционировать после превышения популяцией размера $|S|^{\max}$ и заключается в уничтожении сорняков с меньшей приспособленностью до достижения популяцией этого размера.

Таким образом, растения и их потомки оцениваются вместе, и тем из них, которые обладают лучшей приспособленностью, позволяют размножаться. Данный механизм позволяет растениям с меньшей приспособленностью воспроизводиться, и, если их потомки обладают хорошей приспособленностью, они могут выжить.

Кроме размеров гиперпараллелепипеда Π^0, свободными параметрами алгоритма являются начальная и максимальная численности популяции $|S^0|$, $|S|^{\max}$; минимальное и максимальное числа семян $n_{\min}, n_{\max}$, которые могут быть произведены каждым из сорняков на данной итерации; начальное и конечное значения σ_b, σ_e стандартного отклонения σ; параметр модуляции m.

Выполненное авторами алгоритма широкое исследование его эффективности на тестовых функциях «сфера», Гривенка и Растригина (Приложение Б) показывает, что алгоритм сравним, а в некоторых случаях и лучше таких популяционных алгоритмов, как генетический, меметический, роя частиц и лягушачий.

4.5. КУКУШКИН ПОИСК

Алгоритм кукушкиного поиска (*Cuckoo Search*, *CS*) предложен и разработан Янгом (*X.-Sh. Yang*) и Дебом (*S. Deb*) в 2009 г. Алгоритм вдохновлен поведением кукушек в процессе вынужденного гнездового паразитизма, когда некоторые виды кукушек подкладывают яйца в гнезда птиц других видов.

4.5.1. Биологические предпосылки

Кукушки – удивительные птицы не только из-за прекрасных звуков, которые они издают, но и из-за своей паразитической стратегии размножения. Такие виды кукушек, как *Ani* и *Guira*, откладывают яйца в коллективные гнезда вместе с другими кукушками, хотя могут выбрасывать яйца конкурентов, чтобы увеличить вероятность вылупления их собственных птенцов. Целый ряд видов кукушек занимается гнездовым паразитизмом, подкладывая свои яйца в гнезда других птиц, как своего вида, так и часто других видов.

Некоторые птицы могут конфликтовать с вторжением кукушек. К примеру, если хозяин гнезда обнаружит в нем яйца иного вида, то он либо выбросит эти яйца, либо просто покинет данное гнездо и соорудит на новом месте другое. Известны виды кукушек, например *Tapera*, которые эволюционировали

таким образом, что их женские особи могут имитировать цвет и структуру яиц хозяина. Это снижает вероятность того, что гнездо будет покинуто или яйцо кукушки будет выброшено, и тем самым увеличивает вероятность воспроизводства этого вида кукушек.

Паразитарные кукушки часто выбирают гнезда, где хозяин только что отложил свои яйца. При этом из яиц кукушек птенцы появляются несколько раньше, чем птенцы хозяина гнезда. Как только первый такой птенец вылупился, его первым инстинктивным действием является выталкивание остальных яиц из гнезда, что увеличит долю продовольствия, предоставляемую птенцу хозяином гнезда, и повышает вероятность выживания этого птенца. Кукушонок может имитировать звуки других птенцов, чтобы получить доступ к дополнительному питанию.

4.5.2. Схема алгоритма

В алгоритме *CS* каждое яйцо в гнезде представляет собой решение, а яйцо кукушки – новое решение. Цель заключается в использовании новых и потенциально лучших (кукушкиных) решений, чтобы заменить менее хорошие решения в гнездах. В простейшем варианте алгоритма в каждом гнезде находится по одному яйцу.

Положим, что речь идет о задаче глобальной безусловной оптимизации, в которой фитнес-функция подлежит максимизации.

Алгоритм основан на трех следующих правилах: каждая кукушка откладывает одно яйцо за один раз в случайно выбранное гнездо; лучшие гнезда с яйцами высокого качества (высоким значением пригодности) переходят в следующее поколение; яйцо кукушки, отложенное в гнездо, может быть обнаружено хозяином с некоторой вероятностью $\xi_a \in (0;1)$ и удалено из гнезда.

Схему алгоритма *CS* можно представить в следующем виде.

1) Инициализируем популяцию $S = \{s_i, i \in [1:|S|]\}$ из $|S|$ хозяйских гнезд и кукушку, то есть определяем векторы X_i^0 и вектор начального положения кукушки X_c.

2) Выполняя полеты Леви (п. 1.4.2), находим новое положение кукушки X_c.

3) Случайным образом выбираем гнездо $s_j, j \in [1:|S|]$ и, если $\varphi(X_c) > \varphi(X_j)$, заменяем яйцо в этом гнезде на яйцо кукушки, то есть полагаем $X_j = X_c$.

4) С вероятностью ξ_a удаляем из популяции некоторое число худших случайно выбранных гнезд (включая, быть может, гнездо s_j) и по правилам шага 1 строим такое же число новых гнезд.

5) Если условие окончания итераций выполнено, то в качестве приближенного решения задачи принимаем лучшее из найденных решений и заканчиваем вычисления, иначе переходим к шагу 2.

Начальные положения гнезд и кукушки принимаем случайными, равномерно распределенными в некотором гиперпараллелепипеде.

Полеты Леви кукушки реализуем по формуле

$$X_c' = X_c + V \otimes L_{|X|}(\lambda). \tag{4.10}$$

Обычно полагают все компоненты вектора V одинаковыми и равными v, где величина v связана с масштабами области поиска.

Основными свободными параметрами алгоритма являются константы, определяющие траекторию полетов Леви, и вероятность удаления гнезд ξ_a.

Известно несколько модификаций алгоритма *CS*. В каноническом алгоритме *CS* кукушка в своих полетах Леви никак не учитывает информацию о лучших найденных решениях. В *модифицированном алгоритме поиска кукушки* (*Modified Cuckoo Search*, *MCS*) компоненты вектора V вычисляют по формуле вида

$$V = U_1(0;1)\left(X_{i_*} - X_{i_1}\right),$$

где X_{i_*} – лучшее текущее гнездо, а X_{i_1} – случайно выбранное гнездо; $i_* \neq i_1$. Так определенный вектор V обеспечивает большую вероятность полета кукушки к гнездам, имеющим высокую приспособленность.

С другой стороны, в каноническом алгоритме *CS* вероятность ξ_a и параметры полета Леви являются фиксированными константами. С целью диверсификации поиска для ранних поколений целесообразно использовать большие значения величин ξ_a, v. На завершающих итерациях для повышения точности локализации экстремума разумны меньшие значения указанных величин. *Улучшенный алгоритм поиска кукушки* (*Improved Cuckoo Search, ICS*) использует динамические значения параметров ξ_a, v:

$$\xi_a(t) = \xi_a^{\max} - \frac{t}{\hat{t}}(\xi_a^{\max} - \xi_a^{\min}),$$

$$v(t) = v^{\max}\exp(dt),\ d = \frac{1}{\hat{t}}\ln\left(\frac{v^{\min}}{v^{\max}}\right).$$

Здесь $\xi_a^{\min}$, $\xi_a^{\max}$, $v^{\min}$, $v^{\max}$ – заданные константы.

4.5.3. Эффективность алгоритма

Результаты авторов алгоритма *CS* по исследованию его эффективности показывают, что этот алгоритм может превосходить генетический алгоритм и алгоритм роя частиц в случае сложных мультимодальных целевых функций. Важным достоинством алгоритма *CS* является небольшое число свободных параметров. Это делает алгоритм более универсальным и надежным в сравнении со многими другими популяционными алгоритмами. Многообещающими являются гибридизации алгоритма *CS* с другими популяционными алгоритмами.

Пример 4.1. Рассмотрим задачу глобальной безусловной минимизации функций «сфера» и Растригина (см. Приложение Б). Исследуем эффективность алгоритма *CS* в зависимости от размера популяции $|S|$ и величины вероятности ξ_a. В качестве критериев эффективности алгоритма используем среднее число итераций $\overline{t}_{end}$, определяемое началом стагнации вычислений, а также надежность – оценку $\tilde{\xi}$ вероятности локализации глобального минимума фитнес-функции по результатам мультистарта из 100 запусков

алгоритма. Полагаем, что критерий стагнации вычислительного процесса имеет вид $\delta_t = 20$.

Результаты исследования представлены на рис. 4.9 – 4.12.

Рис. 4.9, 4.10 иллюстрируют зависимость среднего числа итераций $\overline{t}_{end}$ от размера популяции $|S|$ и показывают, прежде всего, быстрое уменьшение этой величины с ростом числа агентов в популяции. Следует отметить, что при этом среднее число испытаний $\overline{n}_f$ для сферической функции имеет чет-

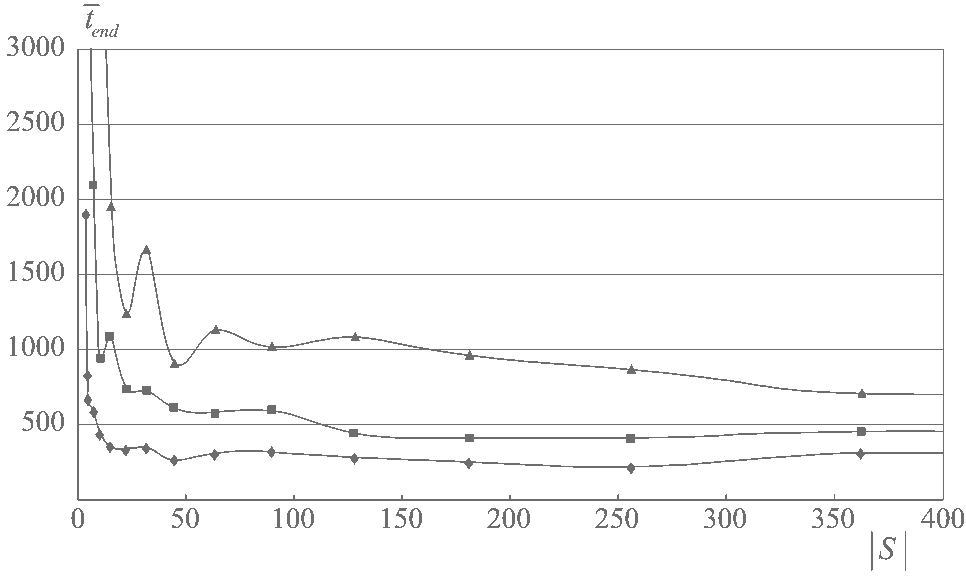


Рис. 4.9. Число итераций $\overline{t}_{end}$ в зависимости от размера популяции $|S|$: сферическая функция; $\xi_a = 0{,}25$: ♦ – $|X| = 2$; ■ – $|X| = 4$; ▲ – $|X| = 8$

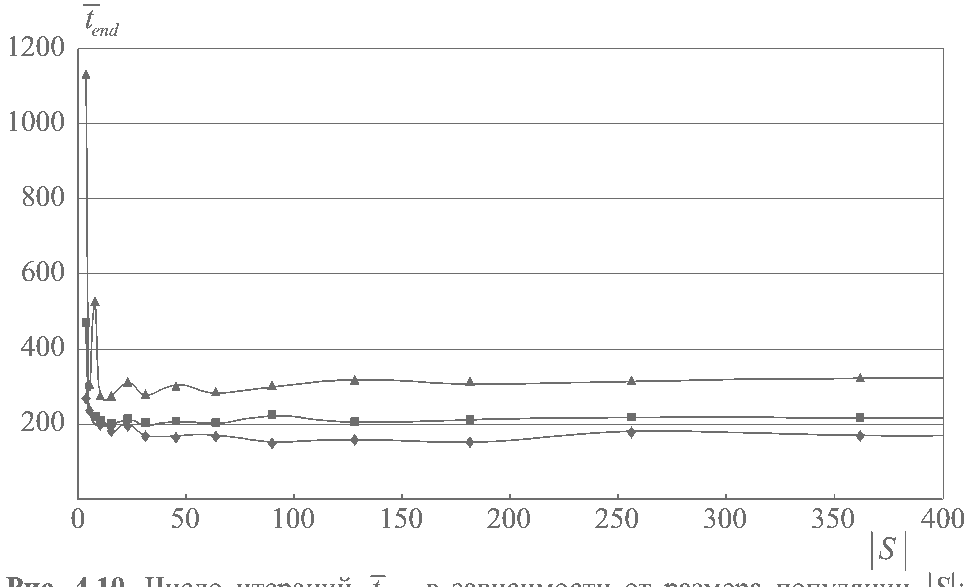


Рис. 4.10. Число итераций $\overline{t}_{end}$ в зависимости от размера популяции $|S|$: функция Растригина; $\xi_a = 0{,}25$: ♦ – $|X| = 2$; ■ – $|X| = 4$; ▲ – $|X| = 8$

кий минимум, а для функции Растригина демонстрирует почти линейный рост (рис. 4.11, 4.12).

Результаты исследования показывают, что при всех рассмотренных размерах популяции, начиная с $|S| = 4$, оценка $\tilde{\xi}$ вероятности локализации глобального минимума сферической функции равна 100 %. В случае функции Растригина эта оценка становится равной 100 % при $|S| = 8$, 11 и 23 для размерности вектора варьируемых параметров $|X| = 2,\ 4,\ 8$ соответственно.

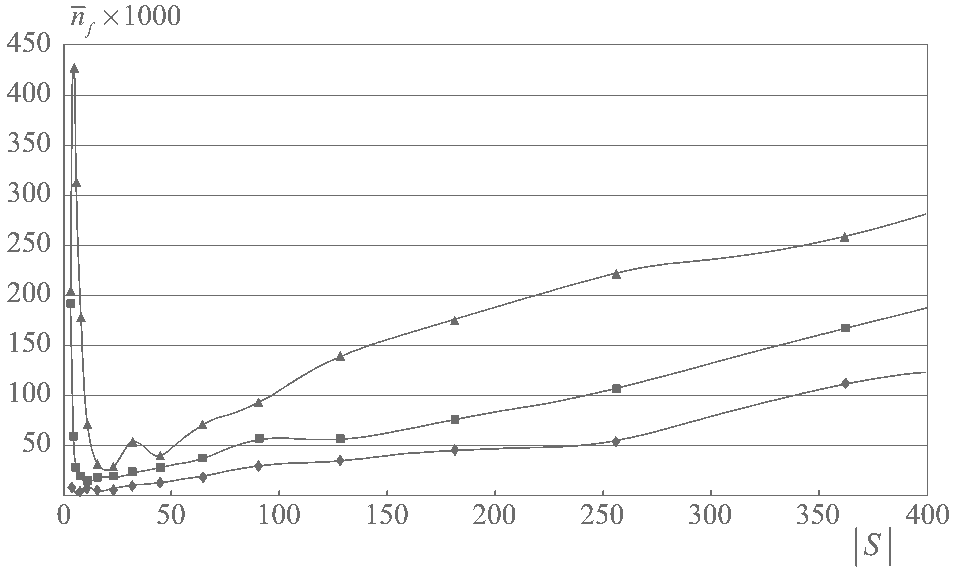


Рис. 4.11. Число испытаний $\overline{n}_f$ в зависимости от размера популяции $|S|$: сферическая функция; $\xi_a = 0{,}25$: ♦ – $|X| = 2$; ■ – $|X| = 4$; ▲ – $|X| = 8$

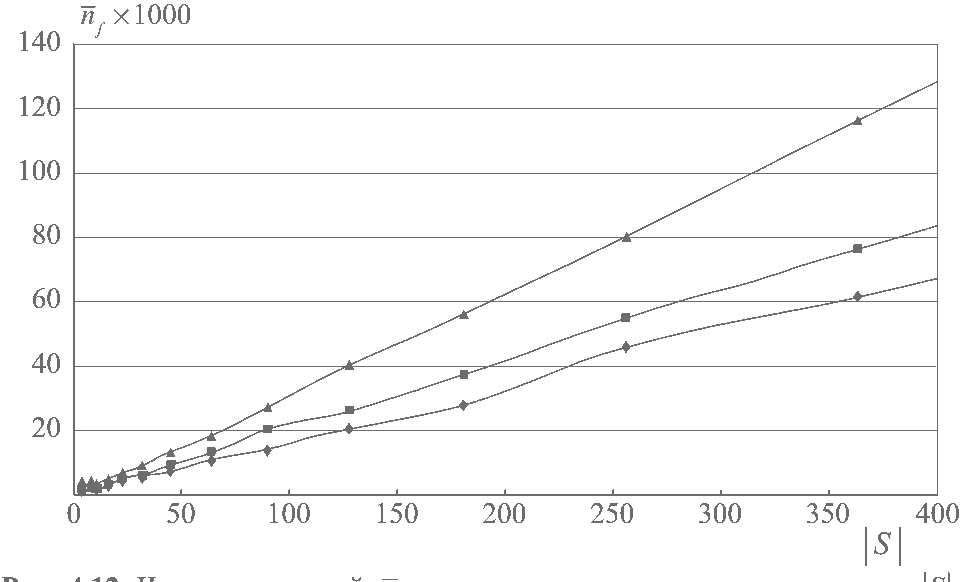


Рис. 4.12. Число испытаний $\overline{n}_f$ в зависимости от размера популяции $|S|$: функция Растригина; $\xi_a = 0{,}25$: ♦ – $|X| = 2$; ■ – $|X| = 4$; ▲ – $|X| = 8$

Рис. 4.13, 4.14 иллюстрируют зависимость числа итераций $\overline{t}_{end}$ от величины вероятности ξ_a.

Результаты, представленные на этих рисунках, показывают, что для обеих рассматриваемых функций в широком диапазоне изменения вероятности ξ_a (от 0 до примерно 0,8) число итераций $\overline{t}_{end}$ относительно слабо зависит от этой величины. При высокой вероятности ξ_a для обеих функций резко возрастает число итераций, что объясняется превалированием в данном случае стратегии диверсификации поиска над стратегией его интенсификации.

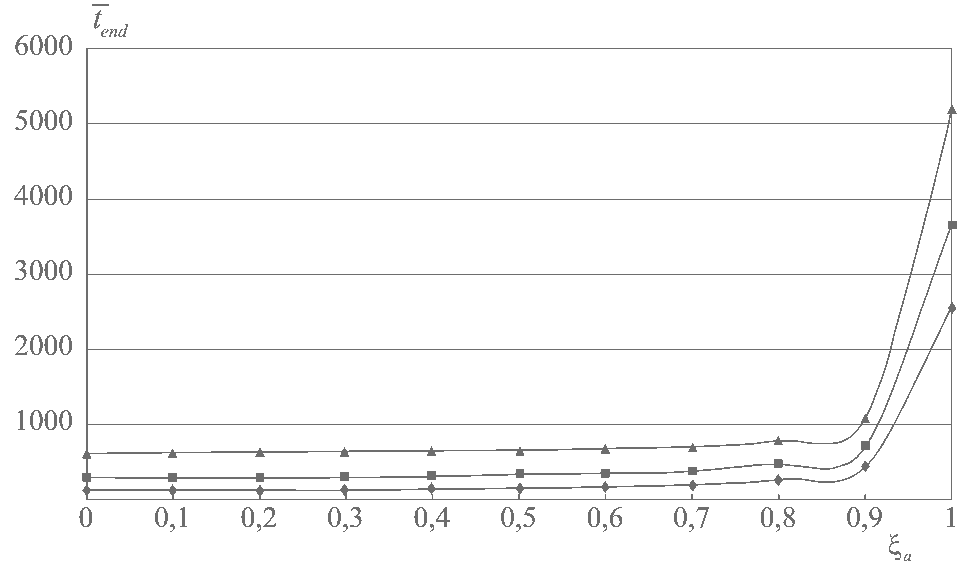


Рис. 4.13. Число итераций $\overline{t}_{end}$ в зависимости от величины вероятности ξ_a: сферическая функция; $|S| = 32$: ♦ – $|X| = 2$; ■ – $|X| = 4$; ▲ – $|X| = 8$

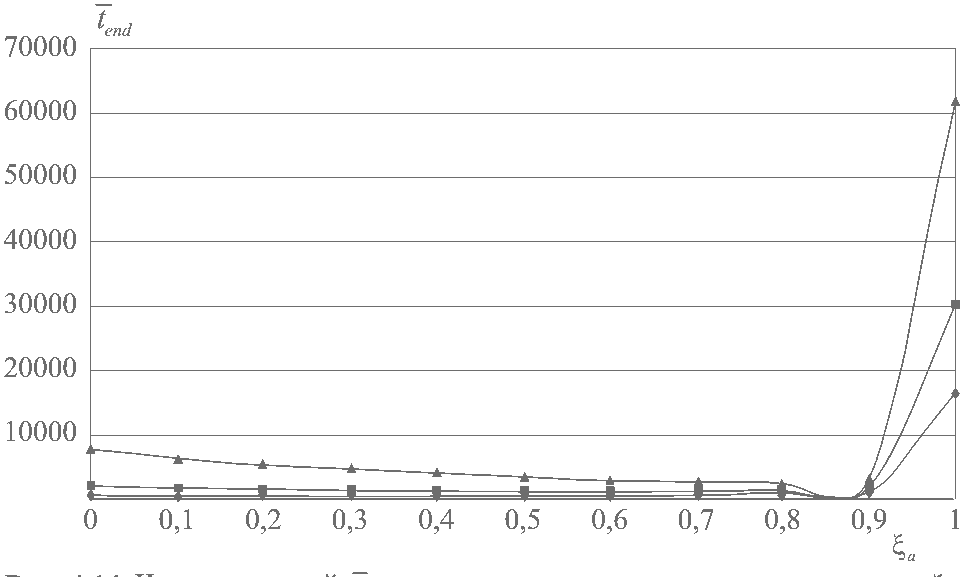


Рис. 4.14. Число итераций $\overline{t}_{end}$ в зависимости от величины вероятности ξ_a: функция Растригина; $|S| = 32$: ♦ – $|X| = 2$; ■ – $|X| = 4$; ▲ – $|X| = 8$

Для всех рассмотренных значений величины ξ_a оценка $\tilde{\xi}$ вероятности локализации глобального минимума сферической функции равна 100%. Для функции Растригина эта оценка отлична от 100% только при $\xi_a = 1$, что объясняется, опять же, высокими диверсификационными и низкими интенсификационными свойствами алгоритма в данном случае •

4.6. АЛГОРИТМЫ, ВДОХНОВЛЕННЫЕ ПОВЕДЕНИЕМ ОБЕЗЬЯН

Известны два существенно различных алгоритма поисковой оптимизации, вдохновленных некоторыми аспектами поведения обезьян в процессе поиска ими пищи, – алгоритм обезьяньего поиска и обезьяний алгоритм.

4.6.1. Обезьяний поиск

Алгоритм обезьяньего поиска (*Monkey Search*, *MS*) предложен Мучерино (*A. Mucherino*) и Шерефом (*O. Seref*) в 2008 г. Алгоритм вдохновлен поведением обезьяны, лазающей по дереву в поисках пищи. Обезьяне ставится в соответствие агент, который обследует деревья решений для поиска экстремума в задаче глобальной максимизации. Базовым предположением алгоритма *MS* является то, что обезьяна исследует дерево с целью узнать, какие ветви приводят к большему количеству пищи.

Таким образом, в алгоритме *MS* максимальное количество пищи представляет собой желаемое решение, а ветви дерева – варианты выбора между соседними допустимыми решениями. Этот выбор может быть как полностью случайным, так и основанным на известных алгоритмах решения задачи глобальной оптимизации. Закончив поиск на одном дереве, обезьяна может перейти к другим деревьям, так что, можно сказать, речь идет о поиске в джунглях потенциальных решений. Алгоритм использует бинарные деревья поиска, то есть от каждой данной ветки (кроме ветвей, образующих вершину дерева) отходят две другие ветви с решениями, располагающимися на их концах.

Канонический алгоритм *MS* не является популяционным, поскольку предполагает поиск с помощью только одной обезьяны. Известны «многообезьяньи» модификации канонического алгоритма, что и позволяет нам включить данный алгоритм в рассмотрение.

Если в текущий момент времени обезьяна находится на конце некоторой ветви, то далее она с равной вероятностью перемещается по левой или правой исходящим ветвям. В точке пространства поиска, соответствующей концу ветви, на которой находится обезьяна, вычисляем значение фитнес-функции. Если это решение лучше найденного ранее лучшего решения, то запоминаем его, и по рассмотренной схеме обезьяна продолжает движение вверх. Движение останавливаем при достижении обезьяной вершины дерева, определяемого его максимально допустимой высотой $l_{\max}$. Все посещенные обезьяной ветви дерева запоминаем.

Если не все пути в дереве исследованы, то всякий раз после достижения обезьяной вершины дерева она спускается до текущей лучшей точки и сно-

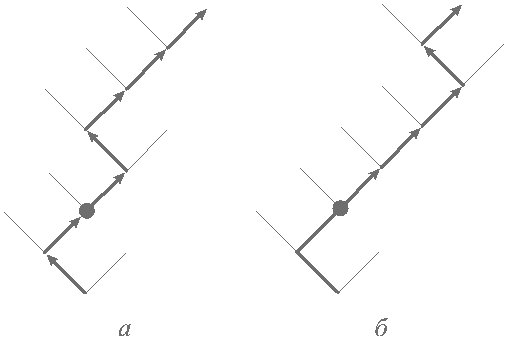


Рис. 4.15. Фазы поведения обезьяны на дереве поиска, темным кружком отмечено лучшее решение:
а – первый путь; *б* – второй путь

ва начинает движение вверх, возможно проходя некоторые из уже пройденных ветвей (рис. 4.15). После завершения исследования данного дерева может быть по той же схеме организован поиск на новом дереве.

Принципиальным в алгоритме *MS* является способ генерации ветвей дерева поиска, то есть генерации решений, соседствующих с текущим положением обезьяны. Эти решения получают путем локального возмущения текущего решения, для чего могут быть использованы миграционные операторы любых популяционных алгоритмов. Например, если X, X_* – текущее и лучшее положения обезьяны, то два следующих ее возможных положения X_1', X_2' могут быть найдены путем скрещивания решений X, X_*. Известны модификации алгоритма *MS*, использующие для генерации ветвей алгоритмы имитации отжига, поиска гармонии, колонии муравьев и т. д.

Для уменьшения вероятности преждевременной стагнации поиска может быть ограничено максимально допустимое число путей, подлежащих изучению в данном дереве. Если это число достигнуто, то лучшее полученное решение дерева запоминаем. Затем, исходя из *семени* (*seed*) – случайной точки пространства поиска – «выращиваем» новое дерево поиска.

После исследования по рассмотренной схеме заданного числа деревьев, в качестве семян новых деревьев могут быть использованы случайные решения из списка лучших решений. Поиск останавливаем, когда обследовано все предопределенное число деревьев.

Авторами алгоритма *MS* выполнено широкое исследование его эффективности при решении сложных тестовых задач глобальной оптимизации, возникающих при поиске стабильных атомных кластеров, взаимодействие атомов в которых подчинено закону Джонса (*L. Jones*) (Приложение Б). Результаты показали, что алгоритм *MS* является конкурентоспособным по сравнению с другими популяционными алгоритмами, используемыми для решения указанных задач.

4.6.2. Обезьяний алгоритм

Обезьяний алгоритм (*Monkey algorithm*) предложен Чжао (*R. Zhao*) и Тангом (*W. Tang*) в 2007 г. Алгоритм моделирует поведение обезьян в процессе их лазания по горам с целью поиска пищи. Полагают, что обезьяны исходят из того, что чем выше гора, тем больше пищи на ее вершине. Местность, которую обследуют обезьяны, представляет собой ландшафт фитнес-функции, так что решению соответствует самая высокая гора (рассматриваем задачу глобальной максимизации).

Из своего текущего положения каждая из обезьян движется вверх до тех пор, пока не достигнет вершины горы. Затем обезьяна делает серию локальных прыжков в случайном направлении в надежде найти более высокую гору, и движение вверх повторяется.

После выполнения некоторого числа подъемов и локальных прыжков обезьяна полагает, что в достаточной степени исследовала ландшафт в окрестности своего начального положения. Для того чтобы обследовать новую область пространства поиска, обезьяна выполняет длинный глобальный прыжок.

Указанные выше действия повторяются заданное число раз. Решением задачи объявляется самая высокая из вершин, найденных данной популяцией обезьян.

Инициализацию популяции обезьян s_i, $i \in [1:|S|]$ выполняют по общим правилам путем равномерного случайного распределения агентов в поисковом пространстве.

Процесс движения вверх (*climb process*) представляет собой процесс локального поиска и может быть реализован многими способами. В оригинальном алгоритме авторы предлагают использовать алгоритм на основе процедуры стохастической аппроксимации (п. 1.2.1).

Локальные прыжки (*watch-jump process*). Схема процесса локальных прыжков для агента s_i, $i \in [1:|S|]$ имеет следующий вид.

1) Исходя из текущего положения агента X_i, генерируем его новое возможное положение X_i' по формуле

$$x_{i,j}' = U_1\big((x_{i,j} - b); (x_{i,j} + b)\big),\ i \in [1:|S|],\ j \in [1:|X|],$$

то есть полагаем, что по каждой из координат новое положение агента представляет собой случайную величину, равномерно распределенную в интервале $\big[(x_{i,j} - b); (x_{i,j} + b)\big]$. Здесь $b > 0$ – максимально возможная длина прыжка (свободный параметр алгоритма).

2) Если точка X_i' является допустимой (принадлежит области допустимых значений D) и $\varphi(X_i') \geq \varphi(X_i)$, то полагаем $X_i = X_i'$ и завершаем для данного агента локальные прыжки, в противном случае заданное число раз возвращаемся к шагу 1.

Глобальные прыжки (*somersault process*) агенты выполняют из своего текущего положения в направлении текущего центра тяжести их координат по следующей схеме.

1) Генерируем случайное вещественное число $v = U_1(v^-; v^+)$, имеющее смысл величины шага глобального прыжка. Здесь v^-, v^+ – нижняя и верхняя границы этой величины, имеющие в общем случае разные знаки, так что величина v может быть как положительной, так и отрицательной.

2) Находим новое возможное положение X_i' агента s_i по формуле

$$x_{i,j}' = x_{i,j} + v\left(x_j^c - x_{i,j}\right),\ i \in [1:|S|],\ j \in [1:|X|],$$

где $x_j^c = \dfrac{1}{|S|}\sum_{i=1}^{|S|} x_{i,j}$ – текущее положение центра тяжести агентов популяции по j-му координатному направлению.

3) Если положение X_i' является допустимым, то полагаем $X_i = X_i'$ и завершаем для данного агента глобальные прыжки, иначе заданное число раз возвращаемся к шагу 1.

Отметим, что если величина шага глобального прыжка v принимает положительное значение, то агент совершает прыжок в сторону центра тяжести X^c,

а если отрицательное значение – в противоположную сторону. Для диверсификации поиска интервал $[v^-, v^+]$, очевидно, следует расширять, а для интенсификации – сужать.

В качестве критерия завершения итераций используем, как отмечалось выше, достижение заданного числа повторений указанных шагов алгоритма.

Поскольку *M*-алгоритм не гарантирует, что лучшее решение будет получено на последней итерации, каждый раз необходимо сохранять в памяти ЭВМ текущее лучшее решение.

Выполненное авторами *M*-алгоритма исследование его эффективности показало высокие значения скорости сходимости алгоритма и вероятности локализации глобального экстремума.

4.7. ПРОЧИЕ АЛГОРИТМЫ

К прочим алгоритмам относим тасующий алгоритм прыгающих лягушек, алгоритм, инспирированный летучими мышами, алгоритм поиска косяком рыб, а также алгоритм растущих деревьев. Известен также *алгоритм оптимизации мухами* (*Fly Optimization, FO*), который предложил Абидин (*Z. Z. Abidin*) с группой соавторов в 2009 г. В настоящее время алгоритм недостаточно разработан и по этой причине не нашел места в данной книге.

4.7.1. Тасующий алгоритм прыгающих лягушек

Тасующий алгоритм прыгающих лягушек (*Shuffled Frog-Leaping*, *SFL*) предложен Юсуфом (*M. Eusuff*) и другими авторами в 2003 г. По сути, алгоритм представляет собой гибридизацию меметического алгоритма и алгоритма роя частиц. Алгоритм вдохновлен поведением группы лягушек в процессе поиска пищи.

Рассматриваем задачу глобальной условной оптимизации, в которой фитнес-функция подлежит максимизации. Алгоритм *SFL* включает в себя следующие основные шаги.

1) Инициализируем популяцию $S = (s_i, i \in (1:|S|))$ из $|S|$ агентов путем равномерного распределения их в области поиска.

2) Вычисляем пригодности агентов популяции $\varphi(X_i) = \varphi_i$, $i \in (1:|S|)$ и определяем на этой основе глобально лучшего агента s_{i^*} такого, что

$$\max_{i \in [1:|S|]} \varphi_i = \varphi_{i^*}.$$

3) Разделяем агентов на $|S^p|$ наборов $S_j^p = (s_{j,k}, k \in [1:n])$, $j \in [1:|S^p|]$, названных авторами алгоритма *мемеплексами* (*memeplexes*), каждый из которых содержит по $n = \dfrac{|S|}{|S^p|}$ агентов, так что $\bigcap_{j=1}^{|S^p|} S_j^p = S$. Разделение выполняем по следующему правилу соседства: агента s_1 относим к мемеплексу S_1^p; агента s_2 – к мемеплексу S_2^p и так далее до мемеплекса $S_{|S^p|}^p$, к которому относим агента s_{S^p}; агента s_{S^p+1} относим к мемеплексу S_1^p, агента s_{S^p+2} – к мемеплексу S_2^p и так далее до последнего агента $s_{|S|}$, который должен быть отнесен к мемеплексу $S_{|S^p|}^p$.

4) Для каждого из мемеплексов выполняем заданное число раз *процедуру меметической эволюции* (*memetic evolution procedure*), которая реализует локальный поиск в пределах каждого из мемеплексов и состоит в следующем.

4.1) В мемеплексе S_j^p находим лучшего s_{j,k_b} и худшего s_{j,k_w} агентов:

$$\max_{k\in[1:n]} \varphi_{j,k} = \varphi(X_{j,k_b});\ \min_{k\in[1:n]} \varphi_{j,k} = \varphi(X_{j,k_w});\ j \in [1:|S^p|].$$

4.2) Пытаемся улучшить положение худшего агента путем случайного перемещения его в направлении к лучшему агенту или от лучшего агента по формуле

$$X'_{j,k_w} = X_{j,k_w} + U_1(-0,5;0,5)\, v_{\max} \frac{X_{j,k_b} - X_{j,k_w}}{\left\| X_{j,k_b}, X_{j,k_w} \right\|_E},\ j \in [1:|S^p|],$$

где $v_{\max}$ – максимально допустимая величина шага перемещения. Если $\varphi(X'_{j,k_w}) > \varphi(X_{j,k_w})$, то полагаем $X_{j,k_w} = X'_{j,k_w}$, то есть фиксируем удачное перемещение.

4.3) Если предыдущая операция не привела к успеху, то по правилу этой операции пытаемся улучшить положение того же агента путем перемещения его в направлении глобального лучшего агента s_{i^*}.

4.4) Если и последняя операция не привела к улучшению позиции агента s_{j,k_w}, то взамен этого агента случайным образом создаем в области поиска нового агента.

5) Выполняем *процедуру тасования* (*shuffling procedure*) агентов, которая заключается в объединении агентов всех мемеплексов в одну группу и случайном изменении их номеров. В результате при повторном выполнении шага 3 будут получены новые мемеплексы.

6) Проверяем выполнение условия завершения итераций и, если это условие выполнено, завершаем вычисления, полагая позицию текущего глобально лучшего агента решением задачи. В противном случае возвращаемся к шагу 3.

Таким образом, основой алгоритма *SFL* является комбинирование локального поиска в пределах каждого из мемеплексов и глобального поиска путем обмена информацией о положениях лучших агентов мемеплексов.

Известны модификации алгоритма *SFL*, в которых, например, на этапе локального поиска агент движется не точно в направлении лучшего агента соответствующего мемеплекса, а в некотором случайным образом возмущенном направлении. Известны как последовательные, так и параллельные гибридизации алгоритма со многими популяционными алгоритмами, например с алгоритмом искусственной иммунной системы.

4.7.2. Алгоритм, инспирированный летучими мышами

Алгоритм, инспирированный летучими мышами (*Bat-Inspired*, *BI*), предложен Янгом (*X.-Sh. Yang*) в 2010 г.

Летучие мыши – это удивительные животные. Из всего многообразия млекопитающих только летучие мыши имеют крылья. Кроме того, они обладают уникальными средствами эхолокации. Известно около 1000 видов летучих мышей (более 20 % видов всех млекопитающих), весящих от примерно двух граммов до одного килограмма (размах крыла последних достигает двух метров).

Способностью к эхолокации обладает большинство видов летучих мышей. Эхолокация используется мышами для обнаружения добычи и препятствий, а также для обеспечения возможности разместиться в темноте на насесте. Параметры лоцирующего звукового импульса мышей различных видов меняются в широких пределах, отражая их различные охотничьи стратегии. Большинство мышей используют короткие частотно-модулированные в пределах примерно одной октавы сигналы. В то же время некоторые виды не используют частотную модуляцию своих звуковых импульсов.

Длительность каждого из импульсов эхолокации составляет от 8 до 10 миллисекунд, частота ультразвуковых колебаний в импульсе может меняться в диапазоне от 25 до 150 кГц, так что длина волны этих колебаний равна 2–14 мм. Заметим, что такие величины длин волн соответствуют типичному размеру добычи мышей. В одну секунду мышь обычно испускает 10–20 импульсов, но при приближении к добыче число таких импульсов может быть увеличено до 200 в секунду. Громкость ультразвуковых колебаний меняется в широких пределах в зависимости от задачи, решаемой мышью, и может достигать 110 децибел.

Для того чтобы сформировать трехмерную модель окружения, при приеме отраженного сигнала мыши оперируют с различиями в громкости и времени прихода этого сигнала, воспринимаемого разными ушами. Для локализации движущихся объектов мыши используют эффект Доплера. Чувствительность последнего механизма такова, что мыши могут дифференцировать насекомых по частоте колебаний их крыльев. Кроме того, многие летучие мыши имеют хорошее зрение, а большинство из них – еще и прекрасные анализаторы запахов. В процессе охоты мыши используют комбинацию всех чувств, однако алгоритм *BI* моделирует лишь эффект эхолокации.

Алгоритм *BI* предполагает следующую модель поведения летучих мышей.

• С помощью эхолокации все мыши могут измерять расстояние до добычи и препятствий, а также различать их.

• Мыши движутся случайным образом. Текущие положение и скорость мыши s_i, $i \in [1:|S|]$ равны X_i, V_i соответственно. Для поиска добычи мыши генерируют сигналы с частотой ϖ_i и громкостью a_i. В процессе поиска мыши могут менять частоту этих сигналов, а также *частоту повторения* излучаемых импульсов (*rate of pulse*) $r \in [0;1]$.

• Частота сигналов может изменяться в диапазоне $[\varpi^{\min}; \varpi^{\max}]$, $\varpi^{\max} > \varpi^{\min} \geq 0$, а громкость сигналов – в пределах $[0;1]$.

Положим, что речь идет о задаче глобальной безусловной минимизации фитнес-функции. Основные шаги схемы алгоритма *BI* имеют следующий вид.

1) *Инициализация популяции*. Задаем начальные положения агентов s_i, $i \in [1:|S|]$. Определяем глобально лучшего агента и соответствующее ему решение X^{**}.

2) *Миграция агентов*. Выполняем перемещение всех агентов на один шаг в соответствии с используемой миграционной процедурой.

3) *Локальный поиск*. С вероятностью ξ_i^r (свободный параметр алгоритма) реализуем процедуру локального поиска в окрестности лучшего решения

X_i^*, найденного агентом s_i за все предшествующие итерации. Принимаем найденное решение в качестве нового текущего положения агента s_i; $i \in [1:|S|]$.

4) *Глобальный поиск*. В окрестности текущего решения X_i случайным образом генерируем решение X_i'. Если $\varphi(X_i') < \varphi(X^{**})$, то с вероятностью ξ_i^a принимаем решение X_i' в качестве нового текущего положения агента s_i. Находим новое глобально лучшее решение X^{**}.

5) *Эволюция параметров* ξ_i^r, ξ_i^a; $i \in [1:|S|]$.

6) *Окончание итераций*. Проверяем выполнение условий окончания итераций. Если эти условия выполнены, принимаем в качестве решения текущую точку X^{**} и завершаем вычисления, иначе возвращаемся к шагу 2.

Инициализацию популяции осуществляем путем случайного равномерного распределения агентов s_i, $i \in [1:|S|]$ в рассматриваемой области пространства поиска. На этом же этапе задаем начальные значения частот ϖ_i, громкостей a_i и частот повторения импульсов r_i, равномерно случайно распределяя их в соответствующих интервалах $[\varpi^{\min}; \varpi^{\max}]$, $[a^{\min}; a^{\max}]$, $[0; 1]$.

Миграция агентов. Миграцию агента s_i, $i \in [1:|S|]$ осуществляем по формулам:

$$X_i' = X_i + V_i',$$
$$V_i' = V_i + \varpi_i' \left(X_i - X^{**} \right), \tag{4.11}$$
$$\varpi_i' = \varpi^{\min} + (\varpi^{\max} - \varpi^{\min})\, U_1(0;1),$$

где глобально лучшее решение X^{**} определяет выражение

$$\max_{i \in [1:|S|]} \varphi(X_i) = \varphi(X^{**}) = \varphi^{**}.$$

Другими словами, в соответствии с миграционной процедурой агент перемещается в направлении, определяемом суммой вектора перемещения на предыдущей итерации (слагаемое V_i в формуле (4.11)) и случайным образом возмущенного вектора направления на лучшего агента $\left(X_i - X^{**} \right)$. Заметим, что данная процедура имеет много общего с аналогичной процедурой алгоритма роя частиц.

Локальный поиск выполняем по следующей схеме.

1) Случайным образом варьируем текущее положение агента s_i в соответствии с формулой

$$X_i' = X_i + \bar{a}\, U_{|X|}(-1;1),\ i \in [1:|S|],$$

где $\bar{a}$ – текущее среднее значение громкостей всех агентов популяции:

$$\bar{a} = \frac{1}{|S|} \sum_{i=1}^{|S|} a_i.$$

2) Вычисляем значение фитнес-функции в новой точке $\varphi(X_i') = \varphi_i'$. Если $\varphi_i' > \varphi_i$, то завершаем процедуру локального поиска, в противном случае фиксированное число раз возвращаемся к шагу 1.

Эволюцию параметров ξ_i^a, ξ_i^r производим по правилам:

$$\xi_i^a(t+1) = b_a\ \xi_i^a(t),\ \ \xi_i^r(t+1) = \xi_i^r(0)\left(1-\exp(-b_r\ t)\right),\ \ i \in [1:|S|],$$

где $b_a \in (0;1)$, $b_r > 0$ – заданные константы (свободные параметры алгоритма), рекомендуемые значения которых равны 0,9. Таким образом, имеют место предельные соотношения

$$\xi_i^a(t) \underset{t\to\infty}{\to} 0,\ \xi_i^r(t) \underset{t\to\infty}{\to} \xi_i^r(0).$$

Авторами алгоритма выполнено широкое экспериментальное сравнение эффективности алгоритма *BI* с генетическим алгоритмом и алгоритмом роя частиц. Показано, что с точки зрения вероятности локализации глобального экстремума алгоритм *BI* обладает большей эффективностью, чем указанные алгоритмы.

4.7.3. Поиск косяком рыб

Алгоритм поиска косяком рыб (*Fish School Search*, *FSS*) предложили в 2008 г. Фило (*B. Filho*) и Нето (*L. Neto*).

Известно более 25 000 видов рыб, что больше общего числа всех видов позвоночных. Примерно 50–80 % рыб плавают в косяках. *Косяком рыб* (*fish school*) называют агрегацию рыб, которые плавают приблизительно с одной и той же скоростью и ориентацией, поддерживая примерно постоянное расстояние между собой. Обычно косяк содержит рыб, размеры которых отличаются не более чем на 30 %. Доказано, что всякого рода объединения рыб играют важную роль в повышении эффективности поиска ими пищи, защиты от хищников, а также в уменьшении энергетических затрат в процессе плавания.

В алгоритме *FSS* рыбы плавают в аквариуме (области поиска) в поисках пищи (решения задачи оптимизации). Вес каждой рыбы формализует ее индивидуальный успех в поиске решения и играет роль памяти рыбы. Именно наличие веса у агентов популяции является главной особенностью парадигмы *FSS* в сравнении с парадигмой оптимизации роем частиц. Эта особенность алгоритма *FSS* позволяет отказаться от необходимости отыскивать и фиксировать глобально лучшие решения, как это делается в алгоритме роя частиц.

Полагаем, что решается задача глобальной условной максимизации в области D и что всюду в этой области фитнес-функция принимает неотрицательные значения.

Операторы алгоритма *FSS* объединены в две группы:

– оператор кормления формализует успешность исследования агентами тех или иных областей аквариума;

– операторы плавания реализуют алгоритмы миграции агентов косяка рыб.

Оператор кормления (*feeding*). Обозначим w_i текущий вес агента s_i. В алгоритме *FSS* принято, что приращение веса агента на данной итерации t пропорционально нормализованной разности значений фитнес-функции на итерациях t, $(t-1)$, то есть

$$w_i' = w_i + \frac{\varphi_i - \varphi_i(t-1)}{\max\left(\varphi_i, \varphi_i(t-1)\right)},\ i \in [1:|S|].$$

Алгоритм ограничивает максимально допустимый вес агентов величиной $w_{\max} > 0$, так что во всех случаях вес агента s_i удовлетворяет условию $w_i \in [1; w_{\max}]$, где $w_{\max}$ – свободный параметр алгоритма. При инициализации популяции всем агентам в качестве веса присваиваем значение $w_{\max}/2$.

Операторы плавания (*swimming*). В алгоритме *FSS* различают три вида плавания – индивидуальное, инстинктивно-коллективное и коллективно-волевое. Положим, что эти виды плаваний выполняются на интервалах $(t; \tau]$, $(\tau; \theta]$, $(\theta; t']$ основного итерационного интервала $(t;\ t']$ соответственно. Здесь $t < \tau < \theta < t'$, $t' = t + 1$.

Индивидуальное (individual) плавание. Направление перемещения агента в этом случае равновероятно случайно. Если перемещение агента в выбранном направлении выводит агента за пределы области допустимых значений D, то перемещение не производим. Аналогично, перемещение агента s_i не производится, если имеет место неравенство $\varphi_i^{\tau} < \varphi_i^{t}$, то есть если в новой точке X_i^{τ} значение фитнес-функции не выше ее значения в предыдущей точке X_i^{t}.

Величину шага перемещения v_i^{ind} полагаем случайной величиной, равномерно распределенной в интервале $[0; v_{\max}^{ind}]$:

$$v_i^{ind} = U_1(0;1)\, v_{\max}^{ind},\ i \in [1:|S|].$$

Для постепенного перехода от диверсификации поиска на начальных итерациях к его интенсификации на завершающих итерациях линейно уменьшаем величину $v_{\max}^{ind}$ (свободный параметр алгоритма) с ростом числа итераций.

Индивидуальное плавание агента может включать в себя не одну итерацию, как в рассмотренной схеме, а их некоторое фиксированное число. Таким образом, это плавание можно интерпретировать как локальный поиск в окрестности текущего положения агента.

Инстинктивно-коллективное (*collective-instinct*) *плавание* реализуем после завершения всеми $|S|$ агентами индивидуальных плаваний по формуле

$$X_i^{\theta} = X_i^{\tau} + \frac{\sum\limits_{j \in [1:|S|]} V_j^{ind}(\tau)\left(\varphi_i^{\tau} - \varphi_i^{t}\right)}{\sum\limits_{j \in [1:|S|]} \left(\varphi_j^{\tau} - \varphi_j^{t}\right)}, \tag{4.12}$$

где $V_j^{ind}(\tau) = X_j^{\tau} - X_j^{t}$ – перемещение агента s_j в процессе его индивидуального плавания. Второе слагаемое в формуле (4.12) есть не что иное, как общий для всех агентов шаг миграции, представляющий собой взвешенную сумму индивидуальных перемещений агентов. Формула (4.12) означает, что в процессе инстинктивно-коллективного плавания на каждого из агентов оказывают влияние все остальные агенты популяции и это влияние пропорционально индивидуальным успехам агентов.

Коллективно-волевое (*collective volition*) *плавание* выполняем вслед за инстинктивно-коллективным плаванием. Данный вид плавания заключается в смещении всех агентов в направлении текущего центра тяжести популяции,

если суммарный вес косяка в результате индивидуального и инстинктивно-коллективного плавания увеличился, и в противоположном направлении – если этот вес уменьшился. Другими словами, в случае успеха указанных плаваний популяция стягивается к своему центру тяжести, то есть повышает интенсивность поиска. В противном случае популяция расширяется от того же центра, повышая свои диверсификационные свойства.

Координаты центра тяжести X_c^θ косяка после завершения всеми его агентами инстинктивно-коллективных плаваний определяем по формуле

$$X_c^\theta = \frac{\sum\limits_{i\in[1:|S|]} w_i^\theta X_i^\theta}{w_\Sigma^\theta},$$

где $w_\Sigma^\theta = \sum\limits_{i=1}^{|S|} w_i^\theta$ – текущий суммарный вес популяции.

Коллективно-волевое плавание выполняем по правилу

$$X_i' = X_i^\theta \pm v^{vol}\left(X_i^\theta - X_c^\theta\right),\ i\in[1:|S|], \tag{4.13}$$

где знак плюс используем в случае $w_\Sigma^\theta > w_\Sigma^{\theta-1}$, а знак минус – в противном случае. Здесь $w_\Sigma^{\theta-1}$ – суммарный вес популяции после завершения ее агентами инстинктивно-коллективных плаваний на предыдущей итерации. Переменная v^{vol} в формуле (4.13) определяет размер шага перемещений агентов и является случайной величиной

$$v^{vol} = v_{\max}^{vol}\, U_1(0;1).$$

Здесь $v_{\max}^{vol} > 0$ – заданная максимально допустимая длина шага (свободный параметр алгоритма). Авторы алгоритма рекомендуют линейное уменьшение величины $v_{\max}^{vol}$ с ростом числа итераций.

Инициализацию популяции осуществляем путем случайного равномерного распределения агентов в области поиска.

В качестве *условия завершения итераций* канонический алгоритм *FSS* использует условие достижения априори заданного числа итераций $\hat{t}$.

Выполненное авторами алгоритма *FSS* экспериментальное исследование его эффективности показывает преимущества алгоритма *FSS* по сравнению с каноническим алгоритмом роя частиц и некоторыми его модификациями по критерию точности локализации глобального экстремума. Исследование проведено для задачи безусловной оптимизации и тестовых целевых функций Розенброка, Растригина, Гриванка, Экли и Шефеля (см. Приложение Б).

Известен ряд модифицированных алгоритмов *FSS*, например, *плотный алгоритм FSS* (*Density FSS, DFSS*), использующий модифицированные операторы классического алгоритма, а также новые операторы *памяти* и *разбиения*.

Оператор памяти (*memory*) агента s_i строится на основе $(|S|\times 1)$-вектора M_i, формализующего текущее и ряд предшествующих влияний на этого агента других агентов популяции. *Оператор разбиения* (*partitioning*), используя память агентов $M_i, i\in[1:|S|]$, формирует на основе популяции S ряд субпопуляций.

Известны также высокоэффективные гибридные алгоритмы, построенные на основе алгоритма *FSS*. Примером такой гибридизации могут служить *волевой алгоритм роя частиц* (*Volitive PSO, VPSO*).

4.7.4. Алгоритм растущих деревьев

Алгоритм растущих деревьев (*Saplings Sowing and Growing up, SSG*) предложен Карчи (*A. Karci*) с соавторами в 2002 г. Алгоритм вдохновлен эволюцией растущих деревьев и включает в себя два этапа – *этап посадки* (*sowing phase*) и *этап роста* (*growing up phase*).

Рассматриваем задачу глобальной условной оптимизации в случае, когда область допустимых значений D (сад) представляет собой гиперпараллелепипед Π, нижняя и верхняя границы которого по k-му измерению равны x_k^-, x_k^+ соответственно; $k \in [1:|X|]$.

На этапе посадки саженцы (начальные решения) случайным образом равномерно располагаем в области поиска – создаем *равномерный сад (uniform garden)*.

Этапы роста реализуем с помощью трех операторов – *оператор скрещивания (mating operator), оператор ветвления (branching operator)* и *оператор прививки (vaccinating operator)*.

Оператор скрещивания. Пусть текущие координаты агентов $s_i, s_j, i, j \in [1:|S|], i \neq j$ определяют векторы решений X_i, X_j. Скрещивание агентов s_i, s_j заключается в обмене между этими векторами некоторым числом их компонент. Вероятность ξ_m скрещивания агентов s_i, s_j определяем на основе нормированного евклидова расстояния между ними:

$$\xi_m(s_i, s_j) = 1 - \frac{\|X_i, X_j\|_E}{r}.$$

Здесь r – длина диагонали параллелепипеда Π, т. е.

$$r = \left(\sum_{k=1}^{|X|} (x_k^+ - x_k^-)^2 \right)^{\frac{1}{2}}.$$

Оператор ветвления. Рассмотрим агента $s_i, i \in [1:|S|]$, и соответствующее текущее решение X_i. Положим, что компонента $x_{i,k}, k \in [1:|X|]$, этого решения была изменена (в k-й точке имело место ветвление). Тогда вероятность ξ_b того, что будет изменена компонента $x_{i,l}$, $l \in [1:|X|]$, $l \neq k$ того же вектора (в l-й точке будет произведено ветвление) может быть вычислена по *линейной формуле*

$$\xi_b(x_{i,l} \mid x_{i,k}) = 1 - \frac{1}{(l-k)^2}$$

или *нелинейной формуле*

$$\xi_b(x_{i,l} \mid x_{i,k}) = 1 - \frac{1}{\exp(l-k)^2}.$$

Оператор прививки применяют к агентам $s_i, s_j, i, j \in [1:|S|], i \neq j$, в том случае, когда мера $\mu(s_i, s_j)$ их близости (*similarity*) превышает порог $\varepsilon|X|$, где ε – заданная положительная константа (свободный параметр алгоритма). В качестве меры $\mu(s_i, s_j)$ в оригинальном алгоритме *SSG* использована величина

$$\mu(s_i, s_j) = \sum_{k=1}^{|X|} \frac{|x_{i,k} - x_{j,k}|}{x_k^+ - x_k^-}.$$

Прививку выполняем по правилу

$$x'_{i,k} = \begin{cases} x_{i,k}, & \dfrac{|x_{i,k} - x_{j,k}|}{x_k^+ - x_k^-} \le \varepsilon|X|, \\ x_{j,k}, & \text{иначе} \end{cases} \quad x'_{j,k} = \begin{cases} x_{j,k}, & \dfrac{|x_{i,k} - x_{j,k}|}{x_k^+ - x_k^-} \le \varepsilon|X|, \\ x_{i,k}, & \text{иначе.} \end{cases}$$

Здесь $k \in [1:|X|]$. Другими словами, если модуль относительной разности между k-ми компонентами векторов X_i, X_j превосходит величину $\varepsilon|X|$, то эти вектора обмениваются k-ми компонентами.

После применения к текущей популяции указанных операторов получаем промежуточную популяцию $S \cup S'$, которой соответствует объединение исходных X_i и модифицированных векторов X'_i; $i \in [1:|S|]$. После вычисления значений фитнес-функции во всех указанных точках из промежуточной популяции отбираем $|S|$ лучших агентов в популяцию S^{t+1}. Итерационный процесс повторяется до достижения заданного числа итераций.

Автором алгоритма *SSG* выполнено его исследование на ряде тестовых задач многомерной глобальной оптимизации, показавшее перспективность развития алгоритма.

Вопросы для самопроверки

1. Каковы биологические предпосылки искусственных иммунных систем?
2. Приведите общую схему решения задачи оптимизации с помощью модели иммунной сети.
3. Представьте схему алгоритма оптимизации на основе искусственной микроиммунной системы.
4. Сформулируйте биологические предпосылки бактериальной оптимизации.
5. Представьте схему канонического алгоритма бактериальной оптимизации.
6. Назовите принципы кооперативной бактериальной оптимизации и алгоритма, моделирующего эффект роения бактерий.
7. Приведите возможные схемы гибридизации бактериального алгоритма с другими популяционными алгоритмами.
8. Сформулируйте идеи, лежащие в основе алгоритма светлячков. Приведите схему этого алгоритма.
9. Назовите отличительные особенности алгоритма оптимизации роем светлячков.
10. Определите основные операторы алгоритма сорняковой оптимизации.
11. Представьте схему канонического алгоритма кукушкиного поиска, а также схему модифицированного алгоритма поиска кукушки.
12. Приведите схемы алгоритма обезьяньего поиска и обезьяньего алгоритма.
13. Назовите основные особенности тасующего алгоритма прыгающих лягушек, алгоритма, инспирированного летучими мышами, алгоритма поиска косяком рыб, алгоритма растущих деревьев.

Глава 5

ПОПУЛЯЦИОННЫЕ АЛГОРИТМЫ, ИНСПИРИРОВАННЫЕ НЕЖИВОЙ ПРИРОДОЙ, ЧЕЛОВЕЧЕСКИМ ОБЩЕСТВОМ, И ДРУГИЕ ПОПУЛЯЦИОННЫЕ АЛГОРИТМЫ

Материал главы можно разделить на три части. В первой части (параграфы 5.1–5.3) рассматриваем популяционные алгоритмы поисковой оптимизации, которые основаны на подражании неживой природе, – это алгоритмы поиска гармонии, гравитационного и электромагнитного поиска. Вторая часть (параграфы 5.4–5.8) включает в себя описание алгоритмов, инспирированных человеческим обществом, – алгоритмы эволюции разума и стохастического диффузионного поиска, культурный и меметические алгоритмы, а также самоорганизующийся миграционный алгоритм. Последняя часть (параграф 5.9) представлена алгоритмами рассеянного поиска и прокладки путей, которые трудно отнести к перечисленным выше классам алгоритмов.

Мы не рассматриваем широко известный *алгоритм интеллектуальных капель воды* (*Intelligent Water Drop algorithm, IWD*), предложенный Шах-Хосейни (*H. Shah-Hosseini*) в 2009 г. Алгоритм *IWD* вдохновлен способностью рек прокладывать близкие к оптимальным пути своего течения и предназначен для решения задач на графах, таких как задача о коммивояжере (*Travelling Salesman Problem, TSP*) или задача экономичного распределения нагрузок (*Economic Load Dispatch Problem, ELDP*). По той же причине в главе не нашел места алгоритм *RFD* (*River Formation Dynamics*), предложенный Рабаналом (*P. Rabanal*) с соавторами в 2007 г.

5.1. ГАРМОНИЧЕСКИЙ ПОИСК

Алгоритм поиска гармонии (*Harmony Search algorithm*, *HS*) предложен в 2001 г. Гимом (*Z. W. Geem*). Алгоритм вдохновлен процессом поиска музыкантами гармонии в музыке. Ситуации идеальной гармонии звуков алгоритм *HS* ставит в соответствие глобальный экстремум в задаче многомерной оптимизации, а процессу импровизации музыканта – процедуру поиска этого экстремума.

В процессе импровизации каждый музыкант на каком-либо такте музыкального произведения воспроизводит звук (в рамках возможностей его музыкального инструмента), так что звуки всех музыкантов оркестра на этом такте образуют один вектор гармонии. Векторы, образующие «хорошие» гармонии, хранятся в памяти каждого из музыкантов и могут быть ими использованы для формирования «хороших» гармоний на следующих тактах произведения. Как правило, в процессе импровизации музыкант выполняет одно из трех следующих требований: сформировать абсолютно случайный звук из доступного ему диапазона звуков; воспроизвести какой-либо звук из своей памяти гармоний; сыграть смежный вектор гармонии из той же памяти.

Основными особенностями алгоритма *HS* являются его простота и возможность использования для решения задач как непрерывной, так и дискретной оптимизации. В силу этого алгоритм привлекает значительное внимание исследователей и быстро развивается как в теоретическом, так и в практическом плане.

5.1.1. Канонический алгоритм

Рассмотрим задачу глобальной условной минимизации в гиперпараллелепипеде Π. Полагаем, что фитнес-функция также подлежит минимизации. Музыкантам ставим в соответствие индивидов s_i, а оркестру – популяцию $S = \{s_i, i \in [1:|S|]\}$. Аккорду, который берет музыкант s_i в данный момент времени, сопоставляем значение вектора варьируемых параметров X_i. Гармонию звуков формализует глобальный минимум фитнес-функции $\varphi(X)$. Совокупность текущих координат X_i, $i \in [1:|S|]$ образует $(|S| \times |X|)$-*матрицу памяти гармоний* (*harmony memory*) H^M.

Схема канонического алгоритма *HS* включает в себя следующие основные шаги.

1) Инициализируем алгоритм.
2) Формируем вектор гармонии.
3) Выполняем пошаговую настройку вектора гармонии.
4) Обновляем матрицу памяти гармоний.
5) Завершаем итерации, если условие окончания итераций выполнено, возвращаемся к шагу 2 в противном случае.

Инициализация. Начальные значения компонент вектора X_i полагаем равномерно распределенными в гиперпараллелепипеде Π:

$$x_{i,j} = x_j^- + U_1(0;1)\left(x_j^+ - x_j^-\right),\ i \in [1:|S|],\ j \in [1:|X|]. \tag{5.1}$$

Совокупность так построенных векторов X_i образует исходную матрицу памяти гармоний

$$H^M = \begin{pmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,|X|} \\ x_{2,1} & x_{2,2} & \dots & x_{2,|X|} \\ \dots & \dots & \dots & \dots \\ x_{|S|,1} & x_{|S|,2} & \dots & x_{|S|,|X|} \end{pmatrix}.$$

Формирование вектора гармонии X' выполняем по следующим правилам.

С вероятностью ξ_h в качестве компоненты $x'_{i,j}$ вектора X'_i используем соответствующую компоненту случайного вектора X_{i_1} из текущей матрицы памяти гармоний H^M:

$$x'_{i,j} = H^M_{i_1,j},\ i_1 = U_1(1:|S|),\ j \in [1:|X|].$$

Данная операция моделирует воспроизведение музыкантом какого-либо аккорда из его памяти гармоний и называется *операцией случайного выбора* (*random selection*).

С вероятностью $(1-\xi_h)$ в качестве компоненты $x'_{i,j}$ берем величину, сгенерированную по формуле вида (5.1). Операция адекватна ситуации формирования абсолютно случайного аккорда из доступного музыканту диапазона.

Свободный параметр алгоритма ξ_h имеет смысл относительной частоты операции случайного выбора, то есть *вероятности использования памяти гармоний* (*Harmony Memory Considering Rate, HMCR*).

Пошаговая настройка вектора гармонии. Если компонента $x'_{i,j}$ вектора X'_i выбрана из памяти гармоний, то выполняем следующие действия.

С вероятностью ξ_p изменяем эту компоненту по формуле

$$x'_{i,j} = x'_{i,j} + u_{sign}^{\pm 1} b_w N_1(0;1) \tag{5.2}$$

или с вероятностью $(1-\xi_p)$ оставляем без изменений. Формула означает, что компонента $x'_{i,j}$ с равной вероятностью получает случайное положительное или отрицательное приращение величиной $b_w N_1(0;1)$. Параметр ξ_p определяет относительную частоту пошаговой настройки, то есть *вероятность изменения шага* (*Pitch Adjusting Rate, PAR*), а параметр b_w представляет собой величину шага, которому придается смысл *полосы пропускания инструмента* (*pitch BandWidth Rate, BWR*).

Обновление матрицы памяти гармоний. Вычисляем значение фитнес-функции $\varphi(X'_i)$, соответствующее сформированному вектору X'_i. Если $\varphi(X'_i) < \varphi(X_{i_w})$, то худший вектор X_{i_w} памяти гармоний заменяем вектором X'_i. Вектор X_{i_w} находим из условия

$$\max_{i\in[1:|S|]} \varphi(X_i) = \varphi(X_{i_w}).$$

Завершение итераций. Если условие окончания итераций выполнено, то в качестве решения задачи используем лучший вектор памяти гармоний X_{i_b}, доставляющий наименьшее значение функции приспособленности:

$$\min_{i\in[1:|S|]} \varphi(X_i) = \varphi(X_{i_b}).$$

Рекомендуют следующие значения основных свободных параметров алгоритма *HS*: $|S| \le 50$; $\xi_h = 0,7 \div 0,95$; $\xi_p = 0,1 \div 0,5$. Два последних параметра используют для регулирования соотношения диверсификационных и интенсификационных свойств алгоритма – меньшие значения этих параметров диверсифицируют поиск (за счет, конечно, уменьшения скорости его сходимости).

Наиболее существенное влияние на эффективность алгоритма *HS* оказывает свободный параметр b_w, который, вообще говоря, может принимать значения в диапазоне $(x_j^+ - x_j^-)$. Априорные рекомендации по выбору значений этого параметра затруднительны. Известны, например, следующие правила выбора:

$$b_w = b_w^0\, U_1(0;1);$$

$$b_w(t) = \begin{cases} b_w^v, & n_{ch}(t) \le n_{ch}^{\max}, \\ \dfrac{b_w^v}{2}, & n_{ch}(t) > n_{ch}^{\max}; \end{cases}$$

$$b_w(t) = \begin{cases} b_w^v\, U_1(0;1), & n_{ch}(t) \le n_{ch}^{\max}, \\ \dfrac{b_w^v}{2}\, U_1(0;1), & n_{ch}(t) > n_{ch}^{\max}; \end{cases}$$

$$b_w(t) = b_w^- + \frac{b_w^+ - b_w^-}{\hat{t}-1}(t-1). \tag{5.3}$$

Здесь b_w^0 – заданная величина шага; b_w^-, b_w^+ – его начальное и конечное значения; b_w^v – так называемое варьируемое значение *шага пропускной способности*; $n_{ch}(t)$, n_{ch}^{max} – текущее и максимально допустимое числа обновлений памяти гармоний соответственно. Отметим, что последние три формулы определяют динамический параметр b_w.

5.1.2. Некоторые модификации алгоритма гармонического поиска

Для повышения эффективности канонического алгоритма *HS* разработано большое число его модификаций. В известном обзоре Инграма (*G. Ingram*) и Чжана (*T. Zhang*), посвященном гармоническому поиску, выделены следующие направления его модификации.

• Использование альтернативных правил инициализации популяции.

Так, известна модификация алгоритма *HS*, которая предполагает генерацию $2|S|$ начальных векторов и выбор из них $|S|$ лучших.

• Различные динамические стратегии изменения значений свободных параметров алгоритма.

К примеру, в так называемом *улучшенном алгоритме гармонического поиска* (*Improved HS*, *IHS*), предложенном Махдави (*M. Mahdavi*) и другими авторами, используется динамическое обновление величин b_w, ξ_p по формулам

$$b_w(t) = b_w^+ \exp\left(\ln\left(\frac{b_w^-}{b_w^+}\right)\frac{t}{\hat{t}}\right),\ \xi_p(t) = \xi_p^- + \left(\xi_p^+ - \xi_p^-\right)\frac{t}{\hat{t}},$$

где b_w^-, b_w^+, ξ_p^-, ξ_p^+ – начальные и конечные значения этих величин соответственно.

Недостатком такого сорта модификаций является появление, как правило, новых свободных параметров (в рассмотренном примере – параметров b_w^-, b_w^+, ξ_p^-, ξ_p^+), оптимальные значения которых в значительной мере зависят от свойств решаемой задачи оптимизации.

• Новые или модифицированные операторы канонического гармонического поиска.

Например, в модификации алгоритма *HS*, авторами которой являются Коэлхо (*L. d.S. Coelho*) и Бернерт (*D. L. d. A. Bernert*), при определении вектора X_i' вместо формулы (5.2) используют формулу вида

$$x_{i,j}' = x_{i,j}' + u_{sign}^{\pm 1}\, b_w\, E_1(a,b),$$

где $E_1(a,b)$ – вещественное случайное число, распределенное по экспоненциальному закону с параметрами a, b. Рекомендованные авторами алгоритма значения этих величин равны 0,3; 0,1 соответственно.

Дифференциальный алгоритм гармонического поиска (*Differential HS algorithm*, *DHS*), разработанный Чакраборти (*P. Chakraborty*) и другими авторами, предполагает вычисление компонент вектора X_i' на основе оператора дифференциальной эволюции

$$x_{i,j}' = x_{i,j}' + a\left(x_{i_1,j} - x_{i_2,j}\right),\ j \in [1:|X|],\ i_1, i_2 \in [1:|S|],\ i_1 \neq i_2,$$

где a – скалярный свободный параметр.

Глобально лучший HS-алгоритм (*Global HS, GHS*), предложенный Махдави (*M. Mahdavi*) и Омраном (*M. Omran*), использует при определении вектора X_i' элементы алгоритма роя частиц. Вектор X_i' выбирают в данной модификации таким образом, чтобы он в некотором смысле был близок к текущему лучшему вектору памяти гармоний X_{i_b}. Тем самым в алгоритм гармонического поиска добавляется социальный аспект.

Упомянем также алгоритм гармонического поиска, использующий неравномерный генетический оператор мутации.

• Различные варианты учета ограничений при формировании вектора X_i'.

Возможно, например, использование алгоритма на основе скользящего допуска (см. 1.3.1) по следующей схеме. После формирования тем или иным способом вектора X_i' проверяем степень нарушения им ограничения $X_i' \in D$. Если степень нарушения ограничений превышает текущее значение критерия скользящего допуска $\delta = \delta(t)$, то от этого вектора отказываемся. В противном случае вектор X_i' по общему правилу может быть включен в матрицу памяти гармоний. Напомним, что величина $\delta(t)$ с ростом номера итерации t монотонно убывает.

• Различные критерии включения нового вектора X' в память гармоний.

Например, таким критерием может быть выполнение не условия $\varphi(X_i') < \varphi(X_{i_w})$, как в каноническом алгоритме *HS*, а условия $\varphi(X_i') < \overline{\varphi}$, где $\overline{\varphi}$ – средняя пригодность гармоний в текущей матрице H^M. Известны критерии, построенные на основе близости в некоторой метрике вектора X_i' и соответствующего вектора X_i из матрицы H^M.

• Изменение условий окончания итераций.

• Изменение структуры алгоритма.

Эти изменения могут быть небольшими, как, например, многократные гармонии, предложенные Ли (*L. Li*) с соавторами. Изменения могут быть также кардинальными, основанными на низкоуровневой гибридизации вложением алгоритма *HS* с другими популяционными и не популяционными алгоритмами (гл. 6). В настоящее время известны такие гибридизации алгоритма *HS* с одним, двумя и тремя алгоритмами. Известны, например, гибридизации алгоритма *HS* с генетическим алгоритмом; генетическим алгоритмом и алгоритмом имитации отжига; алгоритмом роя частиц; алгоритмом дифференциальной эволюции.

5.1.3. Пример применения модифицированного алгоритма гармонического поиска

Рассмотриваем задачу оптимизации конструкции, состоящей из балки *A* длиной $L = 14$ дюймов и бруска *B*, соединенных сварным швом (рис. 5.1). Задача представляет собой известную практически значимую тестовую задачу для алгоритмов условной оптимизации (Приложение Б) и состоит в том, чтобы при заданной нагрузке P = 6000 фунтов найти допустимые размеры конструкции, минимизирующие общую стоимость ее изготовления $f(X)$, где вектор варьируемых параметров $X = (b, h, l, s)$.

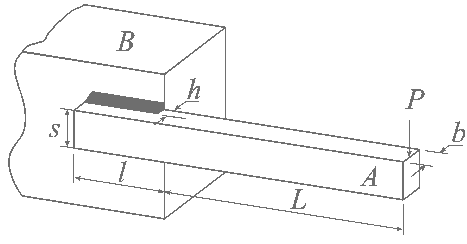


Рис. 5.1. Оптимизируемая конструкция

Целевая и ограничивающие функции имеют следующий вид:

$$f(X) = 1{,}10471\, h^2 l + 0{,}04811\, sb(L+l);$$
$$g_1(X) = \tau_d - \tau(X) \geq 0;$$
$$g_2(X) = \sigma_d - \sigma(X) \geq 0;$$
$$g_3(X) = b - h \geq 0;$$
$$g_4(X) = P_c(X) - P \geq 0;$$
$$g_5(X) = 0{,}25 - \delta(X) \geq 0;$$
$$0{,}12 \leq h \leq 5{,}0;\ 0{,}1 \leq l, s \leq 10{,}0;\ 0{,}1 \leq b \leq 5{,}0.$$

Здесь принято, что $\tau_d = 13600$ фунтов на кв. дюйм – допустимое напряжение сварного шва;

$$\tau(X) = \sqrt{(\tau'(X))^2 + (\tau''(X))^2 + \frac{l\tau'(X)\tau''(X)}{\sqrt{0{,}25\,(l^2 + (h+s)^2)}}} -$$

касательное напряжение в сварном шве; $\sigma_d = 30600$ фунтов на кв. дюйм – допустимый предел текучести материала конструкции; $\sigma(X) = \dfrac{504000}{s^2 b}$ – максимальное напряжение изгиба;

$$P_c(X) = 64746\,(1 - 0{,}0282346\, s) sb^2 -$$

критическая нагрузка; $\delta(X) = 2{,}1952 / s^2 b$ – прогиб конца балки.

Величины $\tau'(X)$, $\tau''(X)$ определяют выражения

$$\tau'(X) = \frac{6000}{l\, h\sqrt{2}},\ \tau''(X) = \frac{6000\,(14 + 0{,}5\, l)\sqrt{0{,}25\,(l^2 + (h+s)^2)}}{2\left(0{,}707\, h\, l\left(\dfrac{l^2}{12} + 0{,}25\,(h+s)^2\right)\right)}.$$

Используем канонический алгоритм *HS* со следующими базовыми значениями свободных параметров: $\xi_h = 0{,}7$; $\xi_p = 0{,}3$; $b_w = 0{,}2$. В качестве условия

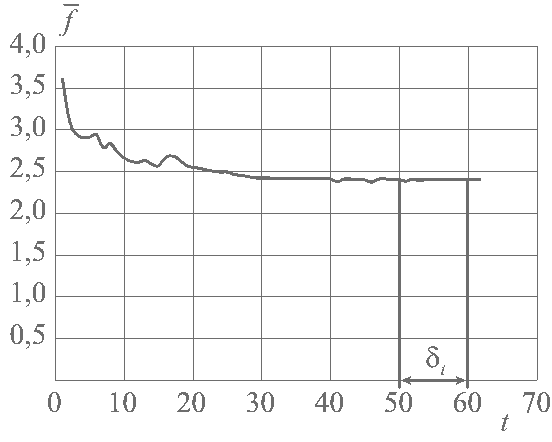


Рис. 5.2. Характер сходимости итерационного процесса

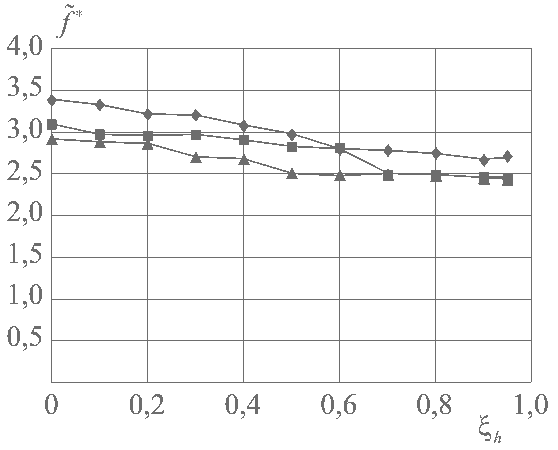


Рис. 5.3. Лучшие достигнутые значения целевой функции в зависимости от вероятности использования памяти гармоний, $b_w = 0,2$: ♦ – $\xi_p = 0,3$; ■ – $\xi_p = 0,6$; ▲ – $\xi_p = 0,9$

окончания итераций используем стагнацию вычислительного процесса в течение $\delta_t = 10$ итераций.

Характер сходимости алгоритма *HS* иллюстрирует рис. 5.2, на котором представлены средние значения целевой функции $\bar{f}$ по $\hat{n} = 30$ стартам алгоритма.

На рис. 5.3 показаны лучшие достигнутые значения целевой функции $\tilde{f}^*$ в зависимости от вероятности использования памяти гармоний ξ_h при различных вероятностях изменения шага ξ_p. Из рисунка следует, что путем повышения вероятности ξ_h удается почти на 30% улучшить значение $\tilde{f}^*$. За счет варьирования значений параметра ξ_p можно улучшить это значение еще примерно на 15%.

На рис. 5.4 аналогично рис. 5.3 проиллюстрирован характер зависимости значений целевой функции $\tilde{f}^*$ от величины ξ_h при различных величинах шага b_w. Рисунок показывает, что, уменьшая величину b_w, можно примерно на 30% улучшить значение $\tilde{f}^*$.

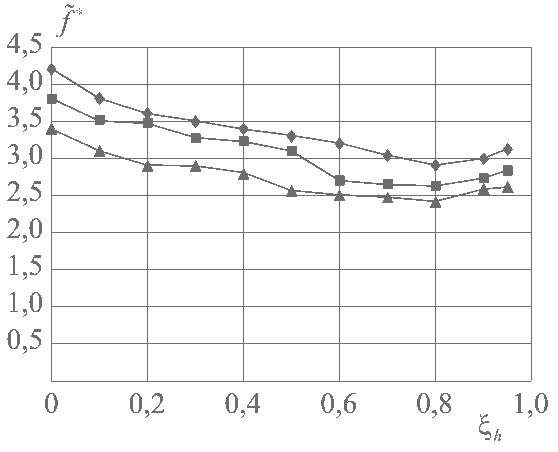


Рис. 5.4. Лучшие достигнутые значения целевой функции в зависимости от вероятности использования памяти гармоний, $\xi_p = 0,4$: ♦ – $b_w = 0,9$; ■ – $b_w = 0,6$; ▲ – $b_w = 0,3$

5.2. АЛГОРИТМ ГРАВИТАЦИОННОГО ПОИСКА

Алгоритм гравитационного поиска (*Gravitational Search algorithm*, *GS*) предложен Рашеди (*E. Rashedi*) и другими авторами в 2009 г. Алгоритм использует аналогию движения тяжелых тел вследствие их гравитационного взаимодействия. Силу гравитации между двумя такими телами, имеющими массу m_1, m_2, определяет закон всемирного тяготения, в соответствии с которым эта сила равна

$$F = \gamma \frac{m_1 m_2}{r^2},$$

где γ – гравитационная постоянная, а r – евклидово расстояние между центрами масс гравитирующих тел. По второму закону Ньютона ускорение, которое приобретает тело массой m под действием этой силы, равно

$$a = \frac{F}{m}.$$

Алгоритм *GS* является развитием детерминированного алгоритма глобального многомерного поиска, получившего название *алгоритма оптимизации центральной силой* (*Central Force Optimization*, *CFO*). Рассмотрим прежде данный алгоритм. Полагаем, что речь идет о максимизации фитнес-функции $\varphi(X)$.

Алгоритм оптимизации центральной силой

Алгоритм *CFO* представляет собой детерминированную модель зондов, движущихся в области поиска под действием силы тяжести. Исходное распределение зондов в области поиска является равномерным, так что

$$x_{i,j} = x_j^- + \frac{(i-1)\,(x_j^+ - x_j^-)}{|S|-1},\ i \in [1:|S|],\ j \in [1:|X|],$$

где $|S|$ – число зондов.

Полагаем, что каждый из зондов s_i имеет переменную массу, равную значению функции приспособленности $\varphi(X)$ в точке текущего положения зонда. Координаты зондов изменяем в процессе итераций по формуле

$$x'_{i,j} = x_{i,j} + \frac{1}{2} a_{i,j},$$

где величину $a_{i,j}$ интерпретируем как текущее ускорение зонда s_i по j-й координате, равное

$$a_{i,j} = \gamma \sum_{\substack{k \in [1:|S|] \\ k \neq i}} \chi\left(sign(\varphi_k - \varphi_i)\left(\varphi_k - \varphi_i\right)^b \frac{x_{k,j} - x_{i,j}}{\left\| X_k - X_i \right\|^c} \right),\ i \in [1:|S|],\ j \in [1:|X|]. \quad (5.4)$$

Здесь $\varphi_l = \varphi(X_l), \varphi_k = \varphi(X_k)$, $\chi(\cdot)$ – ступенчатая единичная функция, используемая с целью исключения отрицательной массы; b, c, γ – заданные константы, из которых γ отождествляют с гравитационной постоянной.

Формулу (5.4) иллюстрирует рис. 5.5, на котором $F_{1,i}$ – сила, действующая на тело массой m_1 со стороны тела с массой m_i, и F_1 – суммарная сила, действующая на первое тело и порождающая ускорение a_1.

Можно сказать, что агенты (зонды) в алгоритме *CFO* коммутируют посредством силы гравитации. Отметим, что в соответствии с формулой (5.4) тяжелые

зонды (которым соответствуют хорошие решения) движутся медленнее, чем более легкие зонды (соответствующие менее хорошим решениям). В результате, спустя некоторое число итераций, все массы должны быть, в пределе, притянуты самой тяжелой массой, которой будет соответствовать решение задачи.

Если некоторый зонд «вылетает» из области решений D, то его возвращают в середину между предыдущим (допустимым) и текущим (недопустимым) положениями.

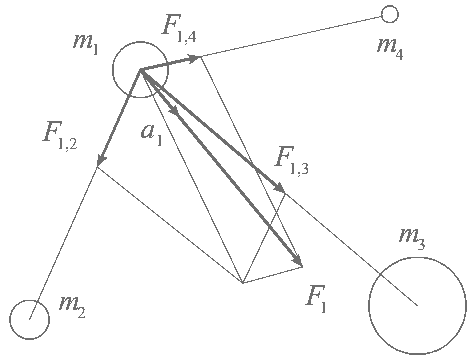


Рис. 5.5. Результат гравитационного взаимодействия тел с различными массами

Схема алгоритма GS

Основные отличия алгоритма *GS* от алгоритма *CFO* состоят в следующем: алгоритм *GS* является стохастическим, а алгоритм *CFO* – детерминированным; массы и координаты зондов в указанных алгоритмах вычисляют по несколько отличным формулам; начальное распределение зондов в алгоритме *CFO* является детерминированным, а в алгоритме *GS* носит случайный характер; в алгоритме *CFO* гравитационная постоянная γ является постоянной, тогда как в алгоритме *GS* она по некоторым правилам изменяется в процессе поиска.

В отличие от того, как это имеет место в природе, алгоритм *GS* в общем случае использует три вида гравитирующих масс – активная m^A, пассивная m^P и инертная m^I массы. Полагаем, что гравитационная сила $F_{i,j}$, которая действует на массу m_i со стороны массы m_j, равна

$$F_{i,j} = \gamma \frac{m_i^P \, m_j^A}{r_{i,j}}, \; i, j \in [1:|S|], \; i \neq j,$$

то есть обратно пропорциональна не квадрату расстояния $r_{i,j}$ между зондами, а лишь первой степени этого расстояния. Авторы утверждают, что такое определение силы повышает эффективность алгоритма *GS*. Ускорение, приобретаемое массой m_i в результате действия силы $F_{i,j}$, определяет формула

$$a_i = \frac{F_{i,j}}{m_i^I}.$$

Говоря более строго, алгоритм *GS* использует следующее правило вычисления силы, действующей на массу m_i со стороны массы m_j по k-му измерению:

$$F_{i,j,k} = \gamma \frac{m_i^P \, m_j^A}{r_{i,j} + \varepsilon} \left(x_{j,k} - x_{i,k} \right), \; k \in [1:|X|].$$

Здесь ε – малая константа, $r_{i,j}$ – текущее евклидово расстояние между массами m_i, m_j. Гравитационную постоянную γ уменьшаем с ростом числа итераций по правилу

$$\gamma(t) = \gamma(0) \left(\frac{1}{t} \right)^d, \; d \in (0;1),$$

где $\gamma(0)$ – ее начальное значение, а d – свободный параметр алгоритма.

Суммарную силу $F_{i,*,k}$, действующую на зонд s_i по k-му измерению, полагают случайной величиной, равной

$$F_{i,*,k} = \sum_{j=1,\ j\neq i}^{|S|} U_1(0;1) F_{i,j,k}. \tag{5.5}$$

Ускорение указанного зонда по тому же измерению находим по формуле

$$a_{i,*,k} = \frac{F_{i,*,k}}{m_i^I}.$$

Если положение зонда s_i по k-му измерению на данной итерации есть $x_{i,k}$, то на следующей итерации его определяет выражение

$$x'_{i,k} = x_{i,k} + v'_{i,k},\ i \in [1:|S|],\ k \in [1:|X|], \tag{5.6}$$

где скорость $v'_{i,k}$ равна

$$v'_{i,k} = U_1(0;1)\, v_{i,k} + a_{i,*,k}. \tag{5.7}$$

Простейший алгоритм GS предполагает равенство активной, пассивной и инертной масс m^A, m^P, m^I, точнее говоря, полагается, что

$$m_i = \frac{\tilde{m}_i}{\sum\limits_{j\in[1:|S|]} \tilde{m}_j},\ i \in [1:|S|],$$

где

$$\tilde{m}_i = \frac{\varphi(X_i) - \varphi^{\min}}{\varphi^{\max} - \varphi^{\min}};$$

$$\varphi^{\max} = \max_{i\in[1:|S|]} \varphi(X_i),\ \varphi^{\min} = \min_{i\in[1:|S|]} \varphi(X_i).$$

Эффективность поиска с помощью алгоритма GS может быть повышена, если с ростом числа итераций сокращать по некоторому правилу в формуле (5.5) число зондов, оставляя лишь наиболее массивные из них. Тем самым можно обеспечить лучший компромисс между диверсификацией поиска на начальных итерациях и его интенсификацией на завершающих итерациях.

Обозначим S^{**} текущий набор лучших (самых тяжелых) зондов, а I^{**} – совокупность номеров этих зондов. Тогда модифицированная формула (5.5) примет вид

$$F_{i,*,k} = \sum_{j\in I^{**},\ j\neq i} U_1(0;1) F_{i,j,k},\ k \in [1:|X|].$$

Число $|S^{**}|$ лучших зондов с ростом числа итераций уменьшают, например, по линейному закону, так что, в конце концов, остается лишь один зонд, притягивающий остальные зонды.

Поскольку в общем случае каждый зонд гравитационно связан со всеми остальными, движение каждого данного зонда определяют все остальные зонды. Заметим, что в алгоритме роя частиц движение агента s_i зависит только от локально и глобально лучших агентов s_i^*, s_i^{**} соответственно.

Зонд, имеющий большую массу, имеет больший радиус притяжения. Поэтому, аналогично алгоритму роя частиц, агенты популяции имеют тенденцию двигаться в сторону лучшего (с точки зрения фитнес-функции) агента.

Зонды с большими инерционными массами движутся медленнее. Вместе с предыдущей особенностью алгоритма *GS* это обстоятельство обеспечивает сужение пространства поиска с ростом числа итераций (интенсификацию поиска).

В отличие от алгоритма роя частиц, алгоритм *GS* не имеет памяти (в алгоритме роя частиц память частицы s_i реализуют частицы s_i^*, s_i^{**}). Движение агентов в алгоритме *GS* зависит от расстояния между взаимодействующими агентами в пространстве поиска, а не в пространстве соседства, как в алгоритме роя частиц.

Некоторые модификации алгоритма GS

Оригинальный алгоритм *GS* склонен к преждевременной сходимости в случае поиска экстремума сложных мультимодальных функций. В этой связи разработано значительное число модификаций алгоритма, имеющих целью преодоление данного его недостатка.

Рассмотрим в качестве варианта решения указанной проблемы введение в оригинальный алгоритм *GS* операторов *знаковой мутации* (*sign mutation*) и *переупорядочивающей мутации* (*reordering mutation*). Данная модификация алгоритма предложена Нобахари (*H. Nobahari*) с соавторами в 2011 г.

Знаковая мутация предполагает вычисление нового положения зонда s_i по k-му измерению по формулам, несколько отличным от формул (5.6), (5.7):

$$x'_{i,k} = x_{i,k} + \tilde{v}'_{i,k},\ \tilde{v}'_{i,k} = \bar{u}_{sign}(\xi_s)\, v'_{i,k},\ i \in [1:|S|],\ k \in [1:|X|], \qquad (5.8)$$

где $\bar{u}_{sign}(\xi_s)$ – случайная величина, с вероятностью ξ_s принимающая значение (–1) и с вероятностью $(1-\xi_s)$ – значение 1:

$$\bar{u}_{sign}(\xi_s) = \begin{cases} -1, & U_1(0;1) < \xi_s, \\ 1, & \text{иначе.} \end{cases}$$

Таким образом, знаковая мутация с вероятностью ξ_s изменяет соответствующую компоненту вектора скорости зонда, вычисленную по правилу (5.7), на противоположную (рис. 5.6).

Переупорядочивающую мутацию применяют к вектору скорости $v'_{i,k}$, вычисленному по формуле (5.7), с вероятностью ξ_v. Мутация состоит в случайном переупорядочении компонент этого вектора (рис. 5.7). Комбинация знаковой и переупорядочивающей мутации обеспечивает в широких пределах варьирова-

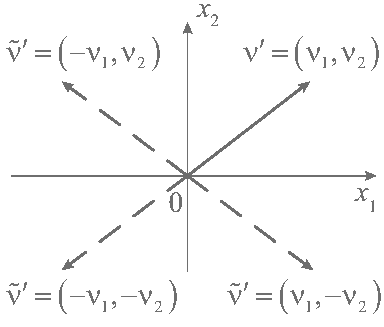


Рис. 5.6. К знаковой мутации: $|X| = 2$

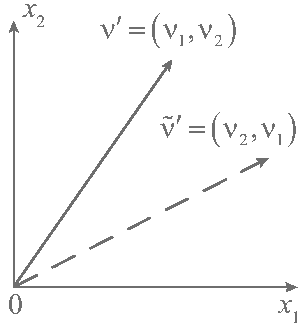


Рис. 5.7. К переупорядочивающей мутации: $|X| = 2$

ние ориентации вектора $\tilde{v}'_{i,k}$. Кроме того, для диверсификации поиска вектор X'_i, вычисленный по формуле (5.8), может быть подвергнут равномерной мутации.

5.3. ЭЛЕКТРОМАГНИТНЫЙ ПОИСК

Рассматриваем два близких алгоритма, основанных на использовании электростатических законов Кулона и законов механики Ньютона, – электромагнитный алгоритм и алгоритм поиска системой зарядов.

Каждого из агентов популяции интерпретируют в данных алгоритмах как заряженную частицу, заряд которой пропорционален значению фитнес-функции в той точке области поиска, в которой на данной итерации находится агент. Текущий заряд частиц популяции определяет суммарную силу, действующую на данную частицу со стороны других частиц, а также направление и величину ее перемещения на текущей итерации. В соответствии с законами электростатики эта сила вычисляется путем векторного суммирования сил притяжения и отталкивания со стороны всех частиц популяции.

Рассматриваем задачу глобальной минимизации в гиперпараллелепипеде *П*.

5.3.1. Электромагнитный алгоритм

Электромагнитный алгоритм (*ElectroMagnetism-like algorithm*, *EM*) предложен Бирбилом (*I. Birbil*) и Фангом (*S. C. Fang*) в 2003 г.

Канонический алгоритм EM

Общая схема алгоритма *EM* включает в себя следующие шаги.

1) Инициализируем популяцию.

2) Выполняем локальный поиск.

3) Вычисляем суммарные силы, действующие на каждую из частиц популяции.

4) Реализуем перемещение частиц.

5) Выполняем проверку выполнения условия окончания итераций и в зависимости от итогов этой проверки либо завершаем вычисления, либо переходим к шагу 2.

Инициализация популяции, во-первых, включает в себя генерацию $|S|$ точек, равномерно распределенных в гиперпараллелепипеде *П*:

$$X_i = X^- + U_{|X|}(0;1)\left(X^+ - X^-\right),\ i \in [1:|S|].$$

Во-вторых, на данном этапе производим вычисление исходных значений фитнес-функции $\varphi(X_i) = \varphi_i$ и определяем лучшее из этих значений

$$\varphi_{i_b} = \min_{i \in [1:|S|]} \varphi_i.$$

Локальный поиск выполняем для каждого из текущих положений частиц X_i с целью сбора локальной информации о ее окружении. Вообще говоря, этот поиск может быть реализован с помощью любого из детерминированных или стохастических алгоритмов локальной оптимизации. Авторы алгоритма *ES* предлагают использовать линейный стохастический поиск (см. 1.2.1).

Вычисление суммарных сил. Из электростатики известно, что модуль силы действия на данную заряженную частицу s_i другой частицы s_j обратно пропорционален расстоянию между ними $r_{i,j}$ и прямо пропорционален произведению их зарядов q_i, q_j, то есть

$$F_{i,j} = \frac{q_i\, q_j}{4\,\pi\,\varepsilon\, r_{i,j}},$$

где значение величины ε определяется свойствами среды, в которой находятся взаимодействующие частицы. Вектор силы $F_{i,j}$ лежит на прямой, проходящей через центры частиц s_i, s_j, и направлен от первой частицы ко второй, если знаки их зарядов различны, и в обратную сторону, если знаки зарядов одинаковы.

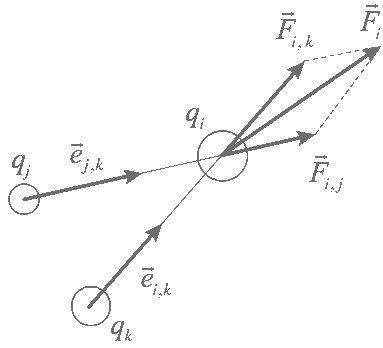


Рис 5.8. К принципу суперпозиции электростатических сил: $|X| = 2$

Если частица s_i находится в электростатическом поле частиц s_j, $j \in [1:|S|]$, $j \neq i$, то справедлив принцип суперпозиции указанных сил, в соответствии с которым результирующий вектор $\vec{F}_i$ сил, действующих на частицу s_i, представляет собой сумму векторов $\vec{F}_{i,j}$. Таким образом, имеет место равенство

$$\vec{F}_i = \sum_{j=1,\, j\neq i}^{|S|} \vec{F}_{i,j} = \sum_{j=1,\, j\neq i}^{|S|} \frac{q_i\, q_j}{4\,\pi\,\varepsilon\, r_{i,j}}\, \vec{e}_{i,j},$$

где $\vec{e}_{i,j}$ – единичный вектор направления $\vec{F}_{i,j}$ (рис. 5.8).

На текущей итерации заряду частицы s_i ставим в соответствие величину

$$q_i = \exp\left(-|X| \frac{\varphi_i - \varphi_{i_b}}{\sum\limits_{j\in[1:|S|],\, j\neq i} \left(\varphi_j - \varphi_{i_b}\right)} \right),\ i \in [1:|S|], \tag{5.9}$$

которая представляет собой нормированное значение фитнес-функции в текущем положении X_i этой частицы. Здесь φ_{i_b} – минимальное достигнутое популяцией к данной итерации значение этой функции. Множитель $|X|$ в формулу (5.9) добавлен с целью предотвращения слишком малых абсолютных значений величины под знаком экспоненциальной функции при высоких размерностях поискового пространства. Аргумент этой функции во всех случаях неположителен, так что заряд q_i всегда положителен и принадлежит интервалу $(0;1]$.

На той же итерации t силе $\vec{F}_i$ ставим в соответствие $(|X| \times 1)$-вектор F_i, определяемый формулой

$$F_i = \sum_{j=1,\, j\neq i}^{|S|} F_{i,j} = \sum_{j=1,\, j\neq i}^{|S|} \begin{cases} \left(X_j - X_i\right) \dfrac{q_i\, q_j}{\left\|X_j - X_i\right\|^2}, & \varphi_j < \varphi_i, \\ \left(X_i - X_j\right) \dfrac{q_i\, q_j}{\left\|X_j - X_i\right\|^2}, & \varphi_i \le \varphi_j. \end{cases} \tag{5.10}$$

Из формулы (5.10) следует, что частица с лучшим значением фитнес-функции притягивает частицу с худшими значениями этой функции, и наоборот –

вторая частица отталкивает первую. Из этой же формулы вытекает, что частица s_{i_b}, имеющая на данной итерации значение фитнес-функции, равное φ_{i_b}, притягивает все остальные частицы популяции (подобно глобально лучшей частице в алгоритме роя частиц с топологией типа клика).

Перемещение частиц выполняем по правилу

$$X_i' = X_i + \lambda\, U_1(0;1)\frac{F_i}{\|F_i\|} \otimes V_i,\ i \in [1:|S|],\ i \neq i_b,\ X_{i_b}' = X_{i_b}. \qquad (5.11)$$

Здесь λ – шаг перемещения (свободный параметр); компоненты вектора V_i имеют значения

$$v_{i,j} = \begin{cases} (x_j^+ - x_{i,j}), & F_{i,j} > 0, \\ (x_{i,j} - x_j^-), & F_{i,j} \le 0, \end{cases} \quad j \in [1:|X|],\ j \neq i. \qquad (5.12)$$

Формулы (5.11), (5.12) означают, что при перемещении частицы s_i из положения X_i в положение X_i' используем нормированную силу (5.10). По каждому из измерений вектора X перемещение производим с шагом случайного размера в направлении соответствующей верхней или нижней границ параллелепипеда Π. Частицу s_{i_b} на данной итерации не перемещаем.

Окончание итераций. В оригинальной версии алгоритма *EM* условием завершения вычислений является достижение заданного числа итераций $\hat{t}$.

Для алгоритма *EM* его авторами рекомендованы следующие значения свободных параметров: число итераций локального поиска $\hat{t}_l$ в несколько раз превышает величину $|X|$; величина δ_s (п. 1.2.1), определяющая максимально допустимую величину шага локального поиска, принимает значения от 10^{-2} до 10^{-4}; максимальное число итераций $\hat{t} = 25|X|$.

Авторы алгоритма выполнили широкое экспериментальное сравнение его эффективности с эффективностью многих известных популяционных и непопуляционных алгоритмов глобальной оптимизации. Исследование показывает, что обычно алгоритм *EM* показывает сравнимые, а часто лучшие результаты, чем эти алгоритмы.

Модификации алгоритма EM

Известно большое число модификаций алгоритма *EM*, часть из которых предложена самими авторами алгоритма.

Локальный поиск. Прежде всего, процедура локального поиска вообще может быть исключена из схемы алгоритма. Такая модификация во многих случаях обеспечивает вполне приемлемую эффективность алгоритма по критерию вероятности локализации глобального экстремума многоэкстремальной функции.

Процедура локального поиска может быть применена не ко всем частицам популяции, как в каноническом алгоритме *EM,* а только к лучшей на данной итерации частице. Цель такой модификации – сократить число испытаний, требуемых процедурой, с тем, чтобы обеспечить лучший баланс между вычислительной сложностью и точностью алгоритма. Среднее число испытаний в этом случае оказывается близким к величине, которую обеспечивает первый вариант модификации, в то время как вероятность локализации глобального экстремума сопоставима с такой же вероятностью канонического алгоритма.

Таким образом, для большинства задач применение локального поиска ко всем частицам популяции может быть излишним.

Алгоритм линейного стохастического поиска, используемый в процедуре локального поиска канонического алгоритма *EM*, может быть заменен другим более мощным популяционным или непопуляционным алгоритмом. Например, известны гибридизации алгоритма *EM* с алгоритмом рассеянного поиска, а также с так называемым алгоритмом поиска по образцам.

Правила вычисления сил F_i. Известно несколько вариантов правила вычисления зарядов q_i, фигурирующих в формуле для этих сил (5.10). Так, может быть использовано иное масштабирование величин зарядов:

$$q_i = \exp\left(-|X|\frac{\varphi_i - \varphi_{i_w}}{\varphi_{i_w} - \varphi_{i_b}}\right),\ i \in [1:|S|].$$

Здесь $\varphi_{i_w} = \max\limits_{i\in[1:|S|]} \varphi_i$ – худшее значение фитнес-функции на данной итерации.

Возможна замена экспоненциальной функции другими монотонными функциями, например функцией вида

$$q_i = -\frac{1}{|X|\dfrac{\varphi_i - \varphi_{i_w}}{\varphi_{i_w} - \varphi_{i_b}}},\ i \in [1:|S|],\ i \neq i_w,$$

которая также гарантирует выполнение условия $q_i \in (0;1]$, но обеспечивает по сравнению с предыдущей формулой более быстрое убывание зарядов частиц, отличных от лучшей частицы, с ростом размерности задачи $|X|$.

Известен вариант вычисления силы F_i с учетом ее значения на предыдущей итерации:

$$F_i = F_i + \beta F_i\ (t-1),\ i \in [1:|S|],\ t > 1.$$

Здесь $\beta \in [0;1]$ – *константа памяти алгоритма*.

Кардинальная модификация канонического алгоритма вычисления силы F_i основана на использовании *возмущающей частицы* (*perturbed point*) s_{i_p}, в качестве которой может выступать частица, располагающаяся на данной итерации наиболее далеко от лучшей частицы s_{i_b}:

$$\varphi(X_{i_p}) = \max_{i\in[1:|S|],\, i\neq i_b} \left\|X_{i_b} - X_i\right\|_E,\quad i \in [1:|S|].$$

Для всех $i \in [1:|S|]$, $i \neq i_p$ сила F_i вычисляется в этом случае по формуле (5.10). Для частицы s_{i_p} формула несколько трансформируется и приобретает вид

$$F_{i_p} = \sum_{\substack{j=1\\ j\neq i_p}}^{|S|} F_{i_p,j},\ F_{i_p,j} = \begin{cases} (X_j - X_{i_p})\dfrac{U_1(0;1)\, q_{i_p} q_j}{\left\|X_j - X_{i_p}\right\|_E^2}, & \varphi(X_j) < \varphi(X_{i_p}),\\[2ex] (X_{i_p} - X_j)\dfrac{U_1(0;1)\, q_{i_p} q_j}{\left\|X_j - X_{i_p}\right\|_E^2}, & \text{иначе;}\end{cases}$$

$$F_{i_p,j} = \begin{cases} F_{i_p,j}, & U_1(0;1) < \overline{\xi},\\ -F_{i_p,j}, & \text{иначе.}\end{cases}$$

Здесь $\overline{\xi}$ – вероятность инверсии величины $F_{i_p,j}$ (свободный параметр алгоритма).

Таким образом, по сравнению с остальными частицами, каждая из компонент силы F_{i_p}, действующей на частицу s_{i_p}, во-первых, возмущена случайной величиной, равномерно распределенной в интервале [0; 1], и во-вторых, с вероятностью $\overline{\xi}$ может быть инвертирована.

Закон перемещения частиц. Очевидной является модификация формулы (5.11) с использованием адаптивной длины шага λ. Можно предложить различные законы изменения этой величины, например, авторы алгоритма *EM* предлагают закон

$$\lambda' = \frac{\lambda}{1+\alpha\lambda}, \ \lambda(0)=1,$$

где $\alpha \in (0;1)$ – заданная константа (свободный параметр алгоритма).

Условия окончания итераций. Другим очевидным направлением модификации канонического алгоритма *EM* является использование условий окончания поиска, отличных от условия, которое использует этот алгоритм. Например, возможно использование условия стагнации вычислительного процесса в течение δ_t итераций.

В случае оптимизации тестовых функций с известным значением фитнес-функции в точке глобального экстремума φ^* может быть использовано условие

$$\frac{\varphi_{i_b} - \varphi^*}{\left|\varphi^*\right|} \le \delta_\varphi,$$

где δ_φ – заданная положительная константа, определяющая требуемую точность локализации экстремума φ^*.

Учет ограничений. Канонический алгоритм *EM* предполагает решение задачи оптимизации в параллелепипеде *П*. Для решения задачи условной оптимизации с ограничениями общего вида могут быть использованы различные варианты барьерных и штрафных функций.

5.3.2. Алгоритм поиска системой зарядов

Алгоритм поиска системой зарядов (*Charged System Search*, *CSS*) впервые представлен Кавехом (*A. Kaveh*) и Талатахари (*S. Talatahari*) в 2010 г.

В обозначениях п. 5.3.1 в алгоритме *CSS текущий заряд* q_i частицы s_i определяет выражение

$$q_i = \frac{\varphi_i - \varphi_{i_b}}{\varphi_{i_w} - \varphi_{i_b}}, \ i \in [1:|S|],$$

а текущее расстояние $r_{i,j}$ между частицами s_i, s_j – выражение

$$r_{i,j} = \frac{\left\| X_i - X_j \right\|_E}{\left\| \frac{X_i - X_j}{2} - X_{i_b} \right\|_E + \varepsilon}, \ i, j \in [1:|S|], \ i \ne j,$$

где положительная малая константа ε введена для того, чтобы избежать сингулярности.

Так же, как в алгоритме *EM,* силы взаимодействия между частицами популяции могут быть притягивающими и отталкивающими. Для частиц s_i, s_j характер этой силы определяет константа $c_{i,j}$:

$$c_{i,j} = \begin{cases} -1, & \varphi_i < \varphi_j, \\ 1, & \varphi_i \geq \varphi_j. \end{cases}$$

В отличие от алгоритма *EM*, каждая из заряженных частиц s_i представляет собой сферу, радиус которой равен величине $a > 0$ (свободный параметр алгоритма). В результате этого соглашения суммарная сила, действующая на частицу s_i со стороны всех остальных частиц популяции, имеет вид

$$F_i = q_i \sum_{\substack{j=1 \\ j \neq i}}^{|S|} \left(\frac{q_j\, r_{i,j}\, b_1}{a^3} + \frac{q_j\, b_2}{r_{i,j}^2} \right) c_{i,j}\, (X_j - X_i),\ i \in [1:|S|],$$

где значения величин b_1, b_2 определяют условия

$$b_1 = 0,\ b_2 = 1, \text{ если } r_{i,j} \geq a,$$

$$b_1 = 1,\ b_2 = 0, \text{ если } r_{i,j} < a.$$

Миграция частицы s_i в области поиска подчинена правилу

$$X_i' = X_i + \lambda_v U_1(0;1) V_i + \lambda_a U_1(0;1) \frac{F_i}{q_i},\ V_i = X_i - X_i(t-1),\ i \in [1:|S|].$$

Здесь второе слагаемое моделирует скорость частицы, а третье – ее ускорение; λ_v, λ_a – весовые коэффициенты, определяющие веса указанных величин соответственно (свободные параметры алгоритма).

Относительные значения величин λ_v, λ_a определяют баланс между диверсификацией и интенсификацией поиска. Увеличение значений параметра λ_v усиливает влияние предыстории поиска, повышая исследовательские свойства алгоритма. Большие значения λ_a приводят к сильному влиянию притягивающих сил и могут вызвать преждевременную сходимость алгоритма. Напротив, малые значения этой величины замедляют скорость сходимости алгоритма, обеспечивая более широкое исследование области поиска.

Для того чтобы в начале поиска обеспечить его диверсификацию, а в конце поиска – интенсификацию, целесообразно использовать динамические значения параметров λ_v, λ_a. По одной из рекомендаций возможны следующие законы их изменения:

$$\lambda_v = 1 + \frac{t}{\hat{t}},\ \lambda_a = 3\left(1 - \frac{t}{\hat{t}}\right).$$

5.4. АЛГОРИТМ ЭВОЛЮЦИИ РАЗУМА

Концепция алгоритма эволюции разума (*Mind Evolutionary Computation*, *MEC*) предложена в 1998 г. в работе Ченгай (*S. Chengyi*) и других авторов. Алгоритм *MEC* моделирует скорее некоторые аспекты поведения человека в об-

ществе, чем, как можно было бы предположить, работу человеческого мозга. В алгоритме *MEC* каждый индивид рассматривается как разумный агент, функционирующий в некоторой группе людей. При принятии решений он ощущает влияние как со стороны членов своей группы, так и со стороны членов других групп. Точнее говоря, чтобы достичь высокого положения в обществе, индивиду приходится учиться у наиболее успешных индивидов в своей группе. В то же время, для того чтобы группа, которой принадлежит данный индивид, становилась более успешной по сравнению с другими группами, и этот индивид, и все индивиды его группы должны руководствоваться тем же самым принципом в межгрупповой конкуренции.

Из рассмотренной концепции алгоритма *MEC* следует, что чрезвычайную важность в этом алгоритме имеет информация индивидов внутри группы об успехах друг друга, а также аналогичная взаимная информированность групп. Данное утверждение отражает тот факт, что для успешного развития общества разумных индивидов эти индивиды, а также их объединения должны непрерывно и свободно обмениваться информацией.

Алгоритм *MEC* удобно интерпретировать как многопопуляционный алгоритм. Мультипопуляция алгоритма *MEC* состоит из *лидирующих групп* (*superior groups*) $\boldsymbol{S}^b = (S_1^b, S_2^b, ..., S_{|\boldsymbol{S}^b|}^b)$ и *отстающих групп* (*temporary groups*) $\boldsymbol{S}^w = (S_1^w, S_2^w, ..., S_{|\boldsymbol{S}^w|}^w)$, включающих в себя $|\boldsymbol{S}^b|$ и $|\boldsymbol{S}^w|$ субпопуляций соответственно. В оригинальном алгоритме *MEC* числа индивидов в каждой из субпопуляций полагаются одинаковыми и равными $|S|$.

Каждая из субпопуляций S_i^b, S_j^w имеет свою коммуникационную среду, которая названа авторами *локальной доской объявлений* (*local billboard*). Обозначим эти доски C_i^b, C_j^w соответственно. Кроме того, вся мультипопуляция $\{\boldsymbol{S}^b, \boldsymbol{S}^w\}$ имеет общую, *глобальную доску объявлений* (*global billboard*) C^g. Если далее в обозначениях субпопуляций и соответствующих локальных досок объявлений индексы b, w опущены, то имеется в виду произвольная из этих субпопуляций и их досок объявлений. Размерность такой субпопуляции обозначаем $|S|$.

В первом пункте рассматриваем так называемый простой алгоритм *MEC*, а в последующих пунктах – расширенный, улучшенный и хаотичный алгоритмы соответственно. Полагаем, что речь идет о задаче условной максимизации фитнес-функции в параллелепипеде *П*.

5.4.1. Простой алгоритм эволюции разума

Исходная версия алгоритма *MEC*, названная авторами *простым алгоритмом эволюции разума* (*Simple MEC*, *SMEC*), построена на основе операций инициализации групп, *локальных состязаний* (*similar-taxis*) и *диссимиляции* (*dissimilation*).

Операция инициализации групп создает группы $\boldsymbol{S}^b$, $\boldsymbol{S}^w$ из $|S|$ индивидов каждая и размещает их в области поиска. Рассмотрим схему операции на примере группы S_i.

1) Генерируем случайный вектор $X_{i,1}$, компоненты которого равномерно распределены в области поиска *П*. Отождествляем этот вектор с индивидом $s_{i,1}$ группы S_i.

2) Определяем начальные координаты остальных индивидов данной группы $s_{i,j}$, $j \in [2:|S|]$ по формуле

$$X_{i,j} = X_{i,1} + N_{|X|}(0, \sigma), \tag{5.13}$$

то есть размещаем остальные индивиды случайным образом вокруг индивида $s_{i,1}$ в соответствии с нормальным законом распределения, имеющим по всем $|X|$ координатам нулевое математическое ожидание и среднее квадратичное отклонение σ (свободный параметр алгоритма).

Операция локальных состязаний реализует локальный поиск максимума фитнес-функции каждой из групп S_i^b, S_j^w, $i \in [1:|\boldsymbol{S}^b|]$, $j \in [1:|\boldsymbol{S}^w|]$. Рассмотрим в качестве примера схему этой операции для группы S_i.

1) С доски объявлений C_i берем информацию о текущем индивиде-победителе группы S_i. Пусть это будет индивид s_{i,j_b}, $j_b \in [1:|S|]$.

2) Определяем новые координаты остальных индивидов $s'_{i,j}$, $j \in [1:|S|]$, $j \neq j_b$, данной группы по правилу вида (5.13), то есть размещаем их случайным образом вокруг победителя в соответствии с нормальным законом распределения.

3) Вычисляем *счета* (*scores*) агентов группы: полагаем, что счет $\varphi'_{i,j}$ индивида $s'_{i,j}$ равен значению фитнес-функции в его текущем положении $X'_{i,j}$, то есть $\varphi'_{i,j} = \varphi(X'_{i,j})$.

4) Определяем нового победителя группы s'_{i,k_b}, $k_b \in [1:|S|]$, то есть индивида данной группы, который имеет максимальный текущий счет:

$$\max_{k \in [1:|S|]} \varphi'_{i,k} = \varphi(X'_{i,k_b}) = \varphi'_{i,k_b}.$$

5) Заносим информацию о новом победителе группы s'_{i,k_b} на доски объявлений C_i, C^g.

Операция диссимиляции управляет глобальным поиском. Схема операции имеет следующий вид.

1) С глобальной доски объявлений C^g считываем текущие счета победителей групп S_i^b, S_j^w, $i \in [1:|\boldsymbol{S}^b|]$, $j \in [1:|\boldsymbol{S}^w|]$.

2) Выполняем сравнение указанных счетов между собой. Если счет некоторой лидирующей группы S_i^b меньше счета одной из отстающих групп S_j^w, то последняя группа занимает место группы S_i^b во множестве лидирующих групп, а группа S_i^b – место группы S_j^w среди отстающих групп. Если счет группы S_j^w ниже счетов всех лидирующих групп, то удаляем группу S_j^w из популяции.

3) С помощью операции инициализации взамен каждой из удаленных групп инициализируем новую группу.

В алгоритме *SMEC* операции локальных состязаний и диссимиляции итерационно повторяются до тех пор, пока имеет место увеличение максимального счета лидирующих групп. При прекращении роста этого счета решение задачи, соответствующее победителю лучшей из лидирующих групп, объявляем точкой глобального максимума.

5.4.2. Расширенный алгоритм

Модификация *EMEC* (*Extended Mind Evolutionary Computation*) алгоритма *MEC* предложена в одной из работ Цзе (*J. Jie*) с соавторами и существенно развивает последний алгоритм. Так, алгоритм *EMEC* использует модифициро-

ванные операции локальных состязаний и диссимиляции, а также новую *операцию кооперации* (*cooperation*). В алгоритме *EMEC* каждая из лидирующих и отстающих групп в процессе своего развития проходит три стадии развития – *инициализация* (*initialization*), *эволюция* (*evolution*), *зрелость* (*maturity*). Группа приобретает статус зрелой, если ее счет не изменяется в течение заданного числа итераций. Если в ходе эволюции данная группа сближается на некоторое фиксированное расстояние с другой группой или перекрывает ее, то первая из групп получает статус *ассимилирующей* (*assimilative*) *группы*. Информация о статусе группы заносится на глобальную доску объявлений.

Модифицированный оператор локальных состязаний. Пусть s_{i,j_b}, $j_b \in [1:|S|]$ – текущий победитель группы S_i. Тогда в соответствии с данным оператором координаты остальных индивидов этой группы на следующей итерации определяет формула

$$\begin{cases} X'_{i,j} = X_{i,j} + \lambda\, N_1(0;1)\, V_{i,j}, & j \in [1:|S|],\ j \neq j_b, \\ X'_{i,j} = X_{i,j}, & j = j_b, \end{cases} \tag{5.14}$$

где $V_{i,j}$ – $(|X| \times 1)$-вектор текущего *шага обучения* (*learning step*) индивида $s_{i,j}$; $\lambda = \lambda(t) \in (0;1)$ – заданный *когнитивный* (*cognitive*) *параметр*, обеспечивающий уменьшение величины шага $V_{i,j}$ с ростом числа итераций.

Направление шага $V_{i,j}$ в формуле (5.14) совпадает с направлением $X_{i,j_b} - X_{i,j}$, то есть с направлением от текущего положения индивида $s_{i,j}$ к текущему положению победителя данной группы s_{i,j_b}. Размер шага $V_{i,j}$ определяет величину обучающего воздействия победителя на индивида $s_{i,j}$ и является некоторой возрастающей функцией разности счетов победителя и данного индивида $\varphi_{i,j_b} - \varphi_{i,j}$. В простейшем случае в качестве этой функции используют линейную функцию, так что

$$\|V_{i,j}\| = \alpha(\varphi_{i,j_b} - \varphi_{i,j}),$$

где α – положительный коэффициент пропорциональности (свободный параметр).

Модифицированная операция диссимиляции. В процессе выполнения операции, в первую очередь, проверяем текущий статус группы. Если она является зрелой или ассимилирующей, то удаляем эту группу из популяции, а взамен создаем новую группу. Пусть это будет группа S_k. Первого индивида $s_{k,1}$ этой группы, в отличие от алгоритма *MEC*, размещаем в области поиска не с помощью генератора случайных равномерно распределенных чисел, а на основе алгоритма имитации отжига (см. 1.4.2). Заметим, что вследствие этого алгоритм *EMEC* следует классифицировать как гибридный популяционный алгоритм.

В отличие от алгоритма *MEC*, полученные координаты $X_{k,1}$ индивида $s_{k,1}$ сравниваем с текущими координатами победителей всех групп (информацию берем с глобальной доски объявлений C^g). Новую группу в окрестности точки $X_{k,1}$ инициализируем только в том случае, если расстояние между индивидом $s_{k,1}$ и этими победителями превышает заданное минимально допустимое расстояние.

Операция кооперации. На содержательном уровне операция состоит в том, что в процессе эволюции каждая из групп на основе информации с глобальной доски объявлений C^g находит соседнюю лучшую группу (обладающую более

высоким счетом) и изменяет свое положение в пространстве поиска, перемещаясь в направлении этой группы.

Пусть s_{i,k_b} – текущий победитель группы S_i. Положим, что s_{j,l_b} – текущий победитель соседней лучшей группы S_j, так что справедливо неравенство $\varphi_{j,l_b} > \varphi_{i,k_b}$. Тогда операцию кооперации для индивида $s_{i,k}$, $k \neq k_b$, определяет формула

$$\begin{cases} X'_{i,k} = X_{i,k} + N_1(0,1)\ V_{i,k}, \\ V_{i,k} = \lambda\,(X_{j,l_b} - X_{i,k_b}), \end{cases} \quad i, j \in [1:|S|],\ i \neq j,\ k \in [1:|S|],$$

где $\lambda > 0$ – длина шага.

Как и в алгоритме роя частиц, в общем случае при определении соседней лучшей группы S_j могут быть использованы различные топологии соседства.

5.4.3. Улучшенный алгоритм

Дальнейшее развитие *MEC*-алгоритма представляет собой так называемый *улучшенный MEC-алгоритм* (*Improved MEC*, *IMEC*), авторами которого являются Цзе (*J. Jie*) и Цзэн (*J. Ch. Zeng*). Основные особенности алгоритма *IMEC* заключаются в следующем: при инициализации групп алгоритм использует стратегию с разделением области поиска на подобласти; используется самоадаптирующийся алгоритм, реализующий операцию локальных состязаний; операция диссимиляции расширена средствами преодоления нишевания (см. 2.6). Последовательно рассмотрим указанные особенности алгоритма.

Инициализация с разделением пространства поиска (*region-partition initialization*) имеет целью предотвратить преждевременную сходимость алгоритма. Идея состоит в том, что на этапе начальной инициализации мультипопуляции область поиска *П* тем или иным образом разделяют на гиперпараллелепипеды $П_1$, $П_2$,... равного объема, и начальные точки каждой из групп генерируют внутри этих подобластей (случайным образом на основе равномерного закона распределения). Требуемое разбиение области *П* на подобласти можно выполнить, например, путем деления каждого из интервалов $[x_i^-; x_i^+]$ на η равных подынтервалов, где величину η находят как минимальное из натуральных чисел, удовлетворяющих условию $\eta^{|X|} > |S^b| + |S^w|$.

Самоадаптирующийся (*self-adaptive*) *механизм* операции локальных состязаний представляет собой один из вариантов метаоптимизации (гл. 7). Механизм использует расширенный вектор варьируемых параметров задачи, включающий в себя помимо вектора *X* также вектор шага миграции индивида. Для индивида $s_{i,j} \in S_i$, $i \in [1:|S|]$ основная формула самоадаптирующейся операции локальных состязаний имеет вид

$$\begin{cases} x'_{i,j,k} = x_{i,j,k} + \lambda_x\ N_1(0,1)\ v_{i,j_b,k}, \\ v'_{i,j_b,k} = v_{i,j_b,k} + \lambda_v\ N_1(0,1)\ a_{i,j_b,k}, \end{cases} \quad j \in [1:|S|],\ j \neq j_b,\ k \in [1:|X|], \qquad (5.15)$$

где $x_{i,j,k}$, $v_{i,j,k}$ – компоненты расширенного вектора $(X_{i,j}, V_{i,j})$ варьируемых параметров индивида $s_{i,j}$; $x_{i,j_b,k}$, $v_{i,j_b,k}$ – аналогичные компоненты расширенного вектора варьируемых параметров победителя данной группы s_{i,j_b}; $\lambda_x, \lambda_v \in (0;1)$ – свободные вещественные параметры, определяющие величины соответствую-

щих шагов. Динамический параметр $a_{i,j_b,k} = a_{i,j_b,k}(t)$ определяет вклад случайной составляющей в компоненту $v'_{i,j_b,k}$:

$$a_{i,j_b,k} = \frac{x_{i,j_b,k} - x_{i,j_b,k}(t_{i,j_b})}{t - t_{i,j_b}}.$$

Здесь t_{i,j_b} – номер итерации, на котором был выявлен индивид s_{i,j_b} в качестве победителя группы S_i.

В качестве критерия окончания итераций (5.15) модифицированная операция локальных состязаний использует условие

$$\frac{1}{|X|}\sum_{k=1}^{|X|} v_{i,j_b,k} \le \varepsilon_v,\ i \in [1:|S|]. \tag{5.16}$$

Иначе говоря, итерации продолжаются до тех пор, пока среднее значение шага миграции победителя данной группы не станет меньше заданной величины $\varepsilon_v > 0$ (свободный параметр алгоритма). Отметим, что условие (5.16) одновременно является условием зрелости группы S_i.

Расширенная операция диссимиляции. Преодоление нишевания в этой операции реализовано путем задания минимально допустимого расстояния между двумя разными группами, в качестве которого используется величина

$$\rho(S_i, S_j) = \sqrt{\frac{1}{|X|}\sum_{m=1}^{|X|}\left(\frac{x_{i,k_b,m} - x_{j,l_b,m}}{x_m^+ - x_m^-}\right)^2},\ i \ne j,$$

где s_{i,k_b}, s_{j,l_b} – победители групп S_i, S_j соответственно.

Если выполняется неравенство $\rho(S_i, S_j) \le \varepsilon_\rho$, то группы S_i, S_j считаем принадлежащими одной нише. Группу, имеющую больший счет, сохраняем в качестве представителя данной ниши, другую группу удаляем из популяции, а взамен ее создаем новую группу. Здесь $\varepsilon_\rho > 0$ – заданная величина (свободный параметр алгоритма).

После инициализации новой группы она проходит серию соревнований со всеми теми группами, информация о которых занесена на глобальную доску объявлений C^g. Целью соревнований является проверка того, что новая группа сгенерирована в еще не исследованной мультипопуляцией части области поиска *П*. Если группа не выдерживает указанную проверку, ее не включают в популяцию и осуществляют инициализацию новой группы.

5.4.4. Хаотический алгоритм эволюции разума

Хаотическая модификация *MEC*-алгоритма предложена Лю (*J. Liu*) с соавторами в 2010 г. Модифицированный алгоритм получил название *CMEC* (*Chaotic MEC*). Опуская детали, можно сказать, что *CMEC*-алгоритм представляет собой гибридизацию рассмотренного выше алгоритма *MEC* и хаотического алгоритма локального поиска (см. 1.4.2), используемого в качестве операции локальных состязаний.

Начальную инициализацию мультипопуляции алгоритм *CMEC* реализует с помощью палаточного отображения (см. 1.4.2). Для инициализации групп, которые в процессе функционирования алгоритма создаются взамен удаленных групп, используется логистическое отображение (п. 1.4.2). Применение

не классических генераторов случайных чисел, а хаотических последовательностей обеспечивает более равномерное распределение индивидов в области решения.

5.5. СТОХАСТИЧЕСКИЙ ДИФФУЗИОННЫЙ ПОИСК

Алгоритм стохастического диффузионного поиска (*Stochastic Diffusion Search*, *SDS*) предложен в 1989 г. Бишопом (*J. Bishop*). До настоящего времени алгоритм преимущественно развивается его автором, а также Насуто (*S. Nasuto*). По сравнению с другими популяционными алгоритмами данный алгоритм отличает достаточно глубокое математическое обоснование. В исходном виде *SDS* представляет собой алгоритм дискретной оптимизации. Модификация алгоритма для задачи глобальной непрерывной оптимизации была предложена только в 2011 г.

Алгоритм инспирирован игрой «Ресторан». Суть игры состоит в следующем. Группа делегатов участвует в длительной конференции в незнакомом городе, в котором имеется большое число ресторанов, каждый из которых предлагает большой выбор блюд. Делегаты ставят перед собой задачу найти лучший из ресторанов, то есть такой, который понравится наибольшему числу делегатов. Проверка каждого ресторана и его ассортимента каждым делегатом займет слишком много времени. Поэтому делегаты используют следующую стратегию поиска.

Каждый делегат формирует свою гипотезу относительно того, какой ресторан является лучшим в городе. Вечером он проверяет свою гипотезу, ужиная в этом ресторане и заказывая одно случайное блюдо из меню. На следующее утро те делегаты, которые не были довольны своим ужином, просят поделиться своими впечатлениями случайного коллегу. Если его впечатления положительны, то недовольный делегат принимает выбор этого коллеги. В противном случае он выбирает новый случайный ресторан. Используя эту стратегию, достаточное большое число делегатов в короткий срок может найти лучший ресторан города.

В терминах алгоритма *SDS* схема рассмотренной стратегии поиска лучшего ресторана имеет следующий вид.

1) На этапе *инициализации* алгоритма агенты высказывают начальную *гипотезу* (*hypothesis*) о лучшем ресторане.

2) Каждый агент *тестирует* (*test*) свою гипотезу, то есть ужиная в выбранном ресторане, подтверждает или опровергает свое первоначальное мнение.

3) Путем прямых контактов между агентами происходит *диффузия* (*diffusion*) или *обмен гипотезами* и результатами их проверки.

4) На этой основе агенты формируют новые гипотезы, тестируют их и вновь обмениваются гипотезами. И так далее до выполнения условий *сходимости* (*convergence*, *halt*) итераций.

Рассмотрим задачу глобальной минимизации в параллелепипеде

$$\Pi = \{X \mid x_j^- \le x_j \le x_j^+,\ j \in [1:|X|]\}.$$

Положим, что фитнес-функция также должна быть минимизирована.

Инициализация. Генерируем случайные начальные положения агентов

$X_i, i \in [1:|S|]$ и вычисляем соответствующие значения фитнес-функции $\varphi(X_i) = \varphi_i$. Рекомендуемая численность популяции $|S|$ достаточно высока – от 100 до 1000 агентов.

Тестирование. Приспособленность агента s_i сравниваем с приспособленностью случайного агента s_j; $i, j \in [1:|S|]$, $j \neq i$. Если $\varphi_i \leq \varphi_j$, то агента s_i объявляем *активным* (*active*), а в противном случае – *неактивным* (*inactive*). Заметим, что смысл понятия *активный агент* существенно различен при решении дискретной и непрерывной задач оптимизации с помощью алгоритма *SDS*. Активность агента в первом случае говорит о том, что его гипотеза подтвердилась. Во втором случае в паре сравниваемых агентов активным считается агент с наилучшим (наименьшим) значением фитнес-функции.

Диффузия. Для каждого из агентов $s_i, i \in [1:|S|]$ выполняем следующие действия.

1) Агент s_i неактивен. Производим выбор случайного агента s_j, $j \in [1:|S|]$, $j \neq i$. Если этот агент является активным, то агент s_i присваивает случайную гипотезу X_i' из некоторой окрестности $d(X_j)$ агента s_j, где

$$d(X_j) = \{X \mid x_j^l \leq x_k \leq x_j^u,\ k \in [1:|X|]\};$$
$$x_j^l = \max\left((x_j - 0{,}5b(x_j^+ - x_j^-),\ x_j^-\right),$$
$$x_j^u = \min\left((x_j + 0{,}5b(x_j^+ - x_j^-),\ x_j^+\right).$$

Значение свободного параметра b лежит в интервале $(0;1)$ и обычно равно 0,1.

2) Агент s_i активен. Случайно или детерминированно выбираем для этого агента новую гипотезу. Рекомендуют выбор гипотезы в центре наибольшей из текущих неисследованных областей множества Π по следующей схеме:

– сортируем текущие координаты всех агентов популяции по каждому из измерений x_k;

– находим наибольший интервал $[x_k^{\min}; x_k^{\max}]$ между двумя соседними точками по каждому из измерений x_k;

– помещаем агента s_i в точку X_i с координатами

$$x_{i,k} = x_k^{\min} + 0{,}5\left(x_k^{\max} - x_k^{\min}\right),\ k \in [1:|X|].$$

Сходимость. Алгоритм *SDS* может использовать различные условия окончания итераций. Кроме критериев, являющихся общими для всех популяционных алгоритмов, могут использоваться так называемые *сильный* и *слабый* критерии.

Сильный критерий сходимости (*strong halting criterion*) использует тот факт, что в процессе итераций алгоритма *SDS* имеет место кластеризация, то есть формирование подмножеств (кластеров) агентов, разделяющих одну и ту же гипотезу. При этом самый многочисленный кластер включает в себя оптимальное решение. В соответствии с сильным критерием, итерации завершают при выполнении следующих условий: мощность самого большого кластера превышает минимально допустимое значение; эта мощность не меняется в течение заданного числа итераций (имеет место стагнация размера кластера).

Слабый критерий (*weak halting criterion*) предполагает завершение итераций в следующих условиях: число активных агентов превышает минимально допустимое значение; это число не меняется в течение заданного числа итераций (имеет место стагнация числа активных агентов).

Рассмотренный канонический алгоритм *SDS* является неадаптивным, поскольку все свободные параметры определены априори. Адаптивная модификация канонического алгоритма *SDS* представляет собой *фокусированный алгоритм* (*focused SDS*).

Канонический алгоритм *SDS* является однопопуляционным. Вместе с тем известны многопопуляционные модификации этого алгоритма, например, так называемый *контекстно-зависимый SDS* (*context-sensitive SDS*).

5.6. КУЛЬТУРНЫЙ АЛГОРИТМ

Культурный алгоритм (*Cultural algorithm*, *C-алгоритм*) представляет собой, по сути, концепцию совершенствования популяционных алгоритмов поисковой оптимизации путем учета опыта, приобретаемого в ходе решения задачи. Принято считать, что алгоритм предложен Рейнольдсом (*R. Reynolds*) в 1994 г.

В процессе развития общества люди накапливают информацию об окружающем мире. Эту информацию можно считать базой знаний общества, которую его члены могут эксплуатировать с целью оптимизации своего поведения. В *C*-алгоритме эту базу знаний формализуют в виде *пространства убеждений* (*belief space*) и используют в качестве еще одного элемента эволюционного воздействия на популяцию. Таким образом, если говорить, например, о генетическом алгоритме, то его *культурная модификация* предполагает учет как генетической, так и культурной информации.

В самом общем виде *C*-алгоритм можно определить как вычислительную модель эволюции популяции, в которой опыт решения задач индивидами накапливается, объединяется, обобщается и конкретизируется в пространстве убеждений, а затем используется в целях управления развитием индивидов и популяции в целом.

5.6.1. Общая схема алгоритма

Рассмотрим задачу глобальной условной минимизации с ограничениями типа неравенств

$$\min_{X \in D \cap \Pi} f(X) = f(X^*) = f^*,$$

$$D = \{X \mid G(X) \geq 0\},\ \Pi = \{X \mid x_i^- \leq x_i \leq x_i^+,\ i \in [1:|X|]\}.$$

Назовем ограничения, определяемые ограничивающей вектор-функцией $G(X)$ и параллелепипедом Π, *ограничениями задачи* и *ограничениями области допустимых значений* соответственно. Положим, что множество $D \cap \Pi$ является выпуклым и всюду в этом множестве фитнес-функция $\varphi(X)$ (также подлежащая минимизации) принимает неотрицательные значения.

Основными компонентами *C*-алгоритма являются популяция и пространство убеждений (рис. 5.9).

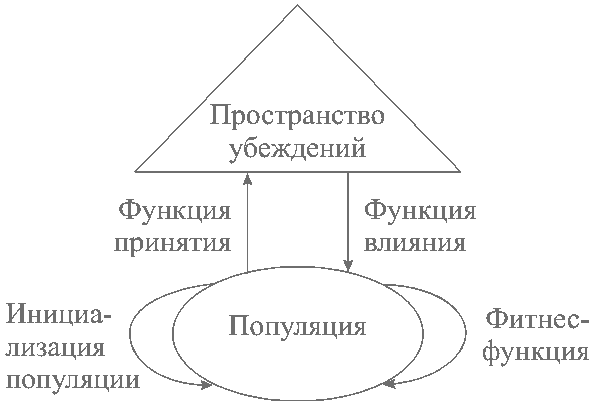


Рис. 5.9. Структура культурного алгоритма

Пространство убеждений и популяция взаимодействуют посредством *функции принятия* (*acceptance*) и *функции влияния* (*influence*). С помощью первой из этих функций определяют множество наилучших индивидов, которые имеют возможность корректировать пространство убеждений. Вторая функция задает правила, по которым убеждения способны влиять на эволюцию индивидов популяции.

В простейшем случае функция принятия использует индивидов с приспособленностью, которая выше средней приспособленности по популяции. При построении функции влияния могут быть использованы два подхода – изменение приспособленности индивидов и изменение принципов модификации индивидов.

Псевдокод культурного алгоритма представлен на рис. 5.10.

```
Begin
    t=0;
    Initialize Population S(t);
    Initialize Belief space B(t);
    Evaluate Population S(t);
    repeat
        Communicate (S(t), B(t));
        Adjust Belief space B(t);
        Communicate (B(t), S(t));
        t=t+1;
        Evolve Population S(t);
        Evaluate Population S(t);
    until (termination condition)
End
```

Рис. 5.10. Псевдокод культурного алгоритма

На рис. 5.10 функции *Initialize Population* $S(t)$ и *Initialize Belief space* $B(t)$ выполняют инициализацию популяции $S(0)$ и пространства убеждений $B(0)$ соответственно. Функция *Evaluate Population* $S(t)$ оценивает индивидов попу-

ляции с помощью фитнес-функции. Функции *Communicate* ($S(t)$, $B(t)$) и *Communicate* ($B(t)$, $S(t)$) реализуют взаимодействие между популяцией и пространством убеждений посредством функций принятия и влияния соответственно. Функция *Adjust Beliefspace* $B(t)$ выполняет коррекцию пространства убеждений, а функция *Evolve Population* $S(t)$ отвечает за эволюцию популяции на текущей итерации.

5.6.2. Построение и коррекция пространства убеждений

Рассмотрим простейший случай, когда в пространстве убеждений концентрируется информация о способности индивидов удовлетворять ограничениям задачи. Информация об ограничениях задачи выражается в пространстве убеждений в форме *сети ограничений* (*interval constraint-network*), которая состоит из *узлов* (*nodes*), соединенных между собой посредством *ветвей* (*branches*). Узел представляет текущий интервал или несвязанное множество интервалов, определяемых ограничениями области допустимых значений и ограничениями задачи. Ветвь показывает зависимость узлов, которые она связывает.

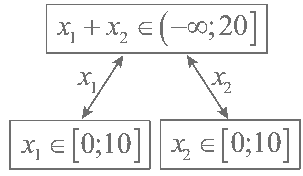


Рис. 5.11. Пример сети ограничений

Рассмотрим простой пример. Положим, что размерность вектора варьируемых параметров задачи равна двум $(|X| = 2)$ и имеют место ограничения области допустимых значений

$$x_1 \in [0;10],\ x_2 \in [0;10]$$

и ограничения задачи

$$g_1(X) = x_1 + x_2 \le 20,$$

($|G| = 1$). На рис. 5.11 продемонстрирована соответствующая сеть ограничений сразу после инициализации алгоритма. Сеть имеет два узла, соответствующих ограничению $X \in \Pi$, и один узел, соответствующий ограничению $G(X) \ge 0$.

Коррекцию пространства убеждений осуществляют путем коррекции сети ограничений с помощью рассматриваемых ниже операций. Каждую из этих операций применяют к текущим интервалам, связанным с соответствующими узлами сети ограничений. Координаты индивидов, отобранных с помощью функции принятия, обозначаются ниже символом «+» и могут находиться как внутри, так и за пределами текущего интервала пространства убеждений [·].

Операция конкретизации (*specialization*) используется в том случае, когда число отобранных индивидов в пределах некоторых частей интервала меньше заданного минимального числа элементов. Суть операции состоит в сжатии интервала (рис. 5.12).

Операцию обобщения (*generalization*) применяют в случае, когда хотя бы некоторые из отобранных индивидов находятся за пределами текущего интервала. Результатом операции является расширенный интервал, включающий в себя всех этих индивидов (рис. 5.13).

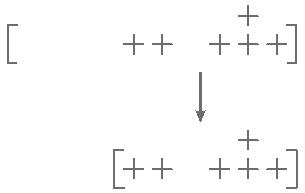


Рис. 5.12. Пример операции конкретизации

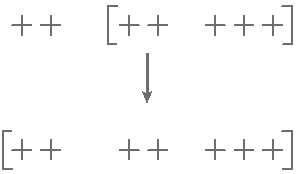


Рис. 5.13. Пример операции обобщения

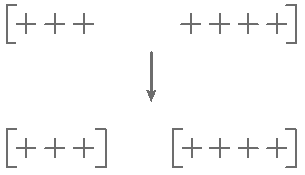


Рис. 5.14. Пример операции разделения

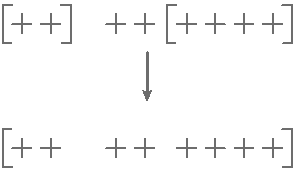


Рис. 5.15. Пример операции слияния

Операция разделения (*fission*) представляет собой особый случай операции конкретизации, при котором внутреннюю пустую часть текущего интервала отбрасывают с целью получения двух интервалов (рис. 5.14).

Операция слияния (*fusion*) предназначена для объединения двух несвязных интервалов при условии существования между ними отобранных индивидов (рис. 5.15).

Операция распространения (*propagation*) позволяет скорректировать ограничения задачи на основе измененных ограничений области допустимых значений. Операция использует результаты интервальной арифметики (рис. 5.16).

Операция обратного распространения (*back propagation*) обратна операции распространения и предназначена для коррекции ограничений области допустимых значений на основе скорректированных ограничений задачи (рис. 5.17).

С использованием рассмотренных операций обновление сети ограничений производим по следующей схеме:

– текущие значения индивидов, отобранных функцией принятия, отображаем в соответствующие ограничения области допустимых значений и ограничения задачи;

– корректируем сеть последовательным применением операторов распространения и обратного распространения.

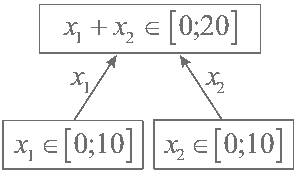


Рис. 5.16. Пример операции распространения

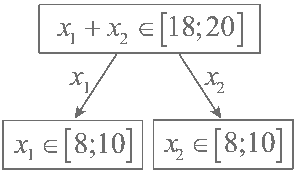


Рис.5.17. Пример операции обратного распространения

5.6.3. Культурная модификация генетических операторов

Как отмечалось выше, возможна культурная модификация любого из популяционных алгоритмов поисковой оптимизации, то есть гибридизация этих алгоритмов с *C*-алгоритмом. Гибридизация сводится к культурной модификации операторов соответствующего популяционного алгоритма. Рассмотрим в качестве примера культурную модификацию некоторых таких операторов, используемых в известной программной системе *GENOCOP* (*GEnetic algorithm for Numerical Optimization for COnstrained Problems*).

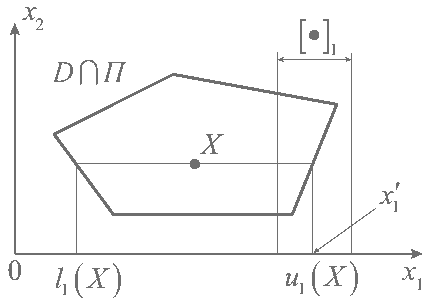


Рис. 5.18. К культурному аналогу граничной мутации: $k = 1$

Оператор граничной мутации. Если $X = (x_1, ..., x_k, ...x_{|X|})$ – хромосома до мутации, то граничную мутацию этой хромосомы выполняем путем случайного выбора гена x_k и присваивания этому гену значения

$$x_k' = \begin{cases} l_k(X), & U_1(0:1) = 0, \\ u_k(X), & U_1(0:1) = 1. \end{cases} \tag{5.17}$$

Здесь $l_k(X)$, $u_k(X)$ – нижняя и верхняя границы области $D \cap \Pi$ по *k*-му измерению в точке *X* соответственно. Таким образом, культурную модификацию оператора граничной мутации определяет правило

$$x_k' = \begin{cases} l_k(X), & (l_k(X) \in [\cdot]_k) \wedge (u_k(X) \notin [\cdot]_k), \\ u_k(X), & (u_k X) \in [\cdot]_k) \wedge (l_k(X) \notin [\cdot]_k), \\ x_k', & \text{иначе}, \end{cases} \tag{5.18}$$

где $[\cdot]_k$ – текущий интервал пространства убеждений по *k*-му измерению; величина x_k' в правой части равенства вычислена по формуле (5.17). Для случая $|X| = 2$ правило (5.18) иллюстрирует рис. 5.18 (показана вторая ситуация).

Оператор неравномерной мутации Михалевича (п. 2.3.2) определяет формула

$$x_k' = \begin{cases} x_k + \delta(t, u_k(X) - x_k), & U_1(0:1) = 0, \\ x_k - \delta(t, x_k - l_k(X)), & U_1(0:1) = 1. \end{cases}$$

Культурный аналог данного оператора имеет вид

$$x_k' = \begin{cases} x_k + \delta(t, u_k(X) - x_k), & (d[\cdot]_k > 0) \wedge (x_k < c[\cdot]_k), \\ x_k - \delta(t, x_k - l_k(X)), & (d[\cdot]_k < 0) \wedge (x_k < c[\cdot]_k), \\ x_k + u_{sign}^{\pm 1}\, \delta(t, x_k - l_k), & \text{иначе}, \end{cases} \tag{5.19}$$

где

$$d[\cdot]_k = \begin{cases} 1, & (c[\cdot]_k' - c[\cdot]_k) > 0, \\ -1, & (c[\cdot]_k' - c[\cdot]_k < 0, \\ 0, & \text{иначе}, \end{cases}$$

$c[\cdot]_k$ – координата центра текущего интервала пространства убеждений по *k*-му измерению.

Гауссов оператор мутации задает выражение

$$x'_k = x_k + N_1(m, \sigma).$$

Культурный вариант этого оператора определяет выражение вида (5.19) с тем отличием, что в качестве шага мутации используется величина

$$v = \begin{cases} N_1(0; 0,1), & [\cdot]_k^+ - [\cdot]_k^- > 0,1, \\ N_1(0; [\cdot]_k^+ - [\cdot]_k^-), & \text{иначе}, \end{cases}$$

где величина $[\cdot]_k^+ - [\cdot]_k^-$ представляет собой длину текущего интервала пространства убеждений по k-му измерению.

5.7. МЕМЕТИЧЕСКИЕ АЛГОРИТМЫ

Википедия определяет *мемéтику* (*memetics*) как подход к эволюционным моделям передачи информации, который основывается на концепции *мемов* (*meme*), а мемы, являющиеся аналогами генов,– как единицы культурной информации, распространяемые между людьми посредством имитации, научения и других средств. Понятие мема было предложено и основы меметики разработаны Докинзом (*C. R. Dawkins*) в 1976 г.

Мемы могут передаваться вертикально – от предшественников, родителей, воспитателей, книг, культурных артефактов и тому подобное. Возможна также горизонтальная передача мемов от человека к человеку и от культуры к культуре. Важно, что, хотя по существу мемы представляют собой чистую информацию, их функционирование приводит к существенным изменениям поведения человека.

Меметические алгоритмы (*Memetic algorithms*, *М-алгоритмы*) определяют как гибридные популяционные метаэвристические алгоритмы поисковой оптимизации, основанные на концепции мема и неодарвиновском принципе эволюции. В контексте *М*-алгоритмов мем является реализацией какого-либо алгоритма локальной оптимизации, уточняющего текущие решения на каждой итерации либо через некоторое число итераций. В широком смысле *М*-алгоритмы можно рассматривать как гибридизацию одного из популяционных алгоритмов глобального поиска и одного или нескольких классических или популяционных алгоритмов локальной оптимизации. Первоначально *М*-алгоритмы были предложены как один из вариантов повышения эффективности генетических алгоритмов.

Как у всех популяционных алгоритмов, эффективность *М*-алгоритмов в значительной степени зависит от значений их свободных параметров. В ряде исследований показано, что очень большое влияние на эффективность этих алгоритмов оказывает выбор используемых мемов. Поэтому данная проблема занимает одно из центральных мест в работах, посвященных *М*-алгоритмам.

5.7.1. Общая схема алгоритма

Рассматриваем задачу глобальной условной минимизации в общей постановке, когда фитнес-функция также требует минимизации.

Общую схему *М*-алгоритма передает следующая последовательность шагов.

1) Полагаем $t = 0$ и инициализируем популяцию $S = \{s_i, i \in [1:|S|]\}$.

2) Для текущего положения X_i каждого из агентов s_i вычисляем соответствующее значение фитнес-функции $\varphi(X_i) = \varphi_i$.

3) Формируем промежуточную популяцию S', включающую в себя $|S'|$ лучших индивидов текущей популяции S.

4) Применяем ко всем агентам популяции S' миграционные операторы рассматриваемого популяционного алгоритма.

5) Исходя из новых текущих положений каждого из агентов s'_i, $i \in [1:|S'|]$, выполняем локальный поиск.

6) Для найденных новых положений X''_i агентов, совокупность которых, полагаем, образует популяцию S'', вычисляем соответствующие значения фитнес-функции $\varphi(X''_i) = \varphi''_i$.

7) На основе популяций S, S'' формируем новую популяцию $S(t+1)$.

8) Полагаем $t = t+1$ и если условия окончания итерации не выполнены, переходим к шагу 2, в противном случае заканчиваем вычисления, принимая в качестве решения задачи лучшее положение агента текущей популяции.

Очевидно, что общая схема *M*-алгоритма оставляет широкие возможности для разработки большого числа вариантов этого алгоритма. Укажем следующие основные направления синтеза этих вариантов.

• Локальный поиск можно использовать не на каждой итерации, а через некоторое фиксированное или изменяющееся по определенному закону число итераций.

• Наряду с приведенным выше правилом, можно реализовать множество других правил отбора агентов в промежуточную популяцию S'. Эти правила могут быть как статическими, не зависящими от номера итерации, так и динамическими, изменяющимися тем или иным образом с ростом номера итерации.

• Можно применить большое число различных условий окончания локального поиска (опять же, как статических, так и динамических).

• Для всех агентов популяции S' на всех итерациях может быть использован один и тот же алгоритм локальной оптимизации или разные алгоритмы для разных групп агентов и/или для различных итераций.

В вычислительной практике чаще всего используют *мультимемные* (*multimemes*) адаптивные алгоритмы. Основной проблемой конструирования таких алгоритмов является выбор целесообразной стратегии использования того или иного мема из роя доступных мемов $M = \{m_j,\ j \in [1:|M|]\}$. Задача относится к классу задач метаоптимизации (гл. 7). Указанные стратегии иногда называют *гиперэвристиками* (*hyperheuristic*).

Выбор мемов из роя M может происходить в зависимости от значений их некоторых характеристик и/или рассматриваемого в данный момент фрагмента области решения. Обычно в качестве характеристики мемов, на основе которой происходит их выбор, используют некоторую оценку эффективности мемов.

5.7.2. Гиперэвристические мультимемные адаптивные алгоритмы

Наиболее известны три следующие категории гиперэвристик, используемых в гиперэвристических адаптивных мультимемных алгоритмах, – случайные (*random*), жадные (*greedy*), гиперэвристики с функцией выбора (*choice-function hyperheuristics*).

Случайные гиперэвристики. В этой категории гиперэвристик самой простой является стратегия *простого случайного выбора* (*simple random choice*), когда в каждой точке принятия решения мем выбирают случайным образом из роя мемов M и вероятность выбора каждого из мемов не меняется в процессе итераций. Данная стратегия, по сути, является основой для всех остальных стратегий.

Другой известной стратегией данной категории является стратегия *случайного спуска* (*random descent choice*). Здесь на первом шаге мем выбирают из роя M случайным образом. Выбранный мем используют до тех пор, пока он не перестанет обеспечивать локальные уточнения решений. Затем случайным образом выбирается другой мем и т. д.

Вариантом последней стратегии является стратегия *спуска со случайной перестановкой* (*random permutation descent choice*). В этом варианте вначале фиксируется случайная перестановка $\tilde{M}$ мемов m_j, $j \in [1:|M|]$. После того, как используемый мем m_j исчерпает себя, циклически используют следующий мем m_{j+1} в перестановке $\tilde{M}$.

Жадные гиперэвристики предполагают, что сначала все мемы m_j, $j \in [1:|M|]$ применяются к каждому из агентов популяции S', а затем выбирается мем, показавший наилучший результат. Недостатком данной эвристики являются высокие вычислительные затраты.

Гиперэвристики с функцией выбора основаны на использовании функции выбора $\phi(m_j)$, которая может быть построена на основе одной или нескольких метрик, отражающих различные аспекты качества мемов. Известным вариантом функции $\phi(m_j)$ является трехкомпонентная функция

$$\phi(m_j, m_k) = \lambda_1\phi_1(m_j) + \lambda_2\phi_2(m_j, m_k) + \lambda_3\phi_3(m_j), \;\; j, k \in [1:|M|], \;\; j \neq k,$$

представляющая собой аддитивную свертку функций $\phi_1(m_j), \phi_2(m_j, m_k), \phi_3(m_j)$ с весами $\lambda_1, \lambda_2, \lambda_3$. Полагаем, что лучшему мему соответствует большее значение функции $\phi(m_j, m_k) = \phi(m_j)|_{m^- = m_k}$, где m^- – предыдущий использованный мем.

Компонента $\phi_1(m_j)$ определяет величину последних улучшений фитнес-функции, сделанных мемом m_j, и формализует ту идею, что если данный мем показал высокую эффективность на предшествующих итерациях, то, вероятнее всего, он будет эффективным и в течение некоторого числа последующих итераций. Вторая компонента $\phi_2(m_j, m_k)$ учитывает возможный синергетический эффект от последовательного применения мемов m_j, m_k. Компонента формализует величину последних улучшений фитнес-функции, сделанных комбинацией (последовательным использованием) этих мемов. Третья компонента $\phi_3(m_j)$ призвана повысить разнообразие используемых мемов путем увеличения вероятности применения тех из них, которые давно не выполняли локальную оптимизацию. В качестве значений функции $\phi_3(m_j)$ может быть использовано число итераций M-алгоритма, в течение которых мем m_j не использовался ни разу.

Известно несколько гиперэвристик на основе функции выбора. Рассмотрим некоторые из них.

Простейшей является *стратегия прямого выбора* (*straight choice*), предполагающая, что на каждом шаге принятия решения выбирается мем m', который обеспечивает наилучшее значение функции выбора $\phi(m', m_k)$:

$$\phi(m', m_k) = \max_{j \in [1:|M|],\, j \neq k} \phi(m_j, m_k).$$

В *стратегии ранжированного выбора* (*ranked choice*) сначала мемы m_j, $j \in [1:|M|]$, ранжируют согласно их текущим значениям функции выбора $\phi(m_j, m_k)$. Затем проводят эксперименты по локальному уточнению текущих решений с помощью нескольких лучших мемов. В качестве мема m' используют мем, обеспечивший наибольшее относительное уменьшение значения фитнес-функции.

Стратегия рулеточного выбора (*roulette choice*) основана на выборе мема $m' = m_l$ с вероятностью

$$\xi_l = \frac{\phi(m_l, m_k)}{\sum_{j=1}^{|M|} \phi(m_j, m_k)}, \quad l \in [1:|M|].$$

В *стратегии декомпозиционного выбора* (*decomposition choice*) в первую очередь формируют набор мемов $\tilde{M} = \{m_{j_1}, m_{j_2}, m_{j_3}, m_{j_4}\}$, которые обеспечивают максимальные значения функций $\phi_1(m_j, m_k)$, $\phi_2(m_j, m_k)$, $\phi_3(m_j, m_k)$, $\phi_4(m_j, m_k)$ соответственно; $j_1, j_2, j_3, j_4 \in [1:|M|]$, $j_1, j_2, j_3, j_4 \neq k$. Затем проводят эксперименты по локальному уточнению текущих решений с помощью каждого из мемов роя $\tilde{M}$. Аналогично стратегии ранжированного выбора в качестве мема m' используют лучший из испытанных мемов. Заметим, что в рое $\tilde{M}$ могут быть совпадающие мемы, так что число различных мемов в этом рое может принимать значения от единицы до четырех.

Известна также *стратегия выбора с запретами* (*tabu-search choice*), в которой на основе функции выбора $\phi(m_j, m_k)$ сначала рой мемов M сужают до роя $\tilde{M}$ путем формирования в каждой точке принятия решения *списка запрещенных мемов* (*tabu-list*). Затем из роя $\tilde{M}$ выбирают мем m' по схеме стратегии декомпозиционного выбора.

Гиперэвристики могут строиться также на основе идеи *декомпозиции на подзадачи* (*sub-problem decomposition*). Эта идея предполагает динамическое разбиение области решения на подобласти, а самой задачи оптимизации – на соответствующие подзадачи; выбор наиболее подходящего мема для каждой из этих подзадач. При выборе мема такие гиперэвристики используют информацию, накопленную в процессе решения задачи, об эффективности каждого из мемов роя M в соответствующей подобласти поиска. В результате для каждой из подобластей в пределе применяется свой мем, что, как правило, значительно повышает эффективность поиска.

5.7.3. Самоадаптирующиеся мультимемные алгоритмы

Самоадаптирующиеся мультимемные алгоритмы представляют собой меметическую модификацию генетического алгоритма. Каждый агент s_i, $i \in [1:|S|]$ популяции S кодируют мультихромосомой $\overline{X}_i = (X_i, m_{i_k})$, $i_k \in [1:|M|]$, первая часть которой представляет собой соответствующее решение задачи, а вторая часть – номер мема в наборе M, с помощью которого получено данное решение. По общему правилу эволюции мультихромосом генетические операторы применяют к каждой из указанных частей хромосомы в отдельности, не скрещивая эти части между собой. Таким образом, данные алгоритмы реализуют метод самоадаптивного управления (см. 7.6).

Мутация мемной части мультихромосомы $\overline{X}_i$ состоит в случайной замене по тому или иному правилу мема m_{i_k} новым мемом из роя M. Скрещивание мемных частей хромосом $\overline{X}_i$, $\overline{X}_j$, $i, j \in [1:|S|], i \neq j$ производим по следующей схеме. Потомок $\overline{X}'_i$ приобретает мема m_{i_k} в том случае, если оценка улучшения решения, достигнутая данным мемом, лучше такой же оценки мема m_{j_k}. В противном случае потомку $\overline{X}'_i$ передаем мема m_{j_k}. Если оценки эффективности мемов родителей m_{i_k}, m_{j_k} одинаковы, то потомок получает один из этих мемов, выбранный случайным образом.

5.8. САМООРГАНИЗУЮЩИЙСЯ МИГРАЦИОННЫЙ АЛГОРИТМ

Самоорганизующийся миграционный алгоритм (*Self Organizing Migrating algorithm, SOM*) представил Зелинка (*I. Zelinka*) в 1999 г. Алгоритм вдохновлен коллективным поведением интеллектуальных индивидуумов.

Положим, что речь идет о безусловной максимизации фитнес-функции $\varphi(X)$. В алгоритме *SOM* популяция агентов эволюционирует в циклах, которые называют *миграционными циклами* (*migration loops*). В каноническом алгоритме число миграционных циклов (число поколений) фиксировано и равно $\hat{t}$. Общую схему канонического алгоритма *SOM* передает следующая последовательность его основных шагов.

1) Инициализация популяции S – случайным образом распределяем индивидов s_i, $i \in [1:|S|]$ в пространстве поиска. Полагаем $t = 0$.

2) Применяем к текущей популяции миграционные операторы базового популяционного алгоритма.

3) Объявляем индивида $s_{i_b} = s_*$, которому соответствует лучшее значение фитнес-функции, текущим лидером популяции.

4) Для каждого из индивидов популяции s_i, исключая лидера, выполняем следующие действия.

4.1) Принимаем в качестве лучшего положения индивида s_i его текущее положение: $X_i^* = X_i$, $\varphi_i^* = \varphi(X_i)$.

4.2) Перемещаем индивида s_i в направлении лидера s_{i_b} на фиксированное число шагов n. На каждом шаге этого перемещения вычисляем соответствующее значение фитнес-функции и, если оно лучше текущего значения φ_i^*, обновляем это значение.

4.3) Перемещаем индивида s_i в новое лучшее положение X_i^*.

5) Если условия окончания поиска выполнены, завершаем миграционные циклы, полагая лучшее из найденных решений приближенным решением задачи. В противном случае возвращаемся к шагу 2.

Шаг перемещения индивида s_i в направлении лидера s_{i_b} выполняем в соответствии с формулой

$$X'_i = X_i + \lambda V_i \otimes \left(X_{i_b} - X_i\right),\ i \neq i_b,$$

где $\lambda \in (0;1]$ – длина текущего шага; V_i – $(|X| \times 1)$-рандомизирующий случайный бинарный вектор, компоненты которого определяет выражение

$$v_{i,j} = \begin{cases} 0, & U_1(0;1) \leq \xi_m, \\ 1, & \text{иначе}, \end{cases} \quad j \in [1:|X|].$$

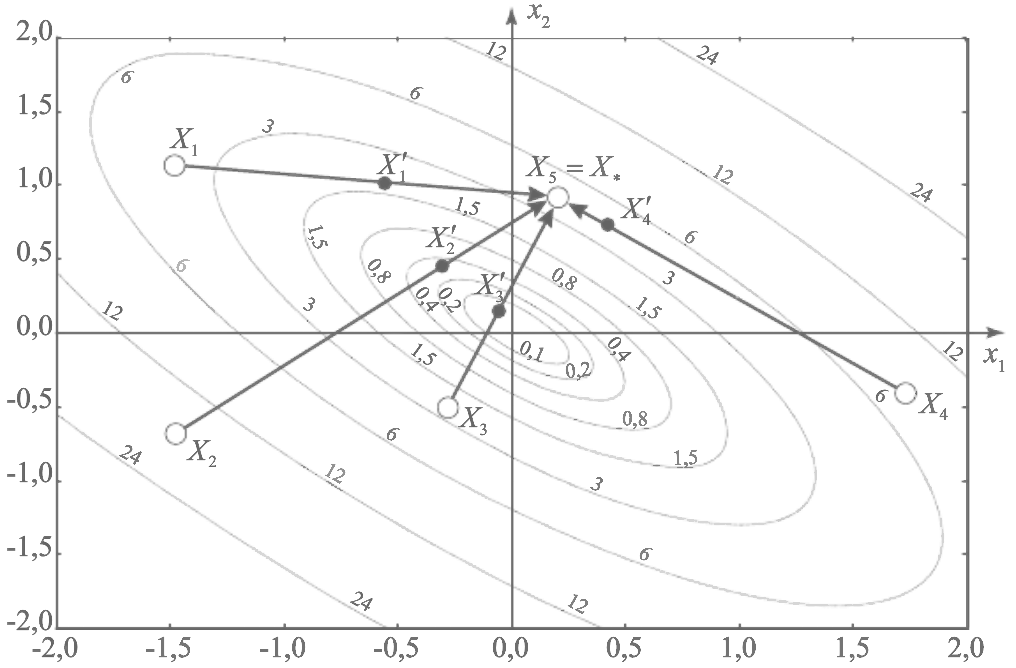


Рис. 5.19. К правилу миграции индивидов: $|X| = 2$; $|S| = 5$; все рандомизирующие векторы V_i, $i \in [1:5]$ являются единичными; $X_* = X_{i_b}$

Здесь $\xi_m \in (0;1]$ – вероятность мутации (свободный параметр алгоритма).

Рандомизация в алгоритме *SOM* преследует те же цели, что и мутация в генетическом алгоритме, то есть прежде всего обеспечение разнообразия популяции. Подчеркнем, что в текущем миграционном цикле для каждого из индивидов s_i рандомизация производится только один раз, так что направление движения в сторону лидера в процессе этого движения не меняется. Нулевое значение компоненты $v_{i,j}$ означает, что перемещение индивида s_i по *j*-му измерению в данном миграционном цикле запрещено (это измерение *заморожено*). Таким образом, единичный вектор V_i означает отсутствие рандомизации.

В случае, когда рандомизация отсутствует, правило миграции индивидов в направлении лидера s_* иллюстрирует рис. 5.19, на котором текущие положения индивидов показаны светлыми кружками, а их положения после миграции – темными.

Эффект рандомизации иллюстрирует рис. 5.20, где приняты следующие значения компонент *рандомизирующего вектора*:

$$V_i^1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \; V_i^2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Обычно в алгоритме *SOM* разрешено замораживание не всех $|X|$ компонент вектора V_i, а лишь их некоторого числа $k \le |X|$. Таким образом, в каждом миграционном цикле поиск производится только по $(|X| - k)$ измерениям. Номера замораживаемых компонент полагают случайными величинами. Рекомендуются следующие значения свободных параметров алгоритма: $\lambda \in [1{,}1; 3]$; $\xi_m = 0{,}1$.

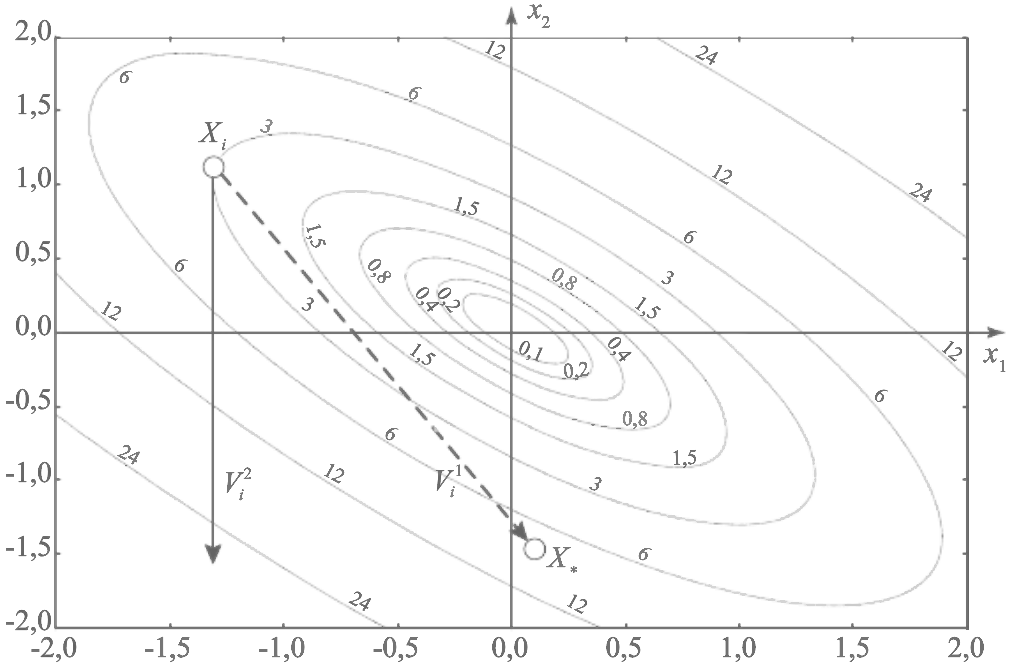


Рис. 5.20. К рандомизации миграции индивидов: $|X| = 2;\ X_* = X_{i_b}$

5.9. АЛГОРИТМЫ РАССЕЯННОГО ПОИСКА И ПРОКЛАДКИ ПУТЕЙ

Непрерывный вариант *алгоритма рассеянного поиска* (*Scatter Search algorithm, SS*) был представлен Гловером (*F. W. Glover*) в 1977 г. *Алгоритм прокладки путей* (*Path Relinking algorithm*, *PR*) предложен тем же автором и Лагуной (*M. Laguna*) в 1997 г. Алгоритмы *SS* и *PR* предназначены для решения задачи глобальной условной оптимизации.

5.9.1. Алгоритм рассеянного поиска

Концептуальная схема алгоритма *SS* имеет следующий вид.

1) Формируем начальный набор векторов решений так, чтобы обеспечить достаточный уровень разнообразия популяции. Применяем к указанным решениям миграционные операторы базового популяционного алгоритма с целью улучшить эти решения.

2) Создаем новые решения, представляющие собой некоторые комбинации так называемых *опорных решений* (*reference set*). Поскольку такое комбинирование может порождать решения как принадлежащие, так и не принадлежащие области допустимых значений, модифицируем решения так, чтобы они стали допустимыми.

3) Применяем к созданным на шаге 2 решениям миграционные операторы, которые использованы на шаге 1, для улучшения этих решений. Операторы должны быть в состоянии обрабатывать как допустимые, так и недопустимые решения, трансформируя последние в допустимые.

4) Выделяем *лучшие* решения из числа решений, полученных на шаге 3, и добавляем их в набор опорных решений.

5) Повторяем шаги 2–4 до тех пор, пока набор опорных решений не перестанет изменяться.

6) Диверсифицируем текущий набор опорных решений путем возврата к шагу 1. Вычисления заканчиваем при выполнении условий окончания итераций.

При выборе лучших решений могут быть использованы не только значения фитнес-функции, но и, например, решения, которые в максимальной степени разнообразят текущий набор опорных решений.

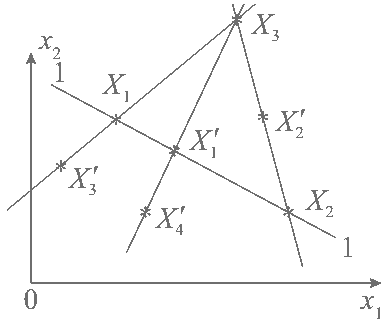


Рис. 5.21. Комбинирование опорных решений: $|X| = 2$

Основная особенность алгоритма *SS* заключена в процедуре комбинирования опорных решений (шаг 2). Для случая $|X| = 2$ простейший вариант этой процедуры иллюстрирует рис. 5.21, на котором принято, что X_1, X_2, X_3 – опорные решения. Использовано правило линейного комбинирования двух опорных решений. Новые решения $X_1' \div X_4'$ получены в результате линейной комбинации пар решений X_1, X_2; X_2, X_3; X_1, X_3; X_3, X_1' соответственно. Точнее говоря, линейная комбинация решений X_1, X_2, например, порождает множество точек, лежащих на прямой 1–1, а решение X_1' соответствует одной из этих точек, выбранных в соответствии с некоторым критерием выбора.

Если сравнивать алгоритм *SS* с генетическим алгоритмом, то процедуру комбинирования опорных решений можно интерпретировать как своеобразный многородительский кроссовер.

Введем следующие обозначения: $\{X_j^r = X_{i_j},\ i_j \in [1:|S|],\ j \in [1:|S^r|]\}$ – набор опорных решений; $S^r = \{s_{i_j}^r,\ j \in [1:|S^r|]\}$ – совокупность соответствующих агентов популяции *S*. Число агентов $|S^r|$ в наборе S^r, вообще говоря, не является фиксированной величиной, но может меняться в процессе итерации.

Положим, что фитнес-функция $\varphi(X)$ подлежит минимизации и всюду в области допустимых значений вектора варьируемых параметров принимает неотрицательные значения. Используем комбинацию опорных решений в виде аддитивной взвешенной свертки.

Схема простейшей версии алгоритма *SS* во введенных обозначениях и соглашениях имеет следующий вид.

1) Полагаем $S = Null$. Генерируем случайный вектор X и, если $X \notin S$, добавляем его во множество S. Повторяем данные действия до тех пор, пока мощность популяции S не станет равной априори заданной величине $|S|$. Выбираем $|S^r| < |S|$ лучших и максимально разнообразных решений из популяции S и помещаем их во множество опорных решений S^r.

2) Применяем ко всем решениям множества S^r миграционные операторы базового популяционного алгоритма и упорядочиваем полученные решения в порядке убывания соответствующих значений фитнес-функции. Для простоты записи положим, что в результате X_1^r оказывается худшим, а $X_{|S^r|}^r$ – лучшим решениями.

3) Из текущих решений S^r формируем множество S^p всех возможных родительских пар решений. Полагаем $\eta = false$.

4) Выбираем из множества S^p очередную пару решений и применяем к этой паре используемую процедуру комбинирования опорных решений – получаем новое решение X'. Если $X' \notin S^r$ и $\varphi(X') < \varphi(X_1^r)$, то полагаем $X_1^r = X'$, помещаем это новое решение во множество S^r взамен старого худшего решения X_1^r и полагаем $\eta = true$. Повторяем указанные действия до исчерпания множества S^p.

5) Если $\eta = true$ (то есть хотя бы одна процедура комбинирования была успешной), то возвращаемся к шагу 3, в противном случае заканчиваем вычисления, принимая в качестве приближенного решения задачи лучшее решение множества S^r, то есть решение $X^r_{|S^r|}$.

Заметим, что в рассмотренной версии алгоритма *SS* эволюционирует популяция S^r, а не популяция S.

Методология рассеянного поиска очень гибка, поскольку каждый из ее элементов может быть реализован множеством способов разного уровня сложности. Во многом благодаря этому обстоятельству в настоящее время известно большое число версий алгоритма *SS*. Приведем в качестве примера схему одной из современных версий алгоритма.

Введем следующие обозначения: $T = \{T_j = (X_j, t_j), t_j \in [1:\hat{t}]\}$ – табу-множество (множество запрещенных решений), где t_j – номер итерации, на которой решение X_j было помещено в это множество. Мощность табу-множества, вообще говоря, может изменяться в процессе поиска.

1) Инициализируем популяцию $S = (s_i, i \in [1:|S|])$. Вычисляем начальные значения фитнес-функции для всех агентов популяции. На этой основе отбираем $|S^r|$ лучших агентов и помещаем их во множество S^r. Полагаем $\tilde{S} = S - S^r$, $T = Null$, $t = 0$.

2) Случайным образом выбираем $n \geq 2$ опорных решений $X^r_{j_1}, X^r_{j_2}, ..., X^r_{j_n}$ из набора S^r и помещаем их вместе с номером текущей итерации t в табу-множество T. В качестве величины n может быть использовано, например, число, случайно выбранное из некоторого набора натуральных чисел.

3) На основе выбранных опорных решений строим новое решение

$$X' = \sum_{k=1}^{n} \lambda_k X^r_{j_k},$$

где вес λ_k решения $X^r_{j_k}$ пропорционален относительному значению фитнес-функции $\varphi(X^r_{j_k})$, то есть

$$\lambda_k = \frac{\varphi(X^r_{j_k})}{\sum_{l=1}^{n} \varphi(X^r_{j_l})}.$$

Полагаем $t = t + 1$ и передаем решение X' *процедуре улучшения* (*improve procedure*), которая реализует поиск с запретами (п. 1.4.3).

4) Модифицируем множество $\tilde{S}$ по правилу

$$\tilde{S} = \tilde{S} \cup \{X_{j_k} \mid (X_{j_k}, t_k) \in T, t_k < t\}.$$

Другими словами, включаем во множество $\tilde{S}$ те из решений, содержащихся в табу-множестве, которые были помещены в него на предыдущих итерациях.

5) Случайным образом выбираем $|S^r|$ решений из множества $\tilde{S}$, а затем из этих решений выбираем n лучших. Исключаем последние решения из множества $\tilde{S}$ и добавляем во множество S^r.

6) Если $\varphi(X') < \varphi(\bar{X}_{j_w})$, то решение X' добавляем во множество $\tilde{S}$, а решение $\bar{X}_{j_w}$ исключаем из этого множества. Здесь $\bar{X}_{j_w}$ – худшее решение во множестве $\tilde{S}$:

$$\varphi(X_{j_w}) = \min_{s_j \in \tilde{S}} \varphi(X_j).$$

7) Если условия окончания итераций не выполнены, то переходим к шагу 2, в противном случае лучшее из найденных решений принимаем в качестве решения задачи.

В качестве условия окончания поиска в алгоритме *SS* (как и в других популяционных алгоритмах) может быть использовано падение текущего *разнообразия популяции* (*population variance*) ниже заданного уровня. Меры разнообразия популяции S могут строиться на основе оценки дисперсии соответствующих решений по каждому из $|X|$ измерений пространства поиска

$$v_j(S) = \frac{\sum_{i=1}^{|S|}\left(x_{i,j} - \bar{x}_j\right)^2}{|S|}, \quad \bar{x}_j = \frac{\sum_{i=1}^{|S|} x_{i,j}}{|S|}, \quad j \in [1:|X|].$$

В качестве указанной меры в этом случае может быть использована величина

$$\mu(S) = \max_{j \in [1:|S|]} v_j(S).$$

Условие окончания поиска на основе разнообразия популяции имеет вид

$$\mu(S) \le \varepsilon_\mu,$$

где ε_μ – заданная константа, рекомендованное значение которой равно 10^{-5}.

Если вернуться к сравнению алгоритма *SS* с генетическим алгоритмом, то следует отметить значительно меньший размер популяции опорных решений S^r по сравнению с размером популяции в последнем алгоритме: если в генетическом алгоритме этот размер равен ~100, то в алгоритме *SS* популяция S^r содержит всего лишь несколько агентов.

Данное обстоятельство обусловлено двумя следующими причинами. Во-первых, генетический алгоритм в силу используемых в нем механизмов эволюции популяции требует большого размера популяции для обеспечения желаемого уровня ее разнообразия. Во-вторых, в алгоритме *SS* комбинация большого числа агентов порождает множество решений большой мощности, так что его обработка требует высоких вычислительных затрат. В этой связи, как правило, в алгоритме *SS* имеют дело не более, чем с 20 комбинациями решений. Авторы алгоритма рекомендуют использовать $\frac{(3|S^r| - 7)|S^r|}{2}$ комбинаций следующих типов: все возможные комбинации двух из множества S^r решений; аналогичные комбинации трех решений и так далее до достижения указанного общего числа комбинаций.

Мощность множества S^r можно значительно увеличить, оставив число комбинаций приемлемым, с помощью следующего приема. По некоторому правилу разделяем это множество на несколько наборов $S_1^r, S_2^r, \ldots$ При формировании комбинаций решений допускаем объединение агентов только в пределах каждого из указанных наборов.

5.9.2. Алгоритм прокладки путей

Алгоритм *PR* был разработан с целью интегрировать интенсификационные и диверсификационные стратегии поиска в контексте алгоритма поиска с запретами. Основная идея алгоритма состоит в продуцировании новых решений на траекториях, которые соединяют найденные ранее пары высококачественных решений. Набор новых решений, называемый *пространством решений* (*solution space*), получают путем комбинирования решений этой пары по схеме алгоритма *SS*. Соответствующая траектория поиска начинается в одном из решений пары, которое называют *инициализирующим* (*initiating*) *решением*, и заканчивается в другом решении, называемом *целевым* (*guiding*) *решением*. Размер пространства решений растет экспоненциально с ростом различия решений в рассматриваемой паре. Алгоритм *PR* исследует небольшую часть этого пространства на основе *жадной рандомизированной стратегии*.

Основную идею алгоритма *PR* иллюстрирует рис. 5.22, на котором показаны два гипотетических пути $P_1(X', X''), P_2(X', X'')$, идущих от инициализирующего решения $X' \in D$ до целевого решения $X'' \in D$, принадлежащего некоторой достаточно малой окрестности решения X'. Путь P_2 (сплошная линия) показывает последовательность случайных перемещений, полученных традиционным способом (с помощью локальных улучшений значений фитнес-функции). Путь P_1 (пунктирная линия) построен алгоритмом *PR*. Путь приводит к лучшему решению $X^\bullet$ по сравнению с решениями X', X''.

Говоря более строго, алгоритм *PR* строит и исследует один или несколько путей $P_j(X', X''), j = 1, 2, \ldots$ с целью обнаружения решения $X^\bullet$ лучшего, чем решения X', X'':

$$\varphi(X^\bullet) = \min_j \varphi(X_j^\bullet), \ \varphi(X_j^\bullet) = \min_{X \in P_j(X', X'')} \varphi(X).$$

Алгоритм может включать в себя локальный поиск в окрестности решения $X^\bullet$.

Положим, что X – допустимое решение на пути $P(X', X'')$ и $d(X)$ – некоторая окрестность этого решения. В процессе прокладки пути не все возможные решения из множества $d(X)$ включают в путь $P(X', X'')$, но только решения, принадлежащие ограничению этого множества $d_T(X)$, которое представляет собой определенный набор комбинаций решений X и X''. Следующее за X решение пути $P(X', X'')$ получают путем выбора из множества $d_T(X)$ лучшего решения (жадная стратегия выбора).

Известно значительное число различных модификаций алгоритма *PR*. Перечислим основные из них. *Многородительский PR-алгоритм* использует не два, а более родительских агентов для построения путей. *Периодический PR-алгоритм* (*periodical PR*) предполагает не систематическую, а только периодическую прокладку путей. *Алгоритм прокладки путей вперед* (*forward PR*) и *алго-*

ритм прокладки путей назад (*backward PR*) используют в качестве инициализирующего решения худшее и лучшее решение из числа X', X'' соответственно. *Комбинированный алгоритм прокладки путей* (*back and forward PR*) строит две различные траектории, первая из которых исходит из решения X', а вторая – из решения X''. *Смешанный PR-алгоритм* (*mixed PR*) прокладывает два пути, идущих из решений X', X'', до тех пор, пока эти пути не образуют решение, равноудаленное от X', X'', то есть пока указанные пути не встретятся. *Рандомизированный PR-алгоритм* (*randomized PR*) в качестве следующего за X решения на пути $P(X', X'')$ выбирает не лучшее решение из множества $d_T(X)$, а случайное решение. Наконец, *усеченный PR-алгоритм* (*truncated PR*) исследует не всю траекторию $P(X', X'')$, а лишь некоторую ее часть.

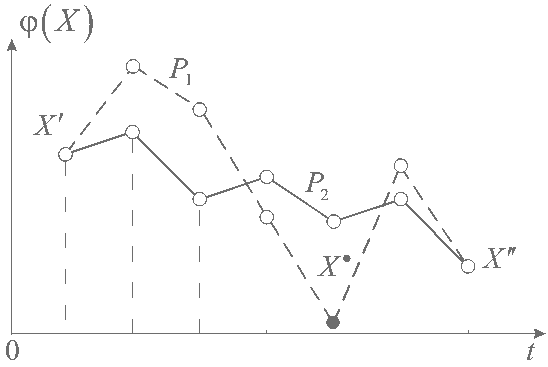


Рис. 5.22. К формированию траекторий в алгоритме *PR*

Вопросы для самопроверки

1. Представьте схему канонического алгоритма поиска гармонии. Определите основные операторы этого алгоритма.

2. Сформулируйте возможные направления модификации канонического алгоритма поиска гармонии.

3. Дайте схему детерминированного алгоритма оптимизации центральной силой, а также схему алгоритма гравитационного поиска как развитие первого из указанных алгоритмов.

4. Назовите основные подходы к модификации алгоритма гравитационного поиска.

5. Приведите схему канонического электромагнитного алгоритма оптимизации. Определите основные операторы этого алгоритма, а также возможные направления его модификации.

6. Представьте схему алгоритма поиска системой зарядов.

7. Дайте схему простого алгоритма эволюции разума. Определите основные операторы алгоритма.

8. Сформулируйте основные отличительные признаки расширенного, улучшенного, а также хаотического алгоритмов эволюции разума.

9. Назовите предпосылки алгоритма стохастического диффузионного поиска, представьте схему алгоритма и детализацию его основных операторов.

10. Определите основные понятия культурного алгоритма, приведите его общую схему, а также примеры операторов, обеспечивающих построение и коррекцию пространства убеждений.

11. Приведите примеры культурной модификации генетических операторов.

12. Дайте определение меметических алгоритмов оптимизации. Приведите общую схему меметического алгоритма. Определите основные гиперэвристики, используемые в гиперэвристических адаптивных мультимемных алгоритмах. Сформулируйте основные принципы построения самоадаптирующихся мультимемных алгоритмов.

13. Представьте схему самоорганизующегося миграционного алгоритма.

14. Сформулируйте основные отличительные черты алгоритмов рассеянного поиска и прокладки путей. Приведите схемы этих алгоритмов.

Глава 6

ГИБРИДИЗАЦИЯ ПОПУЛЯЦИОННЫХ АЛГОРИТМОВ

Опыт решения сложных прикладных задач, сводящихся к задаче глобальной оптимизации либо включающих в себя задачу глобальной оптимизации, показывает, что применение любого одного алгоритма оптимизации (как классического, так и популяционного) далеко не всегда приводит к успеху. В гибридных (комбинированных) алгоритмах, объединяющих различные либо одинаковые алгоритмы, но с различными значениями свободных параметров, преимущества одного алгоритма могут компенсировать недостатки другого. Поэтому одним из основных путей повышения эффективности решения задач глобального поиска в настоящее время является разработка гибридных популяционных алгоритмов. Иногда методы комбинирования метаэвристических алгоритмов называют *гиперэвристиками* (*hyper-heuristics*).

В этой главе представлены общие принципы гибридизации популяционных поисковых алгоритмов и рассмотрены основные методы гибридизации.

6.1. ОБЩИЕ ПРИНЦИПЫ ГИБРИДИЗАЦИИ

Можно предложить значительное число способов гибридизации поисковых алгоритмов оптимизации вообще и популяционных алгоритмов в частности, так что даже существует самостоятельная проблема классификации этих способов. Рассмотрим три следующие классификации: одноуровневая классификация Ванга (*X. Wang*), двухуровневая классификация Эль-Абда и Камэла (*El-Abd*, *M. Kamel*), четырехуровневая классификация Рейдла (*G. R. Raidl*).

6.1.1. Одноуровневая классификация Ванга

В соответствии с классификацией Ванга выделяют три категории гибридных алгоритмов: вложенные (*embedded*) алгоритмы, алгоритмы типа препроцессор/постпроцессор (*preprocessor/postprocessor*), коалгоритмы (*co-algorithms*).

В категории методов *гибридизации вложением* выделяем высокоуровневую и низкоуровневую гибридизации, как это сделано в четырехуровневой классификации Рейдла (см. ниже).

Высокоуровневая гибридизация вложением (*high-level embedded hybridization*) предполагает слабую связь объединяемых алгоритмов, обычно основанную на комбинировании популяционного алгоритма глобального поиска и того или иного алгоритма локального поиска (не обязательно популяционного). Отличительным признаком высокоуровневой гибридизации является то, что объединяемые алгоритмы сохраняют значительную автономию, так что

в гибридном алгоритме относительно легко выделить каждый из используемых алгоритмов. Примером высокоуровневой гибридизации вложением может служить меметический алгоритм (см. 5.7).

При *низкоуровневой* гибридизации вложением (*low-level embedded hybridization*) комбинируемые алгоритмы интегрированы настолько сильно, что выделить составляющие итогового алгоритма обычно невозможно. Другими словами, низкоуровневая гибридизация алгоритмов порождает, по сути, новый алгоритм. К низкоуровневой гибридизации вложением может быть отнесен, например, способ оптимизации значений свободных параметров популяционного алгоритма методом адаптивного управления параметрами (см. параграф 7.5).

Гибридизация типа *препроцессор/постпроцессор* применяется чаще всего. Обычно используют два алгоритма, функционирующих последовательно один за другим. Например, начальные итерации поиска реализует генетический алгоритм, параметры которого подобраны таким образом, чтобы обеспечить широкий обзор области поиска, а завершающие итерации – метод роя частиц, обеспечивающий точную локализацию экстремума, найденного генетическим алгоритмом. В общем случае данный тип гибридизации предполагает последовательное использование более двух алгоритмов.

Выделяем два класса методов гибридизации типа препроцессор/постпроцессор – последовательная гибридизация и конвейерная гибридизация.

Последовательная гибридизация предполагает фиксированный порядок использования двух объединяемых алгоритмов, хотя момент времени (номер поколения) переключения с одного алгоритма оптимизации на другой может априори не фиксироваться. В настоящее время известны примеры гибридизации между собой по данной схеме практически всех рассмотренных популяционных алгоритмов поисковой оптимизации.

Конвейерная гибридизация исходит из того, что более двух алгоритмов исполняются последовательно и очередность включения в работу алгоритмов определяется в процессе решения задачи с помощью некоторого адаптационного алгоритма. Примером конвейерной гибридизации является алгоритм бактериальной оптимизации *CBFO-S*, который использует комбинацию нескольких бактериальных алгоритмов, отличающихся величиной шага перемещения бактерий λ (см. 4.2.3).

Коалгоритмическая гибридизация алгоритмов предполагает, что объединяемые алгоритмы, как минимум, на логическом уровне выполняются параллельно, обмениваясь между собой в процессе поиска информацией. Примером такого способа гибридизации является генетический коэволюционный алгоритм (см. 2.12).

По четырехуровневой классификации Рейдла гибридизация вложением и гибридизация по схеме препроцессор/постпроцессор реализуют интегративную стратегию гибридизации, а коалгоритмическая гибридизация – кооперативную стратегию.

Отметим, что классификация Ванга не различает гибридные алгоритмы по порядку (последовательному или параллельному) выполнения входящих в них алгоритмов.

6.1.2. Двухуровневая классификация Эль-Абда и Камэла

Данная классификация использует две таксономии – на основе гибридизируемых алгоритмов и на основе декомпозиции пространства поиска (рис. 6.1).

Рис. 6.1. Двухуровневая классификация Эль-Абда – Камэла:
а – таксономия на основе гибридизируемых алгоритмов; *б* – таксономия на основе декомпозиции пространства поиска

Первая таксономия включает в себя композицию *гомогенных* (*homogeneous*) и *гетерогенных* (*heterogeneous*) алгоритмов. Те и другие алгоритмы могут быть использованы последовательно и параллельно. В первом случае различные алгоритмы функционируют последовательно на различных стадиях поиска, так что результаты поиска предыдущего алгоритма становятся исходными данными для последующего алгоритма (схема препроцессор/постпроцессор предыдущей классификации). Во втором случае каждый из алгоритмов реализует все стадии поиска с начала до конца, оптимизируя поиск за счет эпизодического получения от других алгоритмов текущих результатов (коалгоритмы в соответствии с рассмотренной одноуровневой классификацией).

Вторая таксономия предполагает совместный поиск на основе неявной, явной или комбинированной *декомпозиции пространства поиска*. *Неявная* (*implicit*) декомпозиция основана на том, что каждый из алгоритмов (гомогенных или гетерогенных) отыскивает решение в своей подобласти пространства решения, однако эти подобласти в явной форме не определены. В случае *явной* (*explicit*) декомпозиции указанные подобласти выделяют и назначают для поиска разным алгоритмам явным образом. *Комбинированная* декомпозиция использует оба указанных подхода.

6.1.3. Четырехуровневая классификация Рейдла

Классификация предложена в известном обзоре гибридных метаэвристических методов, выполненном Рейдлом. Классификация основана на следующих таксономиях: тип гибридизируемых алгоритмов, уровень гибридизации (*level of hybridization*), порядок выполнения (*order of execution*), стратегия управления (*control strategy*).

Таким образом, первым классификационным признаком, как и в рассмотренной выше двухуровневой классификации, является тип используемых алгоритмов (рис. 6.2). С этой точки зрения выделены классы гибридных алгоритмов на основе многоагентных алгоритмов оптимизации; на основе комбинации универсальных многоагентных алгоритмов оптимизации и проблемно-ориентированных алгоритмов; алгоритмы, комбинирующие многоагентные алгоритмы оптимизации и алгоритмы теории исследования операций или искусственного интеллекта. В последнем классе алгоритмов могут быть использованы, во-первых, точные алгоритмы, а во-вторых, алгоритмы на основе других эвристик или на основе методов *мягких вычислений* (*soft-computing methods*).

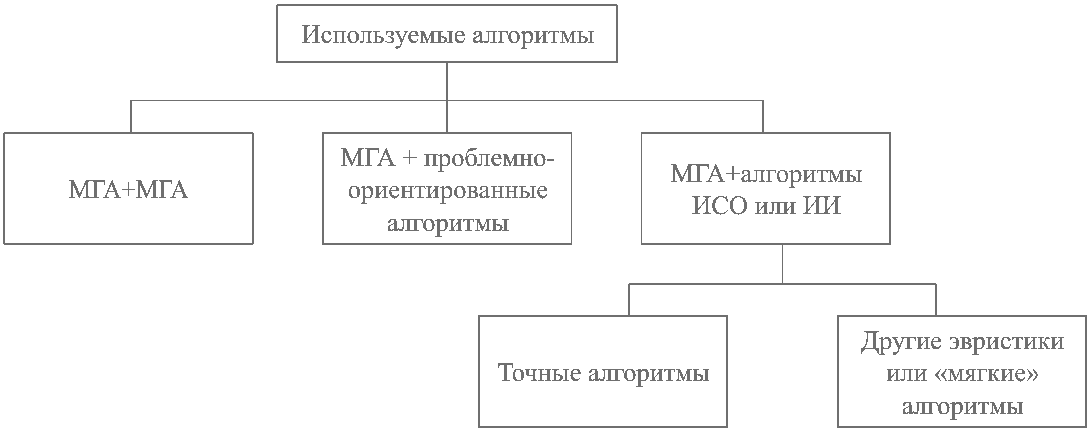


Рис. 6.2. Таксономия Рейдла на основе типа гибридизируемых алгоритмов:
МГА – многоагентный алгоритм; ИСО – исследование операций; ИИ – искусственный интеллект

Вторым классификационным признаком является уровень гибридизации комбинируемых алгоритмов (рис. 6.3). Данный классификационный признак нами использован выше в одноуровневой классификации Ванга.

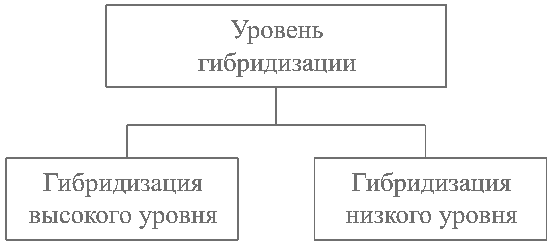


Рис. 6.3. Таксономия Рейдла на основе уровня гибридизации алгоритмов

В этой классификации выделяют *гибридизацию высокого* (*high-level*) и *низкого* (*low-level*) *уровней*. В первом случае комбинируемые алгоритмы сохраняют свои особенности и взаимодействуют между собой только путем обмена данными по четко определенным правилам. Прямые связи между отдельными фрагментами алгоритмов отсутствуют. В случае гибридизации низкого уровня между собой связаны отдельные компоненты или операторы алгоритмов.

Третий классификационный признак Рейдла – это *порядок выполнения* комбинируемых алгоритмов (рис. 6.4). В соответствии с порядком выполнения, во-первых, выделяют *последовательную* (*sequential*) гибридизацию, в которой комбинируемые алгоритмы выполняются последовательно и информация передается в одном направлении – от предыдущего алгоритма к последующему. Предварительная обработка полученных данных при этом не производится. Во-вторых, имеет место параллельный порядок выполнения алгоритмов, которому соответствует *параллельная* (*parallel*) гибридизация. Подчеркнем еще раз, что при этом идет речь о параллелизме на логическом уровне, который не предполагает обязательную физически параллельную реализацию комбинируемых алгоритмов. В-третьих, различают *чередующуюся* (*interleaved*) гибридизацию.

Рис. 6.4. Таксономия Рейдла на основе порядка выполнения алгоритмов

Наконец, четвертый классификационный признак представляет собой *стратегию управления* объединяемыми алгоритмами (рис. 6.5). С этой точки зрения выделяют интегративный и кооперативный подходы к гибридизации.

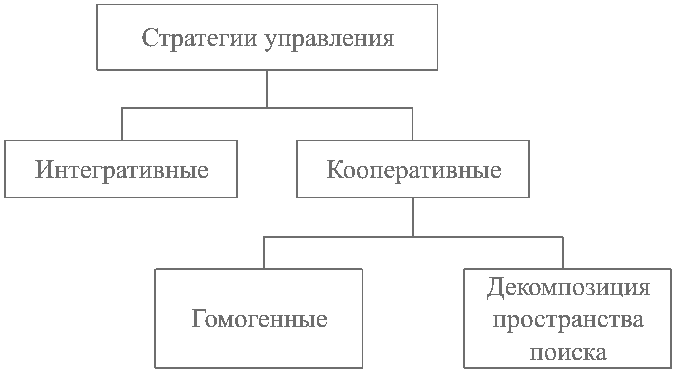


Рис. 6.5. Таксономия Рейдла на основе стратегии управления алгоритмами

При использовании *интегративной* (*integrative*) стратегии управления один из алгоритмов считают зависимым, вложенным компонентом другого алгоритма. Примерами интегративного управления комбинируемыми алгоритмами являются непрерывный ортогональный алгоритм муравьиной колонии (см. 3.2.4), гибридный алгоритм непрерывно взаимодействующей муравьиной колонии (3.2.5), комбинация модифицированного алгоритма колонии искусственных пчел и алгоритма имитации отжига (3.3.4).

Кооперативная (*collaborative*) стратегия управления предполагает, что гибридизируемые алгоритмы обмениваются между собой информацией, но не являются частью друг друга. Ярким примером данного подхода является известная островная модель взаимодействия популяций в многопопуляционных композициях алгоритмов.

Используем далее классификацию Ванга, расширенную рассмотренным выше образом.

6.2. ВЛОЖЕННЫЕ АЛГОРИТМЫ

Как отмечалось выше, различают высокоуровневую и низкоуровневую гибридизации вложением. Данный параграф представляет методики этих видов гибридизации.

6.2.1. Высокоуровневая гибридизация вложением

Выделим два класса методов высокоуровневой гибридизации вложением – последовательная гибридизация и гибридизация путем декомпозиции пространства поиска.

Общая схема *последовательной* высокоуровневой гибридизации вложением может быть представлена в следующем виде.

1) Инициализируем агентов s_i, $i \in [1:|S|]$ используемого популяционного алгоритма.

2) Выполняем фиксированное число итераций (обычно одну) популяционного алгоритма.

3) Исходя из текущих положений координат агентов X_i, $i \in [1:|S|]$, выполняем локальный поиск с помощью второго гибридизируемого алгоритма. Координаты лучших найденных точек X_i' полагаем равными текущим координатам соответствующих агентов.

4) Проверяем выполнение условия окончания итераций. Если это условие выполнено, завершаем вычисления, в противном случае переходим к шагу 2.

Как неоднократно отмечалось выше, в популяционных алгоритмах глобальной оптимизации сложно совместить требования диверсификации и интенсификации поиска. С самой общей точки зрения, высокоуровневая гибридизация вложением реализует первое из указанных требований с помощью основного (популяционного) алгоритма, а второе требование – при помощи вспомогательного (популяционного или иного) алгоритма локального поиска.

В настоящее время известны примеры высокоуровневой гибридизации вложением практически всех рассмотренных популяционных алгоритмов

поисковой оптимизации со многими классическими и неклассическими алгоритмами локальной оптимизации. В качестве примеров укажем на гибридизацию алгоритма муравьиной колонии с алгоритмом на основе планов первого порядка (см. 3.2.4), алгоритма непрерывно взаимодействующей муравьиной колонии с алгоритмом Нелдера – Мида (3.2.5).

Поскольку в рассматриваемой схеме гибридизации основной задачей первого алгоритма является обеспечение широкого исследования пространства поиска, целесообразным может быть использование средств борьбы с нишеванием, то есть использование модификаций этого алгоритма, обеспечивающих принудительное удаление агентов в популяции друг от друга (см. 2.6). Чаще всего для преодоления нишевания близких агентов объединяют в кластеры и приспособленности кластеризованных агентов снижают путем, например, деления их приспособленности на число агентов в кластере. Тем самым повышается относительная приспособленность агентов, не принадлежащих кластерам, и стимулируется удаление решений друг от друга. В качестве недостатка этого подхода можно отметить появление в модифицированном популяционном алгоритме еще одного свободного параметра, который называется диаметром кластера и имеет смысл максимального расстояния между агентами, принадлежащими одному кластеру.

Если вести речь о распараллеливании популяционных алгоритмов оптимизации для *MIMD*-вычислительной системы (см. 9.1), то на этапах локального поиска гибридный алгоритм, построенный по схеме высокоуровневой гибридизации вложением, может быть эффективно распараллелен путем решения каждой из задач локальной оптимизации на своем процессоре.

Высокоуровневая гибридизация вложением на основе *декомпозиции пространства поиска* обычно использует разложение пространства поиска $R^{|X|}$ на $\left\lceil \frac{|X|}{2} \right\rceil$ подпространств, каждое из которых (за исключением, быть может, лишь последнего) имеет размерность, равную двум. Для простоты записи примем, что величина $|X|$ является четной. В этом допущении общая схема гибридного алгоритма данного типа имеет следующий вид.

Шаг 0. Инициализируем мультипопуляцию

$$\boldsymbol{S} = \left\{ S_i, i \in \left[1 : \frac{|X|}{2} \right] \right\},$$

где $\frac{|X|}{2} = |\boldsymbol{S}|$ – число субпопуляций.

Шаг 1. Выполняем с помощью популяции S_1 поиск новых значений компонент x_1, x_2 вектора X, используя в качестве значений остальных компонент их значения, полученные на предыдущей итерации.

Шаг 2. Выполняем с помощью популяции S_2 поиск новых значений компонент x_3, x_4 вектора X, используя в качестве значений компонент x_1, x_2 значения этих величин x_1', x_2', найденные на шаге 1, а в качестве остальных компонент – их значения, полученные на предыдущей итерации.

И так далее.

Шаг $\frac{|X|}{2}$. Выполняем с помощью популяции $S_{|S|}$ поиск новых значений компонент $x_{|X|-1}, x_{|X|}$, используя в качестве значений компонент $x_1, x_2, ..., x_{|X|-2}$ их значения $x'_1, x'_2, ..., x'_{|X|-2}$, найденные на предыдущих шагах поиска.

Шаг $\frac{|X|}{2}+1$. Если условия окончания поиска не выполнены, то возвращаемся к шагу 1, в противном случае завершаем поиск, принимая в качестве решения последние из найденных значений компонент вектора *X*.

Примером гибридного алгоритма, использующего рассматриваемую схему гибридизации вложением, является алгоритм бактериальной оптимизации *CBFO-H* (см. 4.2.3).

6.2.2. Низкоуровневая гибридизация вложением

Как отмечалось выше, низкоуровневая гибридизация вложением предполагает сращивание гибридизируемых алгоритмов так, что, по сути, ее результатом является новый алгоритм, который не может быть декомпозирован на составляющие его алгоритмы. Общая методика низкоуровневой гибридизации вложением отсутствует. Можно лишь констатировать, что основной идеей такой гибридизации является модификация миграционных операторов первого (популяционного) алгоритма на основе другого или других гибридизируемых алгоритмов. Приведем несколько примеров, иллюстрирующих многообразие возможных подходов к такой гибридизации.

• Генетический алгоритм и алгоритм имитации отжига. В таком гибридном алгоритме может быть использована модификация оператора селекции путем встраивания в него алгоритма имитации отжига так, чтобы обеспечить некоторую вероятность включения худшего индивида в следующее поколение.

• Генетический алгоритм и локальный алгоритм оптимизации. Низкоуровневая гибридизация вложением в этом случае может состоять в замене операции мутации несколькими шагами локальной оптимизации.

• Алгоритм роя пчел и алгоритм имитации отжига. В данной комбинации алгоритмов с помощью алгоритма имитации отжига может быть реализована операция выбора лучшей пчелы.

• Алгоритм бактериальной оптимизации и алгоритм роя частиц. Низкоуровневая гибридизация вложением указанных алгоритмов возможна, к примеру, путем реализации операции хемотаксиса с помощью одной или нескольких итераций алгоритма роя частиц.

• Алгоритм роя частиц и генетический алгоритм. При гибридизации этих алгоритмов каждой из частиц роя можно поставить в соответствие хромосому, число генов в которой равно размерности вектора координат частицы (то есть размерности вектора варьируемых параметров рассматриваемой задачи оптимизации). Хромосомы в таком гибридном алгоритме эволюционируют как по правилам движения частиц в рое, так и в соответствии с генетическими операторами.

• Генетический алгоритм и культурный алгоритм. Возможные подходы к культурной модификации генетических операторов представлены в 5.6.

6.3. ГИБРИДИЗАЦИЯ ПО СХЕМЕ ПРЕПРОЦЕССОР / ПОСТПРОЦЕССОР

Как последовательная, так и конвейерная гибридизация по схеме препроцессор/постпроцессор требуют решения следующих основных задач: выбор гибридизируемых алгоритмов; реализация интерфейса между алгоритмами; разработка правила определения моментов времени (поколений), в которые должно производиться переключение с одного алгоритма на другой; определение закона применения гибридизируемых алгоритмов.

6.3.1. Последовательная гибридизация

Повторим основной аргумент в пользу последовательной гибридизации. Популяционные алгоритмы глобальной оптимизации на первых этапах поиска обычно обеспечивают широкий обзор пространства поиска и высокую вероятность отыскания некоторых окрестностей локальных минимумов фитнес-функции. Однако при решении задачи локализации этих минимумов на завершающих этапах поиска популяционные алгоритмы требуют неоправданно большого числа испытаний. В то же время алгоритмы локальной оптимизации решают последнюю задачу значительно более эффективно. Последовательная гибридизация имеет целью объединить указанные преимущества популяционных алгоритмов и алгоритмов локального поиска.

Комбинированные алгоритмы, полученные путем последовательной гибридизации, можно рассматривать как хорошую альтернативу алгоритму мультистарта. Известным недостатком этого алгоритма оказывается необходимость генерации большого числа стартовых точек, для того чтобы обеспечить достаточную вероятность локализации глобального экстремума. Как следствие, в процессе поиска значительное число итерационных процессов, стартовавших из указанных точек, обычно сходится к одним и тем же точкам локального либо глобального экстремума, так что большая часть вычислительных затрат на поиск оказывается излишней.

Выбор гибридизируемых алгоритмов

Последовательная гибридизация использует, как правило, в качестве первого популяционный алгоритм, а в качестве второго – тот или иной алгоритм локальной оптимизации. Чаще других с этой целью применяют алгоритм имитации отжига (см. 1.4.2), который призван обеспечить выход процесса поиска из мелких экстремумов, возможно, лежащих на пути к более глубокому экстремуму. Иногда алгоритм имитации отжига комбинируют с каким-либо другим алгоритмом локальной оптимизации.

Интерфейс между алгоритмами

Поскольку при достаточно большом числе итераций все популяционные алгоритмы с высокой вероятностью обеспечивают сходимость к глобальному экстремуму, по первому впечатлению в качестве стартовой точки для локального алгоритма оптимизации достаточно использовать только точку X_{i_b}, соответ-

ствующую лучшему значению фитнес-функции, достигнутому популяцией. Однако в связи с тем, что рассматриваемый способ гибридизации предполагает достаточно раннее прерывание итераций популяционного алгоритма и переход к фазе локального поиска, выбор одной точки может существенно повысить вероятность «потери» глобального экстремума. Поэтому в качестве стартовых рекомендуется выбирать не только лучшую, но и некоторое число других точек. Обозначим общее число таких точек n_{start} (свободный параметр).

Пусть переключение алгоритмов производится после завершения t_n итераций используемого популяционного алгоритма. Координаты особей популяции в момент времени t_n обозначим $X_i = X_i(t_n)$, а соответствующие значения фитнес-функции – $\varphi_i = \varphi_i(t_n)$; $i \in [1:|S|]$. Положим, что речь идет о минимизации этой функции.

Возможны несколько способов выбора стартовых точек. Очевидным является выбор всех $|S|$ точек ($n_{start} = |S|$). Такой способ отбора является наиболее надежным с точки зрения вероятности локализации глобального экстремума, но может потребовать неприемлемо больших вычислительных затрат.

Естественным также является выбор только лучших точек по следующей схеме:

– сортируем величины φ_i по возрастанию, так что $\varphi_{i_1} \le \varphi_{i_2} \le \ldots \le \varphi_{i_{|S|}}$;

– в качестве стартовых точек используем точки с координатами X_{i_j}, $j \in [1:n_{start}]$.

Среди отобранных по этой схеме точек могут оказаться близкие, то есть находящиеся в области «притяжения» одного и того же экстремума. Поэтому возможен отбор по схеме, близкой к рассмотренной, но с предварительным удалением из последовательности X_i, $i \in [1:|S|]$ точек, удовлетворяющих условию $\|X_i - X_j\| \le \varepsilon$, $i, j \in [1:|S|]$, $i \ne j$, где ε – минимально допустимое расстояние между стартовыми точками (свободный параметр).

Поскольку, как отмечалось выше, переход от этапа глобального поиска к этапу локального поиска может выполняться достаточно рано (задолго до начала стагнации вычислительного процесса), тот факт, что, к примеру, $\varphi_{i_j} < \varphi_{i_k}$, не всегда означает большую близость к глобальному минимуму точки X_{i_j} по сравнению с точкой X_{i_k}. Поэтому можно предложить отбор еще и по следующей схеме:

– выбираем по рассмотренному правилу $n'_{start} < n_{start}$ стартовых точек;

– остальные $(n_{start} - n'_{start})$ точек случайно и равновероятно выбираем из числа оставшихся точек. Здесь n'_{start} – свободный параметр.

На основе тех же соображений возможен способ отбора, заключающийся в случайном равновероятном выборе всех n_{start} стартовых точек.

Наконец, возможен отбор на основе специально созданного архива, содержащего координаты n_{start} лучших точек, полученных на протяжении всех t_n поколений реализации поискового алгоритма.

Момент времени переключения алгоритмов

На начальных итерациях особи популяционного алгоритма широко распределены в области поиска, но, как правило, находятся далеко от решения задачи. Напротив, на завершающих итерациях особи сосредоточены в окрест-

ности найденного экстремума, однако этот экстремум не гарантированно является глобальным. Поэтому в качестве верхней границы момента времени t_n целесообразно использовать начало стагнации вычислительного процесса. В простейшем случае в качестве момента времени t_n используют априори заданное число поколений, гарантированно меньшее числа итераций, при котором начинается стагнация.

Существенно более эффективным может быть адаптивное определение этого числа на основе одного из условий

$$v(t_n) \le v^{\min},\ \dot{v}(t_n) \le \dot{v}^{\min},$$

где $v(t_n)$, $\dot{v}(t_n)$ – оценки скорости и ускорения сходимости итераций в момент времени t_n соответственно; $v^{\min}$, $\dot{v}^{\min}$ – их минимально допустимые значения (свободные параметры). Имея в виду стохастический характер процесса сходимости популяционных алгоритмов, в качестве указанных оценок целесообразно использовать сглаженные величины вида

$$v(t) = \frac{\overline{\varphi}(t) - \overline{\varphi}(t-m)}{m}, \tag{6.1}$$

где

$$\overline{\varphi}(t) = \frac{1}{m}\sum_{\tau=1}^{m}\varphi(t-m+\tau),\ \overline{\varphi}(t-m) = \frac{1}{m}\sum_{\tau=1}^{m}\varphi(t-2m+\tau)\ -$$

средние значения фитнес-функции на итерациях $[(t-m):t]$, $[(t-2m):(t-m)]$ соответственно; $(t-2m) \ge 0$, $t < \hat{t}$, m – целое положительное число (свободный параметр).

Закон применения гибридизируемых алгоритмов

Поскольку последовательная гибридизация предполагает однократное последовательное использование только двух алгоритмов, проблема определения данного закона в этом случае отсутствует.

6.3.2. Конвейерная гибридизация

Напомним, что конвейерная гибридизация предполагает последовательное выполнение более двух поисковых алгоритмов оптимизации.

Выбор гибридизируемых алгоритмов

Данный выбор в значительной мере ограничен требованием простоты интерфейса между объединяемыми алгоритмами. Поэтому при конвейерной гибридизации обычно используют гомогенные, а не гетерогенные алгоритмы, хотя известны примеры такой гибридизации по схеме: популяционный алгоритм + алгоритм имитации отжига + какой-либо алгоритм локального поиска.

Реализация интерфейса между алгоритмами

В зависимости от принципов реализации интерфейса между конвейеризуемыми алгоритмами эти алгоритмы целесообразно разделить на два класса:

класс *нестигмергичных алгоритмов*, которые не используют для обмена информацией между агентами канал стигмергии; класс *стигмергичных алгоритмов*, использующих с этой целью канал стигмергии (возможно, наряду с другими каналами).

• Гибридизация нестигмергичных алгоритмов. Состояние агента в нестигмергичном алгоритме, как правило, определяют его текущие координаты и, быть может, скорость. Поэтому, если опустить детали, реализация интерфейса между нестигмергичными алгоритмами сводится к тому, что при инициализации новой популяции конечные координаты агентов предыдущей популяции используют в качестве начальных координат агентов новой популяции. Аналогично, если алгоритмы обеих популяций используют скорости агентов, то члены новой популяции получают конечные скорости соответствующих агентов предыдущей популяции.

Формализуем приведенные соображения. Не ограничивая общности, положим, что на итерациях $[t_{n-1}:t_n]$, $(t_n+1:\ t_{n+1}]$ используются нестигмергичные алгоритмы a_1, a_2 соответственно; $0 \le t_{n-1} < t_n < t_{n+1} \le \hat{t}$. Пусть эти алгоритмы реализуют популяции $S_1 = (s_{1,i},\ i \in [1:|S_1|])$, $S_2 = (s_{2,i},\ i \in [1:|S_2|])$, где $|S_2| = |S_1| = |S|$.

В указанных обозначениях и соглашениях конвейерная гибридизация алгоритмов a_1, a_2 требует после выполнения итерации t_n лишь инициализировать популяцию S_2 путем присваиваний

$$X_{2,i}(t_n) = X_{1,i}(t_n),\ V_{2,i}(t_n) = V_{1,i}(t_n)\ ,\ i \in [1:|S|],$$

где $X_{1,i}, X_{2,i}$ – векторы координат агентов $s_{1,i}, s_{2,i}$, а $V_{1,i}, V_{2,i}$ – векторы их скоростей соответственно. Если алгоритмы a_1, a_2 не имеют памяти, то их гибридизация сводится к первому из указанных присваиваний.

В случае, когда алгоритм a_1 не имеет памяти, а алгоритм a_2 имеет ее, инициализация популяции S_2 требует задания еще и значений компонент векторов скоростей ее агентов $s_{2,i}$, $i \in [1:|S|]$, например, по одной из следующих схем:

$$V_{2,i}(t_n) = N_{|X|}(m, \sigma);$$

$$V_{2,i}(t_n) = U_{|X|}[a; b];$$

$$V_{2,i}(t_n) = \left(X_{1,i}(t_n) - X_{1,i}(t_n - 1)\right) + N_{|X|}(m, \sigma).$$

Здесь m, σ, a, b – заданные вещественные константы. Поскольку изменение координат агентов во времени носит стохастический характер, в последнем случае может быть целесообразным использование вместо векторов $X_{1,i}(t_n)$, $X_{1,i}(t_n - 1)$ соответствующих векторов, осредненных по нескольким предыдущим итерациям.

Если, наконец, агента $s_{1,i}$ определяют векторы $X_{1,i}$, $V_{1,i}$, а агента $s_{2,i}$ – только вектор $X_{2,i}$, то при инициализации популяции S_2 теряется информация, заключенная в векторе $V_{1,i}$; $i \in [1:|S|]$.

При гибридизации нестигмергичных гетерогенных алгоритмов возможны и другие источники потери информации. Например, если a_1, a_2 – алгоритмы пчелиного роя и роя частиц соответственно, то при их гибридизации теряется информация об элитных и перспективных участках.

• Гибридизация стигмергичных алгоритмов. Метки в пространстве поиска, которые оставляют в этом случае агенты, по сути, формируют модель фитнес-функции решаемой задачи поисковой оптимизации. Поэтому основной проблемой организации интерфейса между стигмергичными алгоритмами является проблема преобразования этих моделей. Общие рекомендации по такому преобразованию отсутствуют.

• Гибридизация стигмергичных и нестигмергичных алгоритмов. Если алгоритмы a_1, a_2 относятся к различным классам, то их комбинирование приводит либо к потере информации о модели фитнес-функции (если алгоритм a_1 является стигмергичным), либо требует увеличения в популяции S_2 числа индивидов по сравнению с числом индивидов в популяции S_1 (если алгоритм a_1 является нестигмергичным). Например, если речь идет об объединении алгоритма a_1 колонии муравьев и алгоритма a_2 роя частиц, то новых агентов последнего алгоритма целесообразно инициализировать в областях с высокой интенсивностью феромонного следа.

Моменты времени переключения алгоритмов

В простейшем случае моменты времени, в которые должно производиться переключение с одного гибридизируемого алгоритма на другой, могут быть заданы пользователем априори. Например, эти моменты может определять следующее правило: в течение $[0:t_1)$ итераций должен использоваться первый из гибридизируемых алгоритмов (например, алгоритм a_1); в течение итераций $[t_1:t_2)$ – алгоритм a_2 и так далее до последнего периода итераций $[t_{\hat{n}}:\hat{t}]$, на котором должен быть использован алгоритм $a_{\hat{n}}$. Здесь t_n – заданный номер итерации; $n \in [1:\hat{n}]$. Среди алгоритмов $a_1, a_2, ..., a_{\hat{n}}$ могут быть совпадающие.

Более сложным является вариант *адаптивного управления переключениями алгоритмов*. При этом могут быть использованы, например, следующие условия переключения: стагнация вычислительного процесса, замедление скорости сходимости ниже допустимого порога, комбинация указанных условий.

При использовании первого из этих условий переключение алгоритмов производим на тех итерациях, на которых разность двух последовательных лучших значений фитнес-функции меньше заданной величины ε_n в течение δ_t итераций.

Для оценки скорости сходимости алгоритма $v(t)$ могут быть использованы формулы вида (6.1). В момент времени t_n переход от текущего используемого алгоритма a_n к алгоритму a_{n+1} выполняем в случае, если $v(t_n) \le \varepsilon_v$. Величина ε_v (свободный параметр) должна по некоторому закону убывать с ростом номера итераций.

Закон переключения алгоритмов

Общая теория адаптивной конвейерной гибридизации алгоритмов поисковой оптимизации предложена в работах Л. С. Растригина и названа *теорией альтернативной оптимизации* (*alternative adaptation*). Рассмотрим основы указанной теории.

Пусть $A = \{a_1, a_2, ..., a_{|A|}\}$ – совокупность алгоритмов поисковой оптимизации (альтернатив), с помощью каждого из которых может быть решена рассмат-

риваемая задача глобальной оптимизации. Положим, что гибридизированы могут быть любые из алгоритмов $a_1, a_2, ..., a_{|A|}$, то есть, что определен интерфейс между любой парой указанных алгоритмов. Без потери общности рассуждений примем, что в процессе решения задачи в течение итераций $[t_{n-1} : t_n)$ используется алгоритм a_n; $n \in [1:\hat{n}]$, $t_0 = 0$, $t_{\hat{n}} = \hat{t}$.

К моменту выполнения рассматриваемой итерации t_n выполнены испытания в некоторой совокупности точек $\boldsymbol{X}_n$ и вычислены соответствующие значения целевой функции $f(X)$ и ограничивающих функций $E(X), G(X)$, образующие множества $\boldsymbol{f}_n$, $\boldsymbol{E}_n$, $\boldsymbol{G}_n$.

В общей постановке задача альтернативной адаптации состоит в отыскании алгоритма

$$a_{n+1} = R\left(\boldsymbol{X}_n, \boldsymbol{f}_n, \boldsymbol{E}_n, \boldsymbol{G}_n, \boldsymbol{A}_n, \boldsymbol{W}_n\right),$$

который должен быть использован на итерациях $[t_n + 1 : t_{n+1})$. Здесь $R(\cdot)$ – правило выбора алгоритма a_{n+1}; $\boldsymbol{A}_n = \{a_k \mid a_k \in A,\ k \in [1:n]\}$ – набор алгоритмов, не обязательно различных, которые были использованы на всех предыдущих интервалах адаптации; $\boldsymbol{W}_n$ – совокупность оценок критерия качества адаптации $w(a)$ на итерациях $[0 : t_n]$. Полагаем, что критерий $w(a)$ подлежит максимизации.

Можно предложить значительное число критериев $w(a)$. При этом на различных этапах поиска могут использоваться различные критерии. Так, на начальных этапах целесообразно использовать критерий, имеющий смысл скорости или ускорения поиска вида (6.1). На завершающих этапах поиска, когда повышения скорости или ускорения поиска ожидать трудно, в качестве критерия качества адаптации представляется разумным использовать приращение фитнес-функции, достигнутое на итерациях $[t_{n-1} : t_n)$.

Ввиду конечности числа альтернатив $A = \{a_l, l \in [1:|A|]\}$ *алгоритм адаптации R* может быть построен в виде конечного автомата

$$B = \langle Q, C, A, \theta, \vartheta \rangle,$$

где $Q = \{q_j, j \in [1:|Q|]\}$ – множество состояний автомата, $C = \{c_k, k \in [1:|C|]\}$ – входной алфавит, A – выходной алфавит, θ – функция переходов автомата из одного состояния другое, ϑ – функция выходов.

Функция θ определяет следующее состояние автомата $q_j \in Q$ на основе его текущего состояния $q_i \in Q$ и входа $c_k \in C$, так что

$$q_j = \theta(q_i, c_k),\ i, j \in [1:|Q|],\ k \in [1:|C|].$$

Аналогично функция ϑ определяет выход автомата $a_l \in A$ по его текущему состоянию q_i и входу c_k:

$$a_l = \vartheta(q_i, c_k),\ i \in [1:|Q|],\ k \in [1:|C|],\ l \in [1:|A|].$$

Различные множества состояний автомата Q, функций перехода θ и выхода ϑ порождают различные алгоритмы адаптации. Так, детерминированные функции θ, ϑ дают детерминированный автомат и соответствующий алгоритм

адаптации, а те же стохастические функции – стохастический автомат и алгоритм адаптации.

В простейшем случае используют бинарный автомат, входной алфавит которого состоит всего из двух символов $c_1 = 0$ (не штраф), $c_2 = 1$ (штраф). При этом первое присваивание выполняют в том случае, когда в течение итераций $[t_{n-1} : t_n)$ имело место увеличение оценки критерия w, а второе присваивание – когда имело место уменьшение этой оценки:

$$c = \begin{cases} 0, & \Delta w_n > 0, \\ 1, & \Delta w_n \le 0, \end{cases} \quad \Delta w_n = w_n - w_{n-1}.$$

Поскольку все рассматриваемые популяционные алгоритмы являются стохастическими, оценки критерия качества являются сильно зашумленными. Поэтому при вычислении величины Δw_n используют то или иное сглаживание указанных оценок.

• Детерминированный алгоритм адаптации. Простейший вариант такого алгоритма реализует так называемый бинарный *детерминированный автомат с целесообразным поведением*. Поведение бинарного автомата называют целесообразным, если в процессе его функционирования среднее значение $\overline{c}$ величины штрафа удовлетворяет неравенству $\overline{c} < \frac{1}{2}$, или, другими словами, если этот автомат за свои действия чаще поощряется ($c = 0$), чем штрафуется ($c = 1$). Такая ситуация имеет место, очевидно, когда оценка приращения критерия качества адаптации Δw_n чаще положительна, чем отрицательна.

Известно большое число бинарных детерминированных автоматов с целесообразным поведением. С точки зрения решения задачи альтернативной адаптации наибольший интерес представляют *автоматы с линейной тактикой*. Такой автомат функционирует по следующему правилу. Если в течение итерации $[t_{n-1} : t_n)$ использовался алгоритм a_n и имела место ситуация $\Delta w_n > 0$, то на следующем периоде итераций $[t_n + 1 : t_{n+1})$ используем тот же алгоритм a_n, то есть принимаем $a_{n+1} = a_n$. В противном случае автомат выбирает алгоритм, следующий за алгоритмом a_n в совокупности алгоритмов A. Если $a_n = a_{|A|}$, то полагаем $a_{n+1} = a_1$. Схему функционирования детерминированного автомата с линейной тактикой для случая, когда множество алгоритмов A содержит только два алгоритма, иллюстрирует рис. 6.6, *а*.

• Стохастический алгоритм адаптации может быть реализован с помощью *стохастического автомата с линейной тактикой*, который производит независимый выбор алгоритма a_i с вероятностью ξ_i,

$$\sum_{i=1}^{|A|} \xi_i = 1. \tag{6.2}$$

Обучение такого автомата сводится к изменению вероятностей ξ_i на каждом шаге принятия решения таким образом, чтобы при выполнении условия (6.2) вероятность выбора алгоритма a_i при нештрафе увеличивалась, а при штрафе – уменьшалась:

$$\xi_i(t_n) = \xi_i(t_{n-1}) + \Delta\xi_i(t_n); \tag{6.3}$$

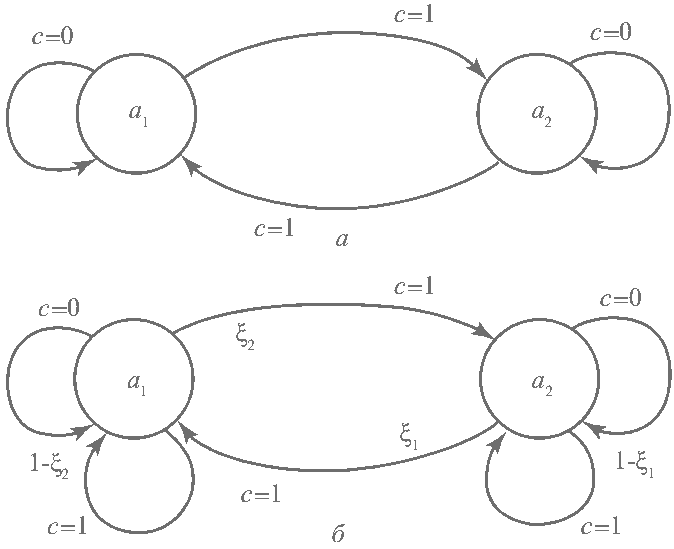


Рис. 6.6. Схемы автоматной альтернативной адаптации, $|A| = 2$: *а* – детерминированный автомат с линейной тактикой; *б* – стохастический автомат с линейной тактикой

$$\Delta\xi_i(t_n) = \begin{cases} p(\xi_i(t_{n-1})), & a_{t_n} = a_i, \\ r(\xi_i(t_{n-1}), \xi_j(t_{n-1})), & a_{t_n} = a_j,\ j \neq i. \end{cases} \tag{6.4}$$

Здесь $p(\cdot), r(\cdot,\cdot)$ – функции, удовлетворяющие условиям

$$\begin{cases} p(\cdot) > 0, \quad r(\cdot,\cdot) < 0, \quad c = 0, \\ p(\cdot) < 0, \quad r(\cdot,\cdot) > 0, \quad c = 1. \end{cases} \tag{6.5}$$

Можно показать, что стратегию обучения (6.2)–(6.5) реализуют, например, функции

$$p(\xi) = \begin{cases} \alpha(1-\xi)^\beta, & c = 0, \\ -\alpha\xi^\beta, & c = 1, \end{cases}$$

$$r(\xi_i, \xi_j) = \begin{cases} -\alpha\dfrac{\xi_j^\beta(1-\xi_i)^\gamma}{\Sigma_1(\xi_j)}, & c = 0, \\ \alpha\dfrac{(1-\xi_j)^\beta\xi_i^\gamma}{\Sigma_2(\xi_j)}, & c = 1, \end{cases}$$

где

$$\Sigma_1(\xi_j) = \sum_{k=1,\,k\neq j}^{|A|} (1-\xi_k)^\gamma,\ \Sigma_2(\xi_j) = \sum_{k=1,\,k\neq j}^{|A|} \xi_k^\gamma.$$

Здесь α, β, γ – свободные параметры, удовлетворяющие неравенствам $0 < \alpha < 1$, $\beta > 0$, $\gamma > 0$. В простейшем случае можно положить $\beta = \gamma = 1$.

Функционирование стохастического автомата с линейной тактикой в случае, когда $|A| = 2$, иллюстрирует рис. 6.6, *б*.

В более общей постановке задачу альтернативной адаптации рассматривают как задачу многокритериальной оптимизации. Например, наряду с рассмотренными критериями качества адаптации в виде скорости или ускорения сходимости итерационного процесса оптимизации используют критерии, формализующие объем исследованной части области поиска, равномерность покрытия этой части и т. д. Для решения указанной задачи многокритериальной оптимизации естественно использовать метод сведения этой задачи к однокритериальной путем сворачивания векторного критерия в скалярный с помощью аддитивной или мультипликативной свертки (параграф 8.1).

6.4. КОАЛГОРИТМЫ

Коалгоритмы можно интерпретировать как модель коэволюционирующих систем. Коэволюцией систем называют ситуацию, когда в процессе своей эволюции система *A* начинает тем или иным образом влиять на эволюцию системы *B*, а система *B* посредством механизма адаптации начинает приспосабливаться к изменениям системы *A*, и наоборот. В результате формируется общий темп эволюции этих систем и, по сути, новая система (метасистема), объединяющая коэволюционирующие системы.

Известно, что процесс коэволюции может многократно ускорять процесс эволюции каждой из коэволюционирующих систем. Если эти системы обладают ресурсами примерно одинаковой величины, то и ускорение темпа эволюции в них будет примерно одинаковым. В случае взаимодействия систем, имеющих существенно различные величины ресурсов, ускорение эволюции менее мощной системы будет выше, и тем выше, чем больше разность величин их ресурсов.

Наличие взаимосвязей между системами не всегда приводит к их коэволюции. Для возникновения эффекта коэволюции системы и связи между ними должны удовлетворять ряду требований. Так, ресурсы коэволюционирующих систем должны быть согласованы, взаимодействия между системами должны быть достаточно регулярными, «сила» взаимосвязей систем должна иметь достаточный уровень. Поясним последнее требование. Низкий уровень «силы» связей сохраняет автономность систем. Слишком жесткие связи приводят к потере системами самостоятельности, так что они становятся подсистемами одной системы.

6.4.1. Классификация коэволюционных алгоритмов

Наиболее известным примером коэволюционного подхода к решению задачи глобальной оптимизации является, вероятно, островная модель *(island model)*. Суть модели состоит в том, что эволюционирующую популяцию делят на субпопуляции, которые эволюционируют, по крайней мере на логическом уровне, параллельно. Эпизодически некоторые агенты перемещаются из одной субпопуляции в другую, передавая целевым популяциям свой опыт. На основе островной модели сформировался класс многопопуляционных ал-

горитмов, широко используемых для организации параллельного решения задачи глобальной оптимизации (гл. 9).

Коэволюционные алгоритмы классифицируем по используемой модели коэволюции, форме коэволюции, числу коэволюционирующих субпопуляций, схеме взаимодействия между субпопуляциями.

Модель коэволюции

Выделяют два основных класса вычислительных моделей коэволюции – простые и композиционные модели.

Чаще всего в вычислительной практике используют *простые* модели коэволюции, в которых в качестве решения исходной задачи может быть использовано решение, найденное любой из коэволюционирующих субпопуляций.

В *композиционных* моделях коэволюции решение задачи представляет собой объединение его фрагментов, найденных различными субпопуляциями. Например, если решается задача структурной и параметрической оптимизации некоторого объекта, то первой субпопуляции может быть поручена структурная оптимизация, а второй субпопуляции – параметрическая оптимизация.

Форма коэволюции

В зависимости от характера взаимодействия между коэволюционирующими субпопуляциями различают две основные формы коэволюции – сотрудничество и соперничество.

Коэволюция типа *сотрудничества* (*cooperative coevolution*) предполагает, что каждая из субпопуляций решает либо одну и ту же задачу оптимизации, либо ее часть. Важным частным случаем данного типа сотрудничества является *композиционно-кооперативное* сотрудничество, когда каждая из субпопуляций отыскивает свою часть общего решения задачи. При этом может быть использована декомпозиция вектора варьируемых параметров и/или декомпозиция целевой функции. Коэволюция данного типа целесообразна в случае решения задач, имеющих высокую вычислительную и/или структурную сложность.

Коэволюция типа *соперничества* (*competitive coevolution*) использует один из следующих типов взаимодействия между коэволюционирующими субпопуляциями:

– взаимодействие по схеме «хозяин–паразит», когда перераспределение ресурсов между субпопуляциями отсутствует, но пригодность агентов данной субпопуляции определяется путем их сравнения с агентами другой субпопуляции;

– взаимодействие субпопуляций, имеющих различные области поиска;

– взаимодействие субпопуляций, отличающихся стратегиями поиска (алгоритмами поиска и/или значениями их свободных параметров).

Коэволюция последнего типа может быть использована для самоадаптации алгоритма поисковой оптимизации (гл. 7).

Указанные типы коэволюции формально можно проиллюстрировать с помощью системы дифференциальных уравнений Лотки – Вольтерры (*A. J. Lotka, V. Volterra)*

$$\frac{d\boldsymbol{n}}{dt} = \boldsymbol{N}\left(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{n}\right),\ \boldsymbol{n}(0) = \boldsymbol{n}^0,$$

используемой в *популяционной динамике* для описания взаимодействия различных популяций. Здесь $\boldsymbol{n} = (n_i,\ i \in [1:|\boldsymbol{n}|])$ представляет собой $(|\boldsymbol{n}| \times 1)$-вектор, определяющий численности коэволюционирующих субпопуляций, так что n_i – численность субпопуляции S_i; $\boldsymbol{n}^0$ – вектор, определяющий начальные численности субпопуляций; $(|\boldsymbol{n}| \times |\boldsymbol{n}|)$-диагональная матрица $\boldsymbol{N}$ имеет вид

$$\boldsymbol{N} = \begin{pmatrix} n_1 & 0 & \dots & 0 \\ 0 & n_2 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & n_{|\boldsymbol{n}|} \end{pmatrix};$$

$\boldsymbol{A} = \left(a_{i,j}, i, j \in [1:|\boldsymbol{n}|]\right)$ есть $(|\boldsymbol{n}| \times |\boldsymbol{n}|)$-матрица, элемент которой $a_{i,j}$ формализует воздействие субпопуляции S_j на численность субпопуляции S_i; $\boldsymbol{b}$ – $(|\boldsymbol{n}| \times 1)$-вектор, компоненты которого определяют скорость роста численности соответствующих субпопуляций; $t \in [0;\hat{t}]$ – время коэволюции. Коэволюцию сотрудничества субпопуляций S_i, S_j определяют неравенства $a_{i,j} < 0$, $a_{j,i} < 0$; коэволюцию соперничества тех же субпопуляций – неравенства $a_{i,j} > 0$, $a_{j,i} > 0$. Ситуации $a_{i,j} < 0$, $a_{j,i} > 0$ соответствует взаимоотношение между субпопуляциями i, j типа *хищник–жертва*.

Число коэволюционирующих субпопуляций

С этой точки зрения различают *многопопуляционные* коэволюции и частный случай последних – *двухпопуляционные* коэволюции.

Взаимодействие между субпопуляциями

Выделяют взаимодействие по последовательной и параллельной схемам.

В *последовательной* схеме обновление текущего числа агентов популяций производится последовательно, так что текущая численность данной субпопуляции зависит от ее численности на предыдущей итерации и текущей численности уже обновленных субпопуляций:

$$n_i' = q_i\left(n_1', n_2', \dots, n_{i-1}', n_i\right),\ i \in [1:|\boldsymbol{n}|]. \tag{6.6}$$

В *параллельной* схеме то же обновление выполняют параллельно в соответствии с выражением

$$n_i' = q_i\left(n_1, n_2, \dots, n_{i-1}, n_i\right),\ i \in [1:|\boldsymbol{n}|]. \tag{6.7}$$

В качестве функций $q_i(\cdot)$ в выражениях (6.6), (6.7) могут быть использованы линейные и нелинейные функции. В простейшем случае эти функции задают с помощью величины штрафа n_p, назначаемой проигравшим субпопуляционным алгоритмам.

Возможны различные комбинации рассмотренной последовательной и параллельной схем взаимодействия коэволюционирующих субпопуляций.

6.4.2. Пример коалгоритма

Рассмотрим в качестве примера коалгоритма гибридный алгоритм *CPSO* (*Co-evolutionary PSO*), предложенный в одной из работ Чжоу (*Y. Zhou*) и Пэя (*Sh. Pei*).

Пусть требуется решить задачу глобальной условной минимизации с ограничениями типа неравенств $g_j(X) \geq 0$, $j \in [1:|G|]$. Алгоритм *CPSO* использует популяцию S_2, число частиц в которой равно числу ограничивающих функций, то есть $|S_2| = |G|$, а также мультипопуляцию $\boldsymbol{S}_1 = (S_{1,1}, S_{1,2}, ..., S_{1,|G|})$, содержащую $|G|$ роев по $|S_1| = |S|$ частиц в каждом (рис. 6.7). Размерность вектора параметров, ассоциированного с каждой из частиц роев $S_{1,i}$, равна $|X|$. В рое S_2 размерность того же вектора равна числу штрафных коэффициентов в штрафной функции, с помощью которой задача условной оптимизации сводится к задаче безусловной оптимизации. В алгоритме *CPSO* эта размерность равна двум (см. ниже).

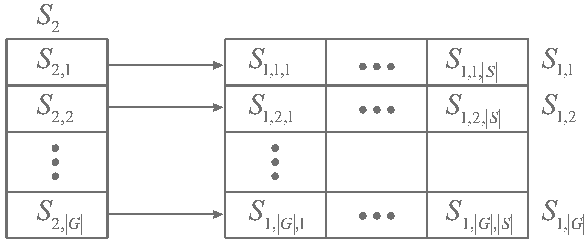


Рис. 6.7. Популяции, используемые гибридным алгоритмом *CPSO*

Рои $S_{1,1}, S_{1,2}, ..., S_{1,|G|}$ используются для параллельного поиска решения упомянутой задачи безусловной оптимизации, а рой S_2 – для поиска оптимальных значений коэффициентов штрафа. Говоря более строго, рой $S_{1,i}$ эволюционирует со значениями коэффициентов штрафа $p_{i,1}, p_{i,2}$, которые определяет частица $s_{2,i}$ роя S_2. Таким образом, в алгоритме *CPSO* имеет место коэволюция $(|G|+1)$ популяций.

Частица $s_{1,i,j}$ роя $S_{1,i}$ использует фитнес-функцию

$$\varphi_{1,i,j}(P_i, X_{i,j}) = f(X_{i,j}) + p_{i,1}\sum_{k=1}^{|G|} g_k(X_{i,j}) + p_{i,2}\left|\tilde{G}_{i,j}\right|, \; i \in [1:|G|], \; j \in [1:|S|], \quad (6.8)$$

где $P_i = (p_{i,1}, p_{i,2})$ – текущий вектор координат частицы $s_{2,i}$, $X_{i,j} = (x_{i,j,1}, x_{i,j,2}, ..., x_{i,j,|X|})$ – текущий вектор координат частицы $s_{1,i,j}$, $\left|\tilde{G}_{i,j}\right| \leq |G|$ – число ограничений $G(X_{i,j}) \geq 0$, нарушенных частицей $s_{1,i,j}$.

Назовем частицу $s_{2,i}$ роя S_2 *правильной* частицей, если в рое $S_{1,i}$ имеется хотя бы одна допустимая частица (удовлетворяющая всем ограничениям $g_j(X) \geq 0$), и *неправильной* частицей – в противном случае. Текущие числа правильных и неправильных частиц в рое S_2 обозначим соответственно $\left|\overline{G}\right|$ и $\left|\tilde{G}\right| = |G| - \left|\overline{G}\right|$. Положим, кроме того, что $\left|\overline{S}_j\right|$ и $\left|\tilde{S}_j\right| = |S| - \left|\overline{S}_j\right|$ – текущие числа соответственно допустимых и недопустимых решений в рое $S_{1,j}$.

Во введенных обозначениях фитнес-функция частицы $s_{2,i}$, $i \in [1:|G|]$ имеет вид

$$\varphi_{2,i}(X_{i,1},...,X_{i,|S|}) = \begin{cases} \dfrac{1}{\left|\overline{S}_i\right|}\sum\limits_{j=1}^{\left|\overline{S}_i\right|} f(X_{i_j}) - \left|\overline{S}_i\right|, & \left|\overline{S}_i\right| > 0, \\ \max\limits_{l \in [1:\left|\tilde{G}\right|]} \varphi_{2,k_l} + \dfrac{1}{\left|\tilde{S}_i\right|}\sum\limits_{m=1}^{\left|\tilde{S}_i\right|} g_{i_m}(X_{i_m}) - \left|\tilde{S}_i\right|, & \left|\overline{S}_i\right| = 0, \end{cases} \tag{6.9}$$

где X_{i_j} – допустимое решение, соответствующее частице s_{1,i_j} роя $S_{1,i}$, $\max\limits_{l \in [1:\left|\tilde{G}\right|]} \varphi_{2,k_l}$ – текущее максимальное значение фитнес-функции среди всех правильных частиц роя S_2; X_{i_m} – недопустимое решение, соответствующее частице s_{1,i_m} роя $S_{1,i}$.

Коэволюционное взаимодействие роев $\{S_{1,i},\ i \in [1:|G|]\}$, S_2 протекает по следующей схеме.

1) Реализуем параллельное развитие роев $S_{1,i}$ фиксированное число итераций в соответствии с каноническим алгоритмом роя частиц, используя в качестве фитнес-функций функции (6.8), а в качестве коэффициентов штрафа $p_{i,1}, p_{i,2}$ – текущие координаты частицы $s_{2,i}$.

2) Выполняем фиксированное число итераций канонического алгоритма роя частиц для роя S_2, используя в качестве фитнес-функций функции (6.9).

Рои $S_{1,i}$, $i \in [1:|G|]$ могут эволюционировать параллельно как на логическом, так и на физическом уровнях. В последнем случае могут быть использованы как синхронные, так и асинхронные итерации.

Примером коалгоритма является генетический коэволюционный алгоритм (см. 2.12). Примером коалгоритмов также может служить алгоритм эволюции разума *МЕС* (см. 5.4), если предположить, что субпопуляции этого алгоритма используют различные популяционные алгоритмы и/или различные значения параметров этих алгоритмов.

Основное отличие коалгоритма *МЕС* от генетического коэволюционного алгоритма по используемым средствам гибридизации состоит в назначении более «сурового» наказания для проигравшей популяции. Если в генетическом коэволюционном алгоритме при проигрыше только уменьшается численность этой популяции, то в коалгоритме *МЕС* такая популяция объявляется временной и затем может быть вообще выведена из конкурентной борьбы.

При синтезе коалгоритма *МЕС* возникает проблема выбора популяционного алгоритма и/или значений его параметров, которые должна использовать новая популяция, создающаяся взамен выбывшей популяции. Возможные пути решения этой проблемы зависят от того, используются ли при гибридизации гомогенные или гетерогенные алгоритмы. В первом случае проблема сводится

к выбору значений их свободных параметров, а во втором – к выбору типа алгоритма (и, быть может, значений его параметров). И в том, и в другом случае можно использовать альтернативную адаптацию или метаоптимизационные подходы, рассматриваемые в гл. 7.

Вопросы для самопроверки

1. Назовите классификационные признаки способов гибридизации популяционных алгоритмов в одноуровневой классификации Ванга, двухуровневой классификации Эль-Абда и Камэла и четырехуровневой классификации Рейдла.

2. Сформулируйте основные способы высокоуровневой и низкоуровневой гибридизации вложением.

3. Определите основные способы гибридизации по схеме препроцессор/постпроцессор.

4. Сформулируйте принципы коэволюционного подхода к решению задачи оптимизации. Определите используемые в коэволюционных алгоритмах модели коэволюции.

5. Дайте схему гибридного коалгоритма на основе алгоритма роя частиц.

Глава 7

МЕТАОПТИМИЗАЦИЯ ПОПУЛЯЦИОННЫХ АЛГОРИТМОВ

Назовем вектор свободных параметров популяционного алгоритма оптимизации *стратегией алгоритма*. Стратегия популяционного алгоритма в значительной степени определяет его эффективность. Поэтому возникает задача поиска оптимальной стратегии алгоритма, которая называется *метазадачей оптимизации*. Алгоритм оптимизации и соответствующую программу назовем *базовым алгоритмом* и *базовой программой*. Аналогично метод, алгоритм и программу решения метазадачи назовем *метаметодом*, *метаалгоритмом* и *метапрограммой*.

Далее представляем постановку метазадачи и классификацию методов ее решения, рассматриваем несколько подходов к решению задачи параметрической метаоптимизации. В завершение обсуждаем задачу структурной метаоптимизации.

7.1. ПОСТАНОВКА МЕТАЗАДАЧИ ОПТИМИЗАЦИИ

Некоторый класс задач многомерной глобальной безусловной минимизации обозначим $\Phi = (\phi_i, i = 1, 2, ...)$, где $\phi_i = \phi$ одна из задач этого класса. Отметим, что поскольку речь идет о безусловной оптимизации, задачу $\phi_i = \phi$ полностью определяет минимизируемая функция $f_i = f$.

7.1.1. Статическая параметрическая метаоптимизация

Статическая параметрическая метазадача (*parameter adaptation problem*) предполагает, что оптимальная стратегия базового алгоритма является статической и отыскивается для класса задач Φ.

Пусть $a(B)$ – рассматриваемый базовый алгоритм решения задач класса Φ, где $B = (b_1, b_2, ..., b_{|B|})$ – вектор стратегий этого алгоритма. Положим, что для каждого из компонент b_i вектора B заданы интервалы его допустимых значений b_i^-, b_i^+, так что определено множество D_B допустимых стратегий

$$D_B = \{b_i \mid b_i^- \le b_i \le b_i^+, i \in [1:|B|]\}.$$

Совокупность алгоритмов, которую определяет множество D_B, обозначим

$$A = \{a(B) \mid B \in D_B\}.$$

Рассматриваем две параметрических метазадачи, первая из которых ставится следующим образом. Для функции $\phi \in \Phi$ найти стратегию B^* (а тем самым, и алгоритм $a^* = a(B^*) \in A$), которая обеспечивает минимальное значение *целевой метафункции* $\mu(\phi, B)$:

$$\min_{B \in D_B} \mu(\phi, B) = \mu(\phi, B^*). \tag{7.1}$$

Очевидно, что можно предложить целый ряд функций $\mu(\phi, B)$, отражающих разные аспекты представлений лица, принимающего решения, об оптимальности стратегии B^*. Так что метазадачу (7.1), вообще говоря, следует рассматривать как многоцелевую. Обозначим частные целевые метафункции этой задачи $\mu_i(\phi, B)$, $i = 1, 2, ...$

В качестве одной из частных функций (пусть это будет функция $\mu_1(\phi, B)$) обычно используют оценку времени решения задачи ϕ с помощью алгоритма $a(B)$. Это время состоит из двух основных компонентов – времени испытаний (вычислений значений функции $f(X)$ и, возможно, ее производных) и времени вычислений, связанных с реализацией самого базового алгоритма. В практически важных задачах первое время многократно превышает второе. Поэтому можно считать, что

$$\mu_1(\phi, B) = v n_f(\phi, B), \tag{7.2}$$

где v – быстродействие используемой вычислительной системы (среднее время выполнения одной арифметической операции), $n_f(\phi, B)$ – суммарное число арифметических операций, необходимых для решения задачи ϕ с помощью алгоритма $a(B)$.

В качестве другой частной метафункции (пусть это будет функция $\mu_2(\phi, B)$) логично использовать оценку вычислительных затрат на решение собственно метазадачи оптимизации (7.1). Например, по аналогии с формулой (7.2) можно положить, что

$$\mu_2(\phi, B) = v n_m(\phi, B), \tag{7.3}$$

где $n_m(\phi, B)$ – суммарное число арифметических операций, необходимых для решения метазадачи (7.1).

Положим, что тем или иным образом, например с помощью аддитивной скалярной свертки векторная метафункция $\mu_i(\phi, B)$, $i = 1, 2, ...$ сведена к скалярной функции,

$$\mu(\phi, B) = \sum_i \lambda_i \mu_i(\phi, B), \tag{7.4}$$

где λ_i – вещественный неотрицательный весовой множитель.

Во введенных обозначениях и соглашениях постановка второй параметрической метазадачи оптимизации имеет следующий вид. Для класса задач Φ найти стратегию B^* (а тем самым и алгоритм $a^* = a(B^*) \in A$), которая обеспечивает минимальное значение целевой метафункции $\mu(\Phi, B)$:

$$\min_{B \in D_B} \mu(\Phi, B) = \mu(\Phi, B^*). \tag{7.5}$$

Практическое решение (7.5) выполняем по следующей схеме:

– формируем представительную совокупность задач ϕ класса Φ;

– для каждой из этих задач и данной стратегии $B \in D_B$ решаем метазадачу (7.1) и вычисляем соответствующее значение функции $\mu(\phi, B)$;

– на основе обработки полученных величин $\mu(\phi, B)$ находим значение функции $\mu(\Phi, B)$;

– повторяем указанную последовательность шагов для достаточно большого числа различных стратегий $B \in D_B$ и на основе результатов этих повторов определяем приближенно оптимальную стратегию $B^* \in D_B$.

Как и задача (7.1), метазадача (7.5) является в общей постановке многоцелевой. Частные целевые функции этой задачи обозначаем $\mu_i(\Phi, B)$, $i = 1, 2, ...$

Функции $\mu_i(\Phi, B)$ можно построить на основе функции $\mu(\phi, B)$, $\phi \in \Phi$. Например, примем, что $\mu_1(\Phi, B)$ представляет собой максимальное гарантированное значение функции $\mu(\phi, B)$ на классе задач Ф:

$$\mu_1(\Phi, B) = \max_{\phi \in \Phi} \mu(\phi, B).$$

Аналогично можно положить, что

$$\mu_2(\Phi, B) = \underset{\phi \in \Phi}{\text{МО}}\, \mu(\phi, B) = \bar{\mu}(B) \tag{7.6}$$

и

$$\mu_3(\Phi, B) = \underset{\phi \in \Phi}{\text{СКО}}\, \mu(\phi, B) = \sigma\big(\mu(\phi, B)\big), \tag{7.7}$$

где МО и СКО – операции вычисления оценок математического ожидания и среднеквадратичного отклонения величин $\mu(\phi, B)$ соответственно.

Целевые метафункции $\mu_i(\Phi, B)$ можно строить также без использования функций $\mu(\phi, B)$. Например, в качестве функции $\mu_4(\Phi, B)$ можно принять оценку $\bar{\xi}(\phi, B)$ вероятности *нелокализации* минимума функции $f(X)$ на классе задач Ф базовым алгоритмом $a(B)$:

$$\mu_4(\Phi, B) = \bar{\xi}(\phi, B). \tag{7.8}$$

С помощью аддитивной или иной скалярной свертки векторную функцию $\mu_i(\Phi, B)$, $i = 1, 2, ...$ сводим к скалярной функции

$$\mu(\Phi, B) = \sum_i \lambda_i\, \mu_i(\Phi, B),$$

аналогичной функции (7.4).

7.1.2. Динамическая параметрическая метаоптимизация

Как неоднократно отмечалось выше, на различных этапах функционирования популяционных алгоритмов оптимальными являются различные значения их свободных параметров, то есть различные стратегии. Например, на ранних этапах эволюции предпочтительным является большой шаг мутации в генетическом алгоритме, обеспечивающий широкое исследование пространства поиска. На поздних этапах эволюции, напротив, этот шаг должен быть уменьшен, чтобы обеспечить высокую точность решения.

Поэтому наряду с рассмотренной статической задачей метаоптимизации в настоящее время параметрическая метазадача оптимизации часто ставится как динамическая. Эта задача предполагает отыскание простых и в то же время эффективных правил изменения свободных параметров базового алгоритма, то есть поиск *динамических стратегий* этого алгоритма. Значительным стимулом для исследований в этом направлении является тот факт, что применение даже простого правила варьирования стратегии базового алгоритма часто дает существенный выигрыш в эффективности алгоритма по сравнению с использованием оптимальных статических стратегий.

Динамическая стратегия метаоптимизации может быть *априорной* (*неадаптивной*) и *апостериорной* (*адаптивной*). В простейшем случае эти стра-

тегии задают в виде простых эвристических правил. Рассмотрим в качестве примера два таких правила.

Пусть речь идет о генетическом алгоритме и операторе гауссовой мутации вида

$$x'_{i,j} = x_{i,j} + N_1(0, \sigma),\ i \in [1:|S|],\ j \in [1:|X|].$$

Придадим величине σ (шагу мутации) смысл стратегии алгоритма, так что $|B| = 1$ и $b_1 = \sigma$. В простейшем случае для всех компонент вектора X_i и всех поколений $t \in [0:\hat{t}]$ используют одно и то же фиксированное значение величины b_1. Известным неадаптивным эвристическим правилом изменения этой величины является правило

$$b_1(t) = 1 - 0{,}9\frac{t}{\hat{t}},$$

в соответствии с которым с ростом числа итераций t, шаг мутации линейно уменьшается с единицы при $t = 0$ до 0,1 в конце итераций, когда $t = \hat{t}$.

Примером адаптивной динамической стратегии является правило 1/5 Рехенберга (*I. Rechenberg*), которое может быть записано в виде:

– если номер текущей итерации t кратен периоду адаптации t_{adap}, то

$$b_1(t) = \begin{cases} \dfrac{b_1(t - t_{adap})}{c_m}, & \varpi_s < 1/5, \\ b_1(t - t_{adap})c_m, & \varpi_s > 1/5, \\ b_1(t - t_{adap}), & \varpi_s = 1/5; \end{cases}$$

– в противном случае

$$b_1(t) = b_1(t-1).$$

Здесь ϖ_s – средняя относительная доля успешных мутаций, измеренная на основе результатов эволюции популяции в течение t_{adap} поколений; $c_m \in [0{,}817; 1{,}0]$ – заданная константа. Успешной называется мутация, порождающая потомка лучшего, чем родитель.

Таким образом, правило 1/5 заключается в следующем (п. 2.7). Доля успешных мутаций должна быть равна 1/5. Если эта доля меньше 1/5, то шаг мутации увеличиваем, а если больше 1/5, то уменьшаем. Изменение шага мутации выполняем после каждых t_{adap} поколений генетического алгоритма.

Рассмотрим общие постановки неадаптивной динамической и адаптивной динамической метазадач оптимизации.

Неадаптивная динамическая метаоптимизация

В общем случае неадаптивную динамическую стратегию отыскивают в виде некоторой функции номера итерации t, то есть в виде $B = B(t) = (b_1(t), b_2(t), ..., b_{|B|}(t))$, и метазадачу оптимизации сводят к отысканию функциональных зависимостей $b_1(t), b_2(t), ..., b_{|B|}(t)$, доставляющих экстремум целевой метафункции. Тем самым, в общей постановке динамическая мета-

задача оптимизации представляет собой задачу оптимального управления, то есть задачу синтеза оптимального программного закона управления.

Поскольку вычислительная сложность задачи оптимального управления обычно многократно превышает вычислительную сложность базовой задачи оптимизации, динамическую метазадачу сводят к задаче нелинейного программирования по следующей схеме.

Фиксируем классы функций, которым принадлежат функции $b_i(t)$, $i \in [1:|B|]$. Примерами таких классов функций являются полиномиальные функции, порядок которых не превышает заданный, экспоненциальные функции и так далее. Вектор параметров, определяющий класс функций, которому принадлежит функция $b_i(t)$, обозначаем C_i, $i \in [1:|B|]$. Если, например, речь идет о классе квадратичных функций, когда

$$b_i(t, C_i) = c_{i,1}t^2 + c_{i,2}t + c_{i,3},$$

то можно принять размерность вектора C_i равной трем.

Для простоты записи положим, что размерности всех векторов C_i, $i \in [1:|B|]$ одинаковы и равны $|C|$. Заметим, что обычно эта величина не превышает двух. Область допустимых значений компонент вектора C_i обозначаем

$$D_{C_i} = \{c_{i,j} \mid c_{i,j}^- \le c_{i,j} \le c_{i,j}^+,\ j \in [1:|C|]\},\ i \in [1:|B|],$$

где $c_{i,j}^-, c_{i,j}^+$ – заданные интервалы допустимых значений параметра $c_{i,j}$.

Введем еще следующие обозначения: $\boldsymbol{C} = (C_1, C_2, ..., C_{|B|})$ – вектор *динамических параметров* стратегии B; $\boldsymbol{D}_C = D_{C_1} \oplus D_{C_2} \oplus \cdots \oplus D_{C_{|B|}}$ – область допустимых значений этого вектора. Здесь $\oplus$ – символ прямого суммирования множеств.

Во введенных обозначениях базовый алгоритм решения задач класса Φ приобретает вид $a(B) = a(\boldsymbol{C})$, а совокупность алгоритмов, которую определяет множество $\boldsymbol{D}_C$, – вид

$$A = \{a(\boldsymbol{C}) \mid \boldsymbol{C} \in \boldsymbol{D}_C\}.$$

Метазадачи неадаптивной динамической оптимизации, аналогичные задачам (7.1), (7.5), записываются в результате в форме

$$\min_{\boldsymbol{C} \in \boldsymbol{D}_C} \mu(\phi, \boldsymbol{C}) = \mu(\phi, \boldsymbol{C}^*), \tag{7.9}$$

$$\min_{\boldsymbol{C} \in \boldsymbol{D}_C} \mu(\Phi, \boldsymbol{C}) = \mu(\Phi, \boldsymbol{C}^*), \tag{7.10}$$

где $\boldsymbol{C}^* = (C_1^*, C_2^*, ..., C_{|B|}^*)$ – оптимальный вектор динамических параметров алгоритма $a(\boldsymbol{C})$, определяющий оптимальную динамическую стратегию $B(t, \boldsymbol{C}^*) = (b_i(t, C_i^*), i \in [1:B])$.

Адаптивная динамическая метаоптимизация

Положим, что $W(t) = (w_i(t), i \in [1:|W|])$ – вектор характеристик текущего состояния популяции. Примерами таких характеристик могут служить текущее минимальное или максимальное расстояние между агентами популяции, оценка текущей скорости сходимости итерационного процесса, некоторая мера разнообразия популяции и так далее.

В общем случае адаптивную динамическую стратегию базового алгоритма отыскивают в виде функции $B = B(W(t)) = (b_i(W(t)), i \in [1:|B|])$, так что метазадача оптимизации сводится к отысканию оптимальных функциональных зависимостей $b_i(W(t))$. В терминах теории управления данная задача представляет собой задачу синтеза оптимального позиционного закона управления. Вычислительная сложность этой задачи многократно превосходит сложность задачи оптимального управления. Поэтому метазадачу адаптивной динамической оптимизации решают путем сведения к задаче нелинейного программирования по схеме, близкой к рассмотренной выше.

В более общем случае динамическая адаптивная стратегия метаоптимизации может иметь вид $B(t, W(t))$.

7.1.3. Структурная метаоптимизация

Положим, что рассматриваемый базовый популяционный алгоритм a включает в себя набор миграционных операторов $o_i, i \in [1:|O|]$, например операторы мутации, кроссовера, отбора и так далее в генетическом алгоритме. Пусть $o_{i,j}, j \in [1:|O_i|]$,– совокупность возможных вариантов реализации оператора o_i.

Некоторый набор вариантов реализации всех $|O|$ операторов алгоритма a представляем в виде $(|O| \times 1)$-вектора $O = (o_{i,j_i}, i \in [1:|O|],\ j_i \in [1:|O_i|])$ и называем *структурной стратегией алгоритма* $a = a(O)$. Множество допустимых стратегий D_O определяет выражение

$$D_O = \left\{ O_{i,j_i}, i \in [1:|O|],\ j_i \in [1:|O_i|] \right\},$$

а его мощность равна

$$|D_O| = \prod_{i=1}^{|O|} |O_i|.$$

Совокупность базовых алгоритмов, которую определяет это множество, обозначаем

$$A = \{a(O) \mid O \in D_O\}.$$

Таким образом, по аналогии с рассмотренной выше статической параметрической метазадачей оптимизации могут быть поставлены следующие *статические метазадачи структурной оптимизации* (*operator adaptation problem*).

1. Для задачи $\phi \in \Phi$ найти стратегию $O^* \in D_O$ (а тем самым, и алгоритм $a^* = a(O^*) \in A$), которая обеспечивает минимальное значение целевой метафункции $\mu(\phi, O)$:

$$\min_{O \in D_O} \mu(\phi, O) = \mu(\phi, O^*). \tag{7.11}$$

2. Для класса задач Φ найти стратегию $O^* \in D_O$ (а тем самым и алгоритм $a^* = a(O^*) \in A$), которая обеспечивает минимальное значение метафункции $\mu(\Phi, O)$:

$$\min_{O \in D_O} \mu(\Phi, O) = \mu(\Phi, O^*). \tag{7.12}$$

Отметим, что если в метазадаче статической параметрической оптимизации выражения (7.1), (7.5) представляют собой задачи непрерывной оптимизации, то аналогичные выражения (7.11), (7.12) определяют задачи дискретной оптимизации.

Относительно функций $\mu(\phi,O)$, $\mu(\Phi,O)$ в метазадачах (7.11, (7.12) справедливы соображения, аналогичные рассмотренным выше относительно функций $\mu(\phi,B)$, $\mu(\Phi,B)$ в метазадачах (7.1), (7.5). Так, функции $\mu(\phi,O)$, $\mu(\Phi,O)$ в общем случае представляют собой скалярные свертки соответствующих частных целевых метафункций $\mu_i(\phi,O)$, $\mu_j(\Phi,O)$.

Аналогично параметрической метаоптимизации, наряду с рассмотренной статической задачей имеет место *динамическая* метазадача структурной оптимизации и два ее варианта – неадаптивная и адаптивная структурные задачи.

В *неадаптивной* динамической метазадаче структурной оптимизации стратегия O является некоторой фиксированной функцией номера поколения t, так что $O = O(t)$. Вычислительная сложность этой задачи также может быть приемлемой только в случае использования простых параметризованных целочисленных функций вида $o_i(C_i(t)), i \in [1:|O|]$, где C_i – вектор вещественных динамических параметров структурной стратегии o_i. Таким образом, в соглашениях предыдущего пункта неадаптивная динамическая метазадача структурной оптимизации сводится к задачам непрерывной оптимизации вида (7.9), (7.10).

В *адаптивной* динамической метазадаче структурной оптимизации стратегия базового алгоритма имеет вид $O(W(t))$ или $O(t, W(t))$.

7.1.4. Структурно-параметрическая метаоптимизация

Положим, что каждый из вариантов $o_{i,j}$ реализации оператора O_i имеет свободные параметры, которые образуют вектор $B_{i,j}$; $i \in [1:|O|]$, $j \in [1:|O_i|]$. Для простоты записи примем, что размерности и области допустимых значений всех векторов $B_{i,j}$ одинаковы и равны $|B_i|$, D_{B_i} соответственно, так что $o_{i,j} = o_{i,j}(B_i)$, $B_i \in D_{B_i}$.

Во введенных обозначениях структурная стратегия O базового алгоритма a оказывается зависимой от вектора параметров $\boldsymbol{B} = (B_i, i \in [1:|O|])$ и становится *структурно-параметрической* стратегией вида $O(\boldsymbol{B})$, $\boldsymbol{B} \in \boldsymbol{D}_B$, где $\boldsymbol{D}_B = D_{B_1} \oplus D_{B_2} \oplus \ldots \oplus D_{B_{|O|}}$; $\oplus$ – символ прямого суммирования множеств. Совокупность базовых алгоритмов, которую определяет множество структурно-параметрических стратегий, приобретает вид

$$A = \{a(O(\boldsymbol{B})) \mid O \in D_O, \boldsymbol{B} \in \boldsymbol{D}_B\}.$$

Таким образом, *статическая* метазадача структурно-параметрической оптимизации сводится к одной из задач смешанного (непрерывно-целочисленного) программирования

$$\min_{O \in D_O,\, \boldsymbol{B} \in \boldsymbol{D}_B} \mu(\phi, O(\boldsymbol{B})) = \mu(\phi, O^*(\boldsymbol{B}^*)),$$

$$\min_{O \in D_O,\, \boldsymbol{B} \in \boldsymbol{D}_B} \mu(\Phi, O(\boldsymbol{B})) = \mu(\Phi, O^*(\boldsymbol{B}^*)).$$

Относительно целевых функций $\mu(\phi, O(\boldsymbol{B}))$, $\mu(\Phi, O(\boldsymbol{B}))$ в последних задачах опять же справедливы соображения, аналогичные рассмотренным выше относительно функций $\mu(\phi,B)$, $\mu(\Phi,B)$ в метазадачах (7.1), (7.5).

По рассмотренной методике легко поставить динамическую неадаптивную метазадачу структурно-параметрической оптимизации, а также динамическую адаптивную метазадачу структурно-параметрической оптимизации.

7.2. КЛАССИФИКАЦИЯ МЕТОДОВ МЕТАОПТИМИЗАЦИИ

Прежде заметим, что в соответствии с результатами предыдущего параграфа имеет место трехуровневая классификация метазадач оптимизации, представленная на рис.7.1.

Рис. 7.1. Классификация метазадач оптимизации

Известно несколько классификаций методов метаоптимизации. Наибольшую известность получили классификация Эйбена (*A. E. Eiben*) и классификация Смита – Фогарти (*J. E. Smith*, *T. C. Fogarty*).

Известны попытки теоретического анализа влияния различных стратегий на эффективность некоторых современных алгоритмов поисковой оптимизации. Например, известны публикации, в которых на основе такого анализа рассмотрен выбор оптимальных значений параметров метода роя частиц. Однако теоретические исследования основаны на сильных упрощениях базового алгоритма и имеют ограниченную практическую ценность. Обычно метазадачу оптимизации решают численными методами.

7.2.1. Классификация Эйбена

Будем говорить для определенности о параметрической метаоптимизации. Классификация легко переносится на структурную и структурно-параметрическую метаоптимизации. В соответствии с классификацией Эйбена на верхнем уровне иерархии выделяют две группы методов метаоптимизации – методы настройки параметров и методы управления параметрами (рис. 7.2).

Методы настройки параметров

Методы настройки параметров (*parameter tuning*) предполагают решение метазадачи оптимизации для ряда задач некоторого класса. Найденные опти-

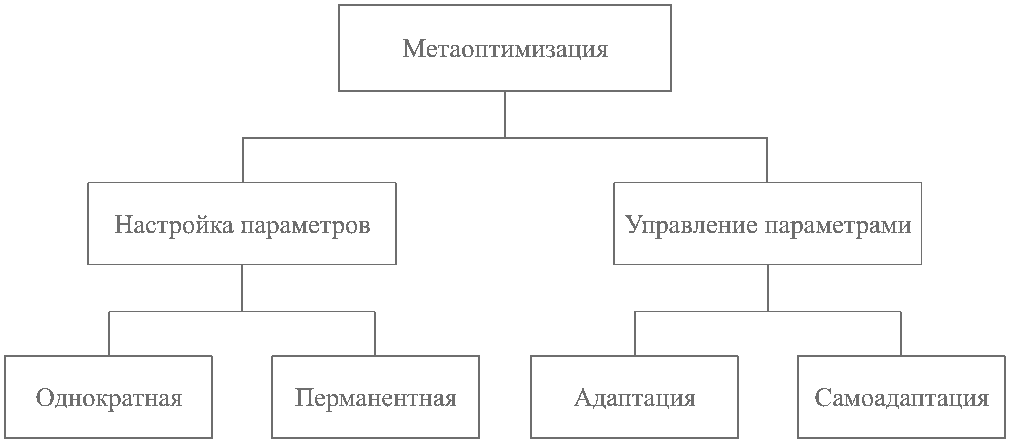


Рис. 7.2. Классификация Эйбена

мальные стратегии базового алгоритма фиксируют и используют для решения других оптимизационных задач данного класса.

На втором иерархическом уровне методы настройки параметров разделяют на методы однократной и методы перманентной настройки. В некоторых публикациях методы настройки параметров называются *эмпирическими* методами (*empirical approach*).

Методы *однократной* настройки параметров (*one-time parameter tuning*) представляют собой классические методы метаоптимизации. Идея методов состоит в том, что программу, реализующую рассматриваемый базовый алгоритм оптимизации, многократно выполняют при различных стратегиях на большом числе задач, принадлежащих рассматриваемому классу оптимизационных задач. На основе результатов исследования выбирают стратегию с наилучшими показателями эффективности.

Исторически методы однократной настройки были впервые широко использованы при метаоптимизации генетического алгоритма. На основе этих исследований были получены стратегии генетического алгоритма, оптимальные для некоторых тестовых задач. При этом ошибочно полагалось, что свободные параметры генетического алгоритма устойчивы к вариациям задач, то есть их оптимальные значения примерно одинаковы для широкого класса задач оптимизации. К настоящему времени установлено, что оптимальные стратегии генетического алгоритма существенно различны для разных классов таких задач.

По схеме, близкой к рассмотренной, методом однократной настройки получена, например, стратегия ставшего классическим алгоритма Нелдера – Мида (п. 3.2.5). В некоторых программных библиотеках, включающих в себя реализацию этого алгоритма, данную стратегию используют по умолчанию (библиотека *NAG*), а в других библиотеках параметры этой стратегии даже не являются настраиваемыми (библиотека системы *MATLAB*).

Основной проблемой метода однократной настройки является экспоненциальный рост вычислительных затрат при увеличении числа параметров в стратегии базового алгоритма. Из-за невозможности исчерпывающего

исследования полученная в результате стратегия может оказаться далекой от оптимальной.

В методе *перманентной* настройки (*permanent parameter tuning*) метапрограмма в процессе функционирования базовой программы непрерывно собирает информацию об эффективности различных стратегий решения конкретных задач рассматриваемого класса и варьирует эти стратегии, пытаясь повысить эффективность базового алгоритма. Известны примеры реализации такого подхода применительно к генетическому алгоритму, алгоритму роя частиц и алгоритму Нелдера – Мида как базовым алгоритмам.

Методы управления параметрами

Методы управления параметрами (*parameter control*) предусматривают решение задачи метаоптимизации в процессе решения базовой задачи оптимизации (с помощью базовой программы). Основная идея состоит в том, что поскольку для каждой из базовых задач оптимальной является, вообще говоря, своя стратегия, поиск этой стратегии необходимо производить в процессе каждого запуска программы, реализующей рассматриваемый базовый алгоритм.

На втором уровне иерархии методы управления параметрами разделяют на методы адаптивного и методы самоадаптивного управления (см. рис. 7.2). Заметим, что в оригинале в рамках данной классификации выделяют еще методы *детерминированного* управления параметрами (*deterministic parameter control*), когда параметры базового алгоритма изменяют по априори заданному правилу. Примером такого подхода к метаоптимизации служит рассмотренное выше правило 1/5 Рехенберга.

В методах *адаптивного* управления параметрами (*adaptive parameter control*) алгоритм метаоптимизации надстраивается над базовым алгоритмом оптимизации и использует информацию, получаемую в процессе функционирования последнего. В роли метода метаоптимизации при этом может выступать какое-либо эвристическое правило. Часто в качестве метаалгоритма в этом случае применяют генетический алгоритм. Например, известны работы, в которых над алгоритмом роя частиц надстраивают генетический алгоритм, который в процессе решения базовой задачи оптимизирует параметры алгоритма роя частиц.

Можно выделить два основных подхода к построению методов *самоадаптивного* управления (*self-adaptive parameter control*) – подход на основе расширения вектора варьируемых параметров исходной задачи и подход на основе коэволюции нескольких субпопуляций.

В первом случае параметры стратегии включают в вектор варьируемых параметров исходной задачи, увеличивая тем самым размерность пространства поиска. Эти же параметры участвуют в формировании фитнес-функции, модифицируя, таким образом, фитнес-функцию исходной задачи. В результате базовый алгоритм начинает выполнять функции метаалгоритма.

Во втором случае используют коэволюцию типа соперничества (см. 6.4), когда субпопуляции отличаются стратегиями поиска. В процессе коэволюции большие ресурсы получают более успешные субпопуляции, тем самым автоматически определяется оптимальная стратегия популяционного алгоритма.

Основное число публикаций, посвященных метаоптимизации на основе управления параметрами, связано с использованием в качестве базового алгоритма того или иного варианта генетического алгоритма. Таким образом, например в качестве оптимальных стратегий, найдены стратегии алгоритма Нелдера – Мида, существенно отличающиеся от стратегий, предложенных его авторами.

Важным преимуществом методов управления параметрами по сравнению с методами настройки параметров является то, что первые методы позволяют относительно просто синтезировать динамические стратегии метаоптимизации.

7.2.2. Классификация Смита – Фогарти

В оригинальной работе Смита и Фогарти данная классификация предложена по отношению к методам метаоптимизации генетических алгоритмов. Однако принципы этой классификации с небольшими изменениями применимы и по отношению к другим популяционным алгоритмам.

Классификационными признаками таксономии Смита – Фогарти являются объект метаоптимизации (*what is being adapted?*) – верхний уровень иерархии; уровень метаоптимизации (*the scope of the adaption*) – второй уровень иерархии; основание для метаоптимизации (*the basis for change*) – нижний уровень иерархии.

Объекты метаоптимизации

Объектами метаоптимизации могут быть миграционные операторы базового алгоритма, а также их параметры.

В первом случае речь может идти, например, о задаче оптимизации вероятностей использования операторов из некоторого их фиксированного набора. Положим, что объектом метаоптимизации является оператор мутации генетического алгоритма o_1 и имеется следующий набор вариантов этого оператора: оператор равномерной случайной мутации $o_{1,1}$, оператор гауссовой случайной мутации $o_{1,2}$, арифметический оператор вещественного сдвига $o_{1,3}$, геометрический оператор вещественного сдвига $o_{1,4}$, степенной оператор мутации $o_{1,5}$ (см. 2.3). Тогда метазадача оптимизации может состоять в определении вероятностей ξ_i, $i \in [1:5]$ применения этих операторов. Здесь ξ_i – вероятность использования оператора $o_{1,i}$; $\sum_{i=1}^{5} \xi_i = 1$.

Уровень метаоптимизации

С этой точки зрения выделяют метаоптимизацию на уровнях популяции, агента и компонентов базового алгоритма.

Метаоптимизация *на уровне популяции* (*population level adaptation*) предполагает поиск оптимальных значений параметров, определяющих функционирование всей популяции в целом, то есть некоторых ее глобальных параметров. Ярким примером адаптации этого уровня является изменение размера популяции с изменением числа поколений на основе некоторой оценки текущей эффективности популяции.

Для генетического алгоритма метаоптимизацию на уровне популяции можно реализовать, например, с помощью алгоритма коэволюции.

Метаоптимизация *на уровне агента (individual level adaptation)* предполагает решение задачи адаптации не на уровне глобальных параметров популяции, а на уровне параметров ее агентов. Если, к примеру, в генетическом алгоритме целью метаоптимизации на уровне популяции является поиск оптимальной вероятности мутации для всей популяции, то на уровне агента целью адаптации может быть определение той же вероятности, но для каждого из агентов в отдельности.

Мотивировать целесообразность метаоптимизации на уровне агента можно тем фактом, что в различных частях области поиска, которая с точки зрения свойств фитнес-функции обычно является гетерогенной, могут быть эффективны различные стратегии агентов. Широко известен подход к адаптации на уровне агента методом самоадаптации (см. ниже).

Метаоптимизация *на уровне компонентов (component-level adaptation)* имеет в виду, скажем, оптимизацию шага мутации генетического алгоритма.

Отметим, что уровень метаоптимизации в значительной мере определяет вычислительные затраты на решение метазадачи. Так, метаоптимизация на уровне компонентов требует минимальных затрат, а адаптация на уровне популяции – максимальных. С другой стороны, эффективность высокоуровневой метаоптимизации может многократно превышать эффективность низкоуровневой оптимизации.

Основание для метаоптимизации

В зависимости от уровня, на котором решается задача метаоптимизации, могут быть использованы различные основания для метаоптимизации.

При адаптации на уровне популяции основанием для запуска процесса метаоптимизации в простейшем случае может быть достижение заданного числа поколений, прошедших после предыдущего акта метаоптимизации. В более развитых алгоритмах метаоптимизации с этой целью используют информацию о текущем распределении индивидов популяции (в частности, среднюю плотность распределения, максимальное расстояние между индивидами и так далее).

В случае адаптации на уровне агента условием необходимости адаптации является, например, понижение текущей приспособленности данного агента ниже уровня средней приспособленности агентов по популяции.

Ограничимся далее классификацией Эйбена.

7.3. ОДНОКРАТНАЯ НАСТРОЙКА ПАРАМЕТРОВ

Выделяют два класса методов однократной настройки параметров – методы однократной настройки для решения статической задачи параметрической метаоптимизации и такие же методы для решения динамической задачи.

7.3.1. Статическая параметрическая метаоптимизация

Как показано в 7.1.1, формально задача статической параметрической метаоптимизации может быть записана в виде (7.5). При практическом решении

этой метазадачи в качестве множества задач Ф используют множество, определяемое набором тестовых целевых функций $F = \{f_i(X) \mid i \in [1:|F|]\}$, для каждой из которых известно точное решение X_i^*, f_i^*. Заметим, что данное требование накладывает сильное ограничение на класс тестовых функций.

Можно предложить два основных метода построения указанного набора функций. Во-первых, этот набор может представлять собой некоторую, достаточно представительную, совокупность «трудных» функций (см. Приложение Б). Во-вторых, набор F может продуцировать одна или несколько параметризованных функций за счет регулярного или случайного изменения их параметров.

Приведем в качестве иллюстрации последнего метода пример построения набора F на основе совокупности некоторого числа квадратичных функций, каждая из которых в общем случае дает локальный минимум искомой функции. Положим, что

$$f_i(X) = \min_{j \in [1:r]} f_{i,j}(X), i \in [1:|F|],$$

где квадратичная функция

$$f_{i,j}(X) = \sum_{k=1}^{|X|} a_{i,j,k}(x_k - x_{i,j,k}^*)^2 + c_{i,j}.$$

Здесь $a_{i,j,k} > 0$, $x_{i,j,k}^*$, $c_{i,j}$ – заданные вещественные константы, $X_{i,j}^* = (x_{i,j,1}^*, x_{i,j,2}^*, ..., x_{i,j,|X|}^*)$ – точка j-го локального минимума со значением $c_{i,j}$.

Генерация набора функций F сводится в данном случае к заданию $r|X||F|$ значений констант $a_{i,j,k}$, $x_{i,j,k}^*$, $c_{i,j}$; $i \in [1:|F|]$, $j \in [1:r]$, $k \in [1:|X|]$. Значения этих констант могут быть получены с помощью генераторов случайных чисел, распределенных по заданному закону в соответствующих областях допустимых значений констант.

Таким образом, решение метазадачи (7.5) сводится к решению $|F|$ задач вида (7.1) и выбору на этой основе стратегии B^*, доставляющей минимум функции приспособленности $\mu(\Phi, B)$. Обычно последнюю задачу решают методом перебора узлов некоторой дискретной сетки, покрывающей множество D_B допустимых стратегий.

Покроем множество D_B сеткой Ω с узлами Ω_I, где $I = (i_1, i_2, ..., i_{|B|})$ – мультииндекс. Общее число узлов сетки обозначим $|\Omega|$. Тогда общую схему однократной настройки параметров методом перебора можно представить в следующем виде.

1) Полагаем $i = j = k = 1$.

2) Выбираем из набора F тестовую функцию $f_i(X)$.

3) Выбираем вектор стратегий B_{I_j}, соответствующий j-му узлу сетки Ω.

4) Генерируем начальную точку $X_{i,j,k}^0 \in \Pi \subset R^{|X|}$, где Π – рассматриваемая часть пространства поиска.

5) Исходя из точки $X_{i,j,k}^0$, с помощью базового алгоритма $a(B_{I_j})$ с заданной точностью ε решаем исходную задачу оптимизации функции $f_i(X)$. Вычисляем значение целевой функции $\mu(X_{i,j,k}^0, f_i, B_{I_j}) = \mu_{i,j,k}$.

6) Если $k \le |X^0|$, то полагаем $k = k + 1$ и переходим к шагу 4. В противном случае определяем величину $\mu_{i,j}$, представляющую собой среднее значение величин $\mu_{i,j,k}$, $k \in [1:|X^0|]$, и выполняем присваивание $k = 1$.

7) Аналогично полагаем $j = j+1$, если $j \le |\Omega|$, и переходим к шагу 3 или определяем величину μ_i (среднее значение величин $\mu_{i,j}, j \in [1:|\Omega|]$) и полагаем $j = 1$.

8) По той же схеме полагаем $i = i+1$, если $i \le |F|$, и переходим к шагу 2. Иначе находим минимальное достигнутое значение целевой метафункции

$$\mu^* = \min_{i \in [1:|F|]} \mu_i,$$

а также соответствующий вектор стратегии B^*, который и принимаем за искомый вектор оптимальной стратегии.

Очевидна модификация рассмотренной схемы, в которой узлы во множестве D_B генерируем случайным образом.

Как отмечалось выше, основной проблемой метода однократной настройки является экспоненциальный рост вычислительных затрат при увеличении числа параметров в стратегии базового алгоритма. Из рассмотренной схемы метода следует, что он требует решения $|X^0||F||\Omega|$ тестовых задач оптимизации. Так что уже при четырех свободных параметрах базового алгоритма и всего пяти возможных значениях каждого из них необходимо $5^4|X^0||F|$ раз решить тестовую задачу оптимизации. Если тестирование производить на 100 задачах из 100 начальных точек, то это потребует $6{,}25 \times 10^6$ запусков базовой программы. Из-за невозможности исчерпывающего исследования полученная в результате стратегия может оказаться далекой от оптимальной.

7.3.2. Динамическая параметрическая метаоптимизация

Динамическая неадаптивная задача

Выше показано, что путем параметризации вектора стратегий $B(t) = (b_1(t), b_2(t), ..., b_{|B|}(t))$ с помощью вектора динамических параметров $\boldsymbol{C} = (C_1, C_2, ..., C_{|B|}) \in \boldsymbol{D}_C$ задача поиска оптимальной стратегии $B^*(t) = B(t, \boldsymbol{C}^*)$ на классе задач Ф может быть записана в виде метазадачи глобальной условной оптимизации (7.10). Данная задача формально идентична задаче (7.5), к которой нами сведена статическая метазадача параметрической оптимизации, и может быть решена по схеме, рассмотренной в предыдущем пункте.

Проблема экспоненциального роста вычислительных затрат с ростом числа параметров в стратегии базового алгоритма для динамической метазадачи оптимизации становится еще более острой, поскольку размерность вектора варьируемых параметров в задаче (7.10) в этом случае оказывается равной $|B||C|$, то есть в $|C|$ раз больше размерности того же вектора в задаче (7.5).

Динамическая адаптивная задача

Задача динамической адаптивной метаоптимизации сводится к отысканию функции $B^*(W(t))$, доставляющей минимум целевой метафункции $\mu(\Phi, B)$. Для решения этой задачи могут быть использованы подходы на основе генетического программирования и так называемого сетевого оператора.

Алгоритм *генетического программирования* (*Genetic Programming*) относится к классу эволюционных алгоритмов. Алгоритм предложил в 1992 г. профессор Стенфордского университета Коза (*J. Koza*) как метод автоматического синтеза алгоритмов или математических выражений. Алгоритм генетического программирования имеет такие известные недостатки, как необходимость использования лексического анализатора при декодировании польской записи, сложность ограничения области поиска и другие.

Метод сетевого оператора предложен А. И. Дивеевым (ВЦ РАН) и предназначен для решения той же задачи автоматического синтеза алгоритмов или математических выражений. Метод свободен от отмеченных недостатков алгоритма генетического программирования. Искомую, вообще говоря, нелинейную функцию $b_i^*(W(t))$, $i \in [1:|B|]$ в методе сетевого оператора отыскивают в виде совокупности унарных функций (например, $\exp(\cdot)$, $\sin(\cdot)$) и бинарных функций (например, $+(\cdot,\cdot)$, $\times(\cdot,\cdot)$), аргументами которых являются функции, заданные константы или переменная t. Поиск оптимальной совокупности указанных функций осуществляют путем решения с помощью генетического алгоритма одно- или многоцелевой задачи глобальной оптимизации. С той же целью может быть использован любой из рассмотренных популяционных алгоритмов.

Для отыскания динамической адаптивной стратегии метаоптимизации в виде $B(t, W(t))$ также могут быть использованы алгоритм генетического программирования и метод сетевого оператора.

7.4. ПЕРМАНЕНТНАЯ НАСТРОЙКА ПАРАМЕТРОВ

Представим один из возможных подходов к организации перманентной настройки параметров, который может быть использован для решения как статической, так и динамической задач параметрической метаоптимизации. Ограничимся рассмотрением первой из указанных задач.

7.4.1. Схема подхода

Введем в рассмотрение $(|Q| \times 1)$-*вектор атрибутов задачи* $\phi = \phi(Q) \in \Phi$. Совокупность всех возможных значений этого вектора обозначим D_Q. Атрибуты Q можно разделить на *функциональные атрибуты*, полностью определяемые математическими свойствами функции $f(X)$ (константой Липшица этой функции, например); *вычислительные атрибуты*, определяемые вычислительной сложностью функции $f(X)$ и, быть может, ее производных; *точностные атрибуты*, определяющие требуемую точность решения задачи.

Значения всех атрибутов $q_1, q_2, ..., q_{|Q|}$ задачи ϕ должны быть заданы до начала решения метазадачи оптимизации. Значения некоторых атрибутов могут уточняться в процессе решения задачи ϕ. Назовем последние атрибуты *апостериорными* и обозначим $Q_1 = (q_{i_1}, q_{i_2}, ...)$. Остальные компоненты вектора Q назовем *априорными* и обозначим $Q_2 = (q_{j_1}, q_{j_2}, ...)$. Таким образом, имеет место равенство

$$Q = Q_1 \cup Q_2.$$

Во введенных соглашениях и обозначениях схема подхода имеет следующий вид.

1) В процессе нормальной эксплуатации базовой программы (то есть в процессе решения задач класса Ф) формируем и непрерывно пополняем множество D_Q.

2) Производим сначала *кластеризацию*, а затем, периодически, *рекластеризацию* этого множества – выделяем кластеры D_Q^i, $i \in [1:n_c]$.

3) Каждому из кластеров D_Q^i ставим в соответствие стратегию B_i^*.

4) Для задачи $\phi(Q_j) \in \Phi$ среди кластеров D_Q^i выбираем кластер $D_Q^{i_j}$, которому принадлежит вектор Q_j.

5) Соответствующую стратегию $B_{i_j}^*$ по некоторому правилу варьируем и полагаем оптимальной стратегией базового алгоритма для данной задачи $\phi(Q_j)$.

Из приведенной схемы следует, что реализация подхода требует решения двух следующих основных задач: кластеризация множества D_Q; оптимизация стратегий B_i^* (то есть собственно метазадача оптимизации). Число рекластеризаций множества D_Q назовем числом эр.

7.4.2. Задачи кластеризации и собственно метаоптимизации

Рассмотрим прежде *задачу кластеризации*. Классические подходы к задаче кластеризации многомерных векторов основаны на сравнении векторов между собой в смысле некоторой меры близости, которой называют величину, имеющую предел и возрастающую с увеличением сходства объектов. В качестве меры близости числовых векторов часто используется евклидово расстояние между ними.

Большое распространение получили алгоритмы кластеризации, основанные на оптимизации некоторой функции, характеризующей качество кластеризации. Среди данных алгоритмов выделяют алгоритмы семейства *k-средних*. В этом случае указанная функция представляет собой сумму квадратов взвешенных отклонений координат векторов от центров искомых кластеров, которые, полагается, имеют сферическую или эллиптическую форму.

Современными перспективными алгоритмами решения задачи кластеризации являются алгоритмы, построенные на основе *самоорганизующихся карт Кохонена SOM* (*T. Kohonen, Self-Organizing Map*). Карта Кохонена представляет собой соревновательную нейронную сеть с обучением без учителя, которая, кроме решения задачи кластеризации, решает также и задачу визуализации результатов кластеризации (методом проецирования многомерного пространства в пространство с более низкой размерностью).

Чаще всего сети Кохонена реализуют с помощью однослойной нейронной сети, в которой каждый нейрон получает все компоненты входного вектора $Q = (q_1, q_2, ..., q_{|Q|})$ (рис. 7.3).

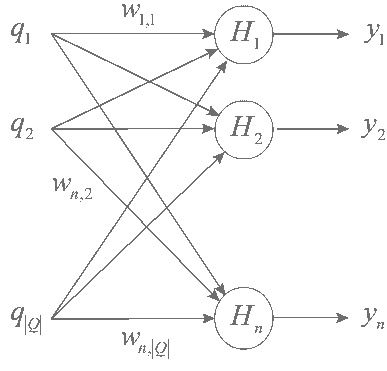


Рис. 7.3. Структура однослойной сети *SOM*: $W_i = (w_{i,1}, w_{i,2}, ..., w_{i,|Q|})$ – вектор входных весов нейрона H_i; $n = n_{claster}$

На начальном этапе функционирования программной системы, реализующей рассматриваемый подход, когда мощность множества D_Q мала, может быть использован только один кластер. С ростом мощности этого множества число кластеров следует увеличивать. Поскольку каждое увеличение числа кластеров требует значительных вычислительных затрат на кластеризацию, интервал между переобучениями целесообразно увеличивать по некоторому закону.

Перейдем к рассмотрению собственно *метазадачи оптимизации*. К метаалгоритму в данном случае уместно предъявить следующие требования. Поскольку для популяционных алгоритмов оптимизации характерна высокая размерность вектора стратегий B, метаалгоритм должен успешно функционировать в пространствах высокой размерности. Вследствие того, что метаоптимизация производится в процессе нормальной эксплуатации базового программного обеспечения, метаалгоритм должен обеспечивать быстрое получение приближенно оптимальных стратегий, то есть высокую скорость сходимости. Метаалгоритм должен быть алгоритмом глобальной оптимизации.

Указанным требованиям удовлетворяют, в частности, эволюционные алгоритмы оптимизации, из всего многообразия которых используем генетический алгоритм как наиболее развитый представитель этих алгоритмов.

Поскольку пространство поиска в метазадаче оптимизации может иметь высокую размерность, традиционный двоичный алфавит для кодирования хромосом в этих условиях может приводить к неприемлемо большим размерам хромосом. Поэтому в данном случае более разумным является использование вещественного кодирования, то есть использование непрерывного генетического метаалгоритма.

7.4.3. Особенности программной реализации подхода

Программная реализация подхода целесообразна в рамках клиент-серверной парадигмы, которая позволяет перенести вычислительные затраты, связанные с метаоптимизаций, на сервер приложений. Целесообразно также использование трехзвенной архитектуры программного обеспечения, включающей в себя рабочие станции, сервер приложений и сервер баз данных. Такая архитектура обеспечивает хорошую масштабируемость программной системы, малую дополнительную нагрузку на рабочие станции клиентов, надежность и высокую безопасность данных.

Клиентская и серверная программы должны предусматривать обработку как минимум следующих исключительных ситуаций.

• Пользователь перед запуском базовой программы оптимизации отказывается от метаоптимизации. Возможной реакцией на данную ситуацию может быть использование клиентской программой «умолчательных» значений параметров базового алгоритма («умолчательной» стратегии).

• Соединение клиентской программы с серверной программой прерывается по тайм-ауту: клиентская программа запускает базовую программу с «умолчательной» стратегией базового алгоритма.

• В момент обращения клиентской программы к серверной программе последняя осуществляет поиск нового поколения стратегий: серверная программа возвращает клиентской программе лучшую стратегию текущего поколения.

• В момент обращения клиентской программы серверная программа осуществляет рекластеризацию данных: серверная программа возвращает клиентской программе «умолчательную» стратегию базового алгоритма.

После задания пользователем значений атрибутов Q решаемой задачи $\phi \in \Phi$ клиентская и серверная части программного обеспечения выполняют следующие основные действия.

1) Клиентская программа посылает серверной программе вектор Q.

2) Получив указанный вектор, серверная программа определяет кластер D_Q^i, которому принадлежит вектор Q, выбирает соответствующую этому кластеру стратегию B_i^*, варьирует ее и отсылает клиентской программе измененную стратегию B_i^{**}.

3) Клиентская программа, получив стратегию B_i^{**}, запускает процесс решения задачи $\phi(Q)$ с помощью алгоритма $a(B_i^{**})$.

4) Завершив решение задачи $\phi(Q)$, клиентская программа формирует вектор Q_1 значений апостериорных атрибутов этой задачи и отсылает его серверной программе.

5) Получив вектор Q_1, серверная программа фиксирует его в своей базе данных.

Серверная программа должна обеспечивать параллельное выполнение следующих функций: кластеризация и рекластеризация данных; решение собственно метазадачи оптимизации; прием и обработка запросов клиентских приложений. База данных программы может включать в себя следующие основные таблицы.

Таблица стратегий, содержащая стратегии-индивиды, которые серверная программа передает клиентской программе для запуска базового алгоритма оптимизации и которые алгоритм метаоптимизации использует с целью формирования новых поколений стратегий.

Таблица эффективностей стратегий, предназначенная для накопления результатов использования различных стратегий из предыдущей таблицы. На основе данных из этой таблицы алгоритм метаоптимизации вычисляет значение фитнес-функции каждого агента и формирует новые поколения.

Таблица счетчиков, в которой хранятся данные о числе запусков базового алгоритма с каждой из стратегий, содержащихся в таблице стратегий. По достижении определенного числа использований каждой из стратегий данного поколения должен быть запущен алгоритм метаоптимизации для формирования нового поколения.

Таблица апостериорной информации, хранящая сведения обо всех ранее решенных задачах. Эти сведения используются для кластеризации и рекластеризации задач.

Клиентская программа должна оценивать значения апостериорных атрибутов задачи. Для реализации этой функции может оказаться необходимым ведение клиентской программой локальной базы данных, содержащей значения

компонент вектора варьируемых параметров X во всех точках испытаний, а также соответствующие значения целевой функции $f(X)$.

7.4.4. Исследование эффективности подхода

Исследование эффективности рассматриваемого подхода к перманентной параметрической метаоптимизации выполнено на основе библиотечной реализации алгоритма Нелдера – Мида в программном комплексе *MATLAB*. Для обеспечения возможности варьирования свободных параметров (стратегии) этого алгоритма выполнена модификация указанной библиотечной программы. Эта модификация обеспечивает также оценку таких апостериорных атрибутов задачи, как константа Липшица функции $f(X)$, число ее локальных минимумов и некоторых других атрибутов.

В качестве свободных параметров алгоритма Нелдера – Мида использованы коэффициенты отражения α, редукции β, сжатия γ и растяжения δ (см. 3.2.5). Таким образом, стратегию алгоритма Нелдера – Мида $a(B)$ определяет четырехмерный вектор $B=(b_1,b_2,b_3,b_4)$, где $b_1=\alpha$, $b_2=\beta$, $b_3=\gamma$, $b_4=\delta$. Множество допустимых стратегий имеет вид

$$D_B=\{b_i \mid b_i^- \le b_i \le b_i^+, i\in[1:4]\},$$

где $b_1^-=0{,}9$; $b_1^+=1{,}0$; $b_2^-=0$; $b_2^+=1{,}0$; $b_3^-=0$; $b_3^+=1{,}0$; $b_4^-=2{,}0$; $b_4^+=3{,}0$.

Класс задач $\Phi=\Phi_{test}$ построен на основе совокупности многомерных параметризованных тестовых функций Химмельблау, Розенброка, Шаффера, «сфера», step, Экли, Растригина, Шекеля (Приложение Б). Свободные параметры указанных функций принимали случайные значения, равномерно распределенные в соответствующих интервалах.

В качестве целевой метафункции $\mu(\Phi, B)$ использована аддитивная свертка следующих функций: оценка математического ожидания числа испытаний; оценка среднего квадратичного отклонения этого числа; оценка вероятности нелокализации минимума рассматриваемой функции $\phi(X)\in\Phi_{test}$ за максимально допустимое число испытаний.

Принято, что трехмерный вектор Q атрибутов задач $\phi(Q)\in\Phi_{test}$ включает в себя следующие величины: q_1 – константа Липшица целевой функции $f(X)$; q_2 – число локальных минимумов этой функции; q_3 – заданная точность решения задачи. Из числа указанных атрибутов константа Липшица q_1 и число локальных минимумов q_2 являются апостериорными атрибутами.

Для решения собственно метазадачи оптимизации использован непрерывный генетический алгоритм с *SBX*-кроссовером (п. 2.4.2). Размер популяции $|S|$ генетического алгоритма положен равным 20. Кластеризация множества значений атрибутов D_Q реализована с помощью самоорганизующейся карты Кохонена.

Как показывают наши исследования, эффективность подхода в значительной степени зависит от двух следующих параметров: n_{start} – число запусков базового алгоритма $a(B)$ со стратегией, определяемой каждым данным агентом; n_{gen} – число поколений каждой из стратегий B_i^*, необходимых для начала процедуры реструктуризации множества D_Q. Принято, что $n_{start}=15$, $n_{gen}=10$.

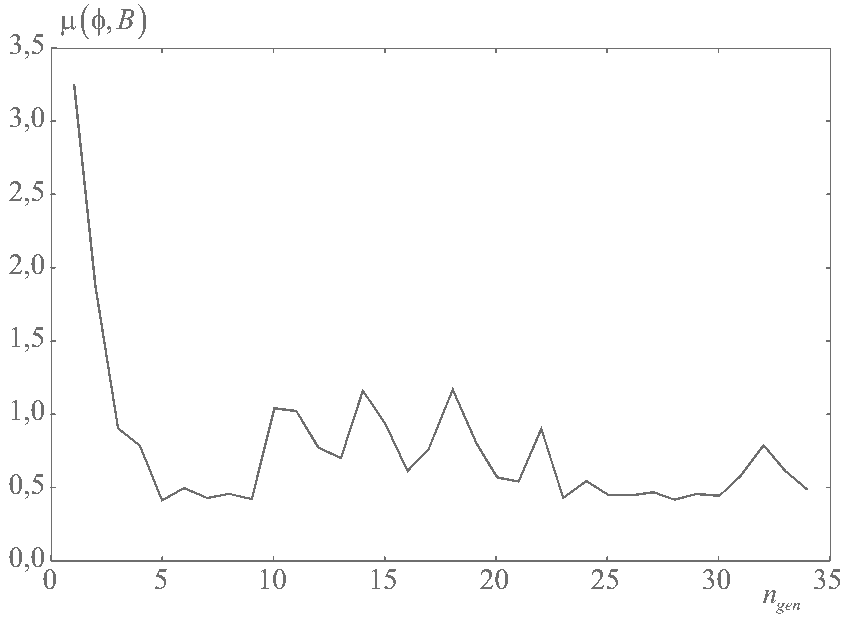


Рис. 7.4. Динамика развития одного из кластеров

Приведенные ниже результаты исследования получены на основе решения более 150 тысяч задач из множества задач Φ_{test}. Число кластеров D_Q^i, выделенных во множестве D_Q, к моменту завершения исследования достигло $n_{claster} = 14$. При этом максимальное поколение, до которого дошло развитие стратегий, было равно $n_{gen} = 35$, то есть к моменту завершения исследования наиболее развитый (чаще всего используемый) кластер прошел 35 смен поколений.

Обозначим B^r стратегию алгоритма Нелдера – Мида, рекомендованную его авторами, и назовем рекомендованной стратегией.

Динамику развития одного из кластеров D_Q^i иллюстрирует рис. 7.4. График показывает, что с ростом числа поколений n_{gen} значение целевой метафункции $\mu(\Phi, B)$ быстро уменьшается, то есть генетический алгоритм метаоптимизации для данного кластера быстро находит стратегию, близкую к оптимальной стратегии B_i^*.

Отметим, что поскольку алгоритм метаоптимизации создает начальные стратегии случайным образом, в начале развития кластеров в среднем требуется большее число испытаний для решения задач $\phi \in \Phi_{test}$, чем базовому алгоритму с рекомендованной стратегией. Аналогично вначале имеет место меньшая вероятность локализации минимума функций $f(X)$. Ситуация меняется на противоположную всего после нескольких поколений генетического метаоптимизирующего алгоритма.

Спустя 14 эр развития кластеров D_Q^i было выполнено исследование эффективности найденных оптимальных стратегий B_i^*. Использовался набор Φ_{val} из 100 задач, случайным образом выбранных из множества Φ_{test}. Результаты исследования иллюстрирует рис. 7.5. С целью сравнения для тех же задач на рисунке приведены результаты, полученные для рекомендованной стратегии B^r. Алгоритмы $a(B^*)$, $a(B^r)$ запускались из одинаковых начальных точек,

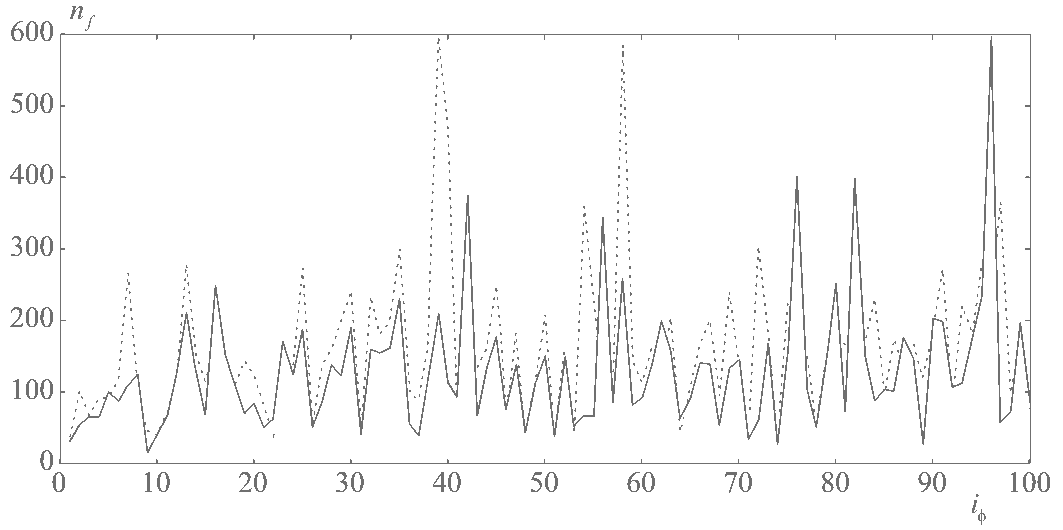


Рис. 7.5. К исследованию эффективности оптимальных стратегий, $i_ф$ – номер задачи из набора Φ_{val}: —— стратегия B^*; – – – стратегия B^r

с одинаковой требуемой точностью решения. Интегральные результаты данного исследования представлены в табл. 7.1.

Рис. 7.5 иллюстрирует тот факт, что вычислительная эффективность алгоритма Нелдера – Мида при использовании стратегий B_i^*, как правило, выше той же эффективности при использовании рекомендованной стратегии B^r. Табл. 7.1 показывает, что в среднем оптимальные стратегии B_i^* на 19 % эффективнее рекомендованной стратегии B^r.

Спустя 14 эр развития кластеров D_Q^i было проведено также исследование эффективности лучших стратегий B_i^{**} этих кластеров. Аналогично предыдущему исследованию использован набор Φ'_{val} из 100 случайных задач множества Φ_{test}. Некоторые результаты исследования представлены на рис. 7.6, а интегральные результаты – в табл. 7.2.

Последние представленные результаты показывают, что лучшие стратегии B_i^{**}, найденные метаалгоритмом, на 46 % эффективнее рекомендованной стратегии B^r.

Сделаем несколько заключительных замечаний.

Во-первых, клиент-серверная архитектура рассмотренного программного обеспечения вносит в систему накладные расходы, обусловленные коммуникациями между клиентской и серверной частями. Для задач, имеющих невысо-

Таблица 7.1 **Интегральные результаты исследования оптимальных стратегий**

Число испытаний n_f	Рекомендованная стратегия B^r	Оптимальная стратегия B^*
Среднее	161	130
Минимальное	35	16
Максимальное	600	600

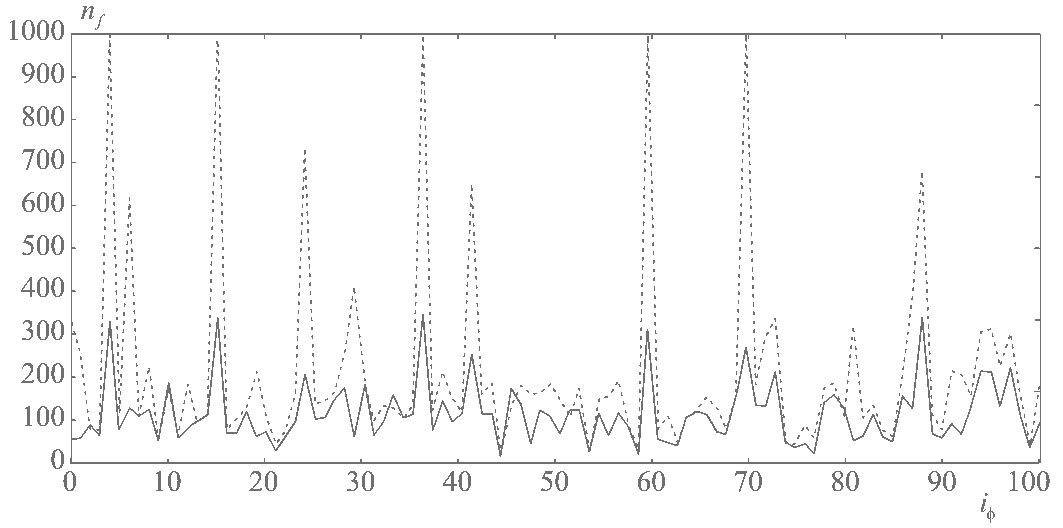


Рис. 7.6. К исследованию эффективности лучших стратегий, i_ϕ – номер задачи из набора Φ'_{val}: —— стратегия B^{**}; – – – стратегия B^r

кую вычислительную сложность, это обстоятельство может приводить к неэффективности метаоптимизации, поскольку выигрыш в числе испытаний может быть перевешен накладными расходами. Однако в задачах, представляющих практический интерес, типичной является высокая вычислительная сложность целевой функции, обеспечивающая малый удельный вес указанных коммуникационных расходов. Как следует из результатов исследования, в такой ситуации метаоптимизация базового алгоритма оптимизации может обеспечить значительный положительный эффект.

Во-вторых, в качестве базовой задачи оптимизации здесь рассмотрена задача безусловной оптимизации. Для решения задач условной оптимизации в вычислительной практике широко используют методы штрафных и барьерных функций, сводящие задачу условной оптимизации к совокупности задач безусловной оптимизации (см. 1.3.1). Таким образом, полученные результаты представляют интерес и для задач условной оптимизации. С другой стороны, имеется большое число методов решения задач условной оптимизации, которые не используют сведение ее к задаче безусловной оптимизации. Это обстоятельство делает актуальной модификацию предложенного подхода и программного обеспечения для решения задач условной оптимизации как самостоятельного класса задач.

Таблица 7.2 **Интегральные результаты исследования лучших стратегий**

Число испытаний n_f	Рекомендованная стратегия B^r	Лучшая стратегия B^{**}
Среднее	213	115
Минимальное	24	13
Максимальное	1000	350

В-третьих, метазадача параметрической оптимизации базового алгоритма представляет собой задачу многоцелевой оптимизации. В рассмотренном подходе для решения этой задачи использован простейший метод ее сведения с помощью аддитивной скалярной свертки к одноцелевой задаче. Представляется перспективной модификация предложенного подхода и программного обеспечения, использующая более тонкие методы решения задачи многоцелевой оптимизации.

7.5. АДАПТИВНОЕ УПРАВЛЕНИЕ ПАРАМЕТРАМИ

Выше показано, что в методах адаптивного управления параметрами метаоптимизация может строиться на основе некоторых эвристических правил, на основе сведения к задаче оптимального управления, а также на основе коэволюционных алгоритмов. Ограничимся рассмотрением последнего случая.

Введем следующие обозначения. Пусть $a(B)$ – рассматриваемый базовый популяционный алгоритм оптимизации со стратегией $B \in D_B$. Вектор варьируемых параметров алгоритма $a(B)$ есть $X = (x_1, x_2, ..., x_{|X|})$, а целевая функция – $f(X)$. Аналогично пусть m – метаалгоритм, варьируемые параметры которого представляют собой вектор $B = (b_1, b_2, ..., b_{|B|})$, а $\mu(\phi, B)$ – целевая метафункция.

В коэволюционных методах адаптивного управления параметрами алгоритму $a(B)$ ставится в соответствие совокупность *субпопуляций* $\boldsymbol{S}^a = \{S_i^a, i \in [1:|\boldsymbol{S}^a|]\}$, в которой субпопуляция S_i^a реализует алгоритм $a(B)$ с допустимой стратегией $B_i \in D_B$. Метаалгоритм m реализует эволюцию *суперпопуляции* $S^m = (s_1^m, s_2^m, ..., s_{|S^m|}^m)$, число агентов которой $|S^m|$ равно числу субпопуляций $|\boldsymbol{S}^a|$, так что агенту $s_i^m \in S^m$ соответствует субпопуляция $S_i^a \in |\boldsymbol{S}^a|$, стратегию которой $B_i \in D_B$ он определяет. В качестве целевой метафункции $\mu(\phi, B)$ популяции S^m используем, например лучшее текущее значение фитнес-функции $\varphi(X)$, достигнутое популяциями S_i^a.

Во введенных обозначениях общую схему коэволюционного адаптивного управления параметрами алгоритма $a(B)$ можно представить в следующем виде.

1) Инициализируем субпопуляции $S_i^a, i \in [1:|\boldsymbol{S}^a|]$ и суперпопуляцию S^m.

2) Параллельно, как минимум на логическом уровне, запускаем итерационные процессы, реализующие эволюцию популяций $\{S_i^a\}$, S^m.

3) Периодически по синхронной или асинхронной схеме заменяем текущие стратегии B_i новыми координатами соответствующих агентов s_i^m.

4) Итерационный процесс завершаем по выполнении в субпопуляциях S_i^a условий окончания итераций.

Таким образом, схема взаимодействия популяций $\{S_i^a\}$, S^m представляет собой пример коалгоритмической гибридизации алгоритмов $a(B_i)$, m (см. 6.4).

Можно выделить два класса коэволюционных методов адаптивного управления параметрами базового алгоритма: методы, использующие гомогенные алгоритмы $a(B)$, m; методы, в которых эти алгоритмы являются гетерогенными. Назовем методы первого класса *гомогенными*, а второго класса – *гетерогенными* коэволюционными методами адаптивного управления параметрами.

7.5.1. Гомогенные методы

В гомогенных коэволюционных методах адаптивного управления параметрами алгоритм $a(B)$ и метаалгоритм m относятся к одному и тому же классу популяционных алгоритмов. Известны примеры коэволюций данного класса, в которых указанные алгоритмы являются генетическими алгоритмами, эволюционными алгоритмами, алгоритмами роя частиц и так далее. Рассмотрим в качестве примера гомогенный алгоритм адаптивного управления параметрами на основе алгоритма роя частиц. Алгоритм предложен и исследован немецким ученым Майсснером (*M. Meissner*) и другими в 2006 г.

В качестве метаалгоритма m авторы использовали канонический алгоритм роя частиц со статическими параметрами $b_I = 0{,}5; b_C = 2{,}0; b_S = 2{,}0$.

В алгоритмах $a(B)$ использован модифицированный канонический алгоритм роя частиц. В отличие от статических параметров b_C, b_S, здесь «инерционный» параметр b_I изменяется в процессе итераций по формуле

$$b_I(t) = b_I^b - \frac{b_I^b - b_I^e}{\hat{t}}\, t,$$

где b_I^b, b_I^e – значения этого параметра в момент начала итераций ($t = 0$) и в момент их окончания ($t = \hat{t}$). Таким образом, стратегию B определяет вектор параметров b_I^b, b_I^e, b_C, b_S, то есть $b_1 = b_I^b; b_2 = b_I^e; b_3 = b_C; b_4 = b_S$, и размерность вектора варьируемых параметров алгоритма m равна $|B| = 4$. Принято, что область D_B допустимых значений компонент этого вектора имеет вид

$$D_B = \{b_i \mid b_i \in (0; 4), i \in [1:4]\}.$$

Популяция S^m, реализующая метаалгоритм m, названа авторами *суперроем* (*super-swarm*). Популяция содержит 30 индивидов, так что число популяций S_i^a, реализующих алгоритм $a(B)$ и названных авторами *суброями* (*sub-swarms*), равно $|S^a| = 30$. Каждая из субпопуляций S_i^a содержит по 20 частиц: $|S_i^a| = |S^a| = 20$.

Значением целевой метафункции $\mu(\phi, B)$ задачи оптимизации, которую решает метаалгоритм m, является величина φ^{best} – лучшее значение фитнес-функции базовой задачи, достигнутое алгоритмами $a(B_i)$. Если, например, речь идет о максимизации значений фитнес-функции, то

$$\varphi^{best} = \varphi^{best}(\phi, B) = \max_{i \in [1:|S^a|]} \tilde{\varphi}_i^*,$$

где $\tilde{\varphi}_i^*$ – текущее лучшее значение фитнес-функции, достигнутое субпопуляцией S_i^a.

Исследование эффективности данного алгоритма метаоптимизации выполнено на совокупности задач Ф, включающей в себя пять следующих известных тестовых функций (Приложение Б): унимодальные функции «сфера» и Розенброка; многоэкстремальные функции Растригина, Шаффера и Гриванка. Размерность вектора варьируемых параметров $|X|$ всех функций, кроме функции Шаффера, которая является двумерной, была принята равной 30. Максимальное число итераций $\hat{t}^m$ суперроя равнялось 100, а суброев $\hat{t}^a$ – 1000. Каждая из указанных тестовых задач была решена 400 раз, исходя из различных стартовых точек.

Найденные оптимальные стратегии B представлены в табл. 7.3, которая показывает значительное отличие оптимальных стратегий от их рекомендованных канонических значений.

Таблица 7.3 **Оптимальные стратегии алгоритма роя частиц**

Оптимальная стратегия	Тестовая функция				
	Шаффера	Гриванка	Растригина	Розенброка	«сфера»
b_I^b	–0,19	0,68	0,76	0,08	0,147
b_I^e	1,57	0,18	0,85	0,63	0,070
b_C	0,66	1,87	1,89	1,20	0,984
b_S	0,48	2,21	0,40	2,57	2,710

Эффективность найденных оптимальных стратегий иллюстрирует табл. 7.4, в которой принято, что $\varepsilon_{\max}$, $\overline{\varepsilon}$, $\sigma(\varepsilon)$ – максимальная ошибка, а также оценки ее математического ожидания и среднего квадратичного отклонения соответственно; $\varepsilon = abs\left(f^* - \tilde{f}^*\right)$. В виде дроби приведены значения указанных величин, полученные для канонической (числитель) и оптимальной (знаменатель) стратегий.

Таблица 7.4 **Эффективность оптимальных стратегий алгоритма роя частиц**[1]

Ошибка	Тестовая функция				
	Шаффера	Гриванка	Растригина	Розенброка	«сфера»
$\varepsilon_{\max}$	$\frac{0{,}0042}{0{,}0030}$	$\frac{0{,}827}{0{,}024}$	$\frac{99{,}5}{46{,}5}$	$\frac{91{,}5}{37{,}4}$	$\frac{4{,}14}{6{,}17\cdot 10^{-8}}$
$\overline{\varepsilon}$	$\frac{0{,}0048}{0{,}0045}$	$\frac{0{,}361}{0{,}040}$	$\frac{27{,}0}{13{,}1}$	$\frac{47{,}2}{24{,}2}$	$\frac{6{,}56}{5{,}53\cdot 10^{-7}}$
$\sigma(\varepsilon)$	$\frac{0}{1{,}91\cdot 10^{-8}}$	$\frac{0{,}914}{0{,}015}$	$\frac{98{,}2}{44{,}8}$	$\frac{85{,}1}{25{,}8}$	$\frac{1{,}89}{1{,}02\cdot 10^{-9}}$

[1] Числитель – каноническая стратегия, знаменатель – оптимальная стратегия

7.5.2. Гетерогенные методы

В гетерогенных коэволюционных методах адаптивного управления параметрами алгоритмы $a(B)$, m относятся к разным классам популяционных алгоритмов. Рассмотрим в качестве примера гетерогенный алгоритм, в котором алгоритм $a(B)$ представляет собой канонический алгоритм роя частиц со стратегией $B = (b_I, b_C, b_S)$, а алгоритм m – один из вариантов генетического алгоритма.

Хромосома $H = (H_i^g, i \in [1:|B|])$ метаалгоритма m содержит в этом случае три вещественных гена, так что гены H_1^g, H_2^g, H_3^g кодируют значения

параметра b_I, b_C, b_S соответственно. Как и в рассмотренном выше гомогенном методе, число субпопуляций S_i^a, реализующих алгоритм $a(B)$, равно числу агентов суперпопуляции S^m, то есть $\left|S^a\right| = \left|S^m\right|$; $\mu(\phi, B) = \varphi^{best}(\phi, B)$.

7.6. САМОАДАПТИВНОЕ УПРАВЛЕНИЕ ПАРАМЕТРАМИ

Как отмечалось выше, можно выделить два подхода к самоадаптивному управлению – подход на основе расширения вектора варьируемых параметров исходной задачи и коэволюционный подход.

7.6.1. Расширение вектора варьируемых параметров базовой задачи

При построении самоадаптивных популяционных алгоритмов данный подход используют чаще всего. Пусть, как и ранее, $a(B)$ – рассматриваемый базовый популяционный алгоритм оптимизации со стратегией $B = (b_1, b_2, ..., b_{|B|})$. Вектор варьируемых параметров задачи $\phi \in \Phi$ есть $X = (x_1, x_2, ..., x_{|X|})$, а фитнес-функция – $\varphi(X)$.

Введем в рассмотрение $((|X| + |B|) \times 1)$-вектор

$$\bar{X} = (X, B) = (x_1, x_2, ..., x_{|X|}, b_1, b_2, ..., b_{|B|}).$$

В случае, когда речь идет о метаоптимизации генетического алгоритма, первую часть соответствующей мультихромосомы $\bar{H}$ называют *экзоном* (*exon*), а вторую часть – *интроном* (*intron*). Заметим, что в молекулярной биологии экзоном называют участок гена, несущий информацию о первичной структуре белка, а интроном – участок ДНК, который является частью гена, но не содержит информации о последовательности аминокислот белка.

Во введенных обозначениях суть рассматриваемого метода самоадаптивной метаоптимизации состоит в том, что популяция $S = (s_i, i \in [1:|S|])$ реализует рассматриваемый базовый популяционный алгоритм $a(B)$, используя в качестве координат своих агентов вектор $\bar{X}$. При этом первая и вторая части этого вектора эволюционируют в соответствии с функциями приспособленности $\varphi(X)$ и $\mu(\phi, B)$ соответственно.

В качестве простейшего примера рассмотрим, как это было сделано в п. 7.1.2, генетический алгоритм и оператор гауссовой мутации с шагом мутации σ. Можно предложить два варианта расширения вектора X.

Положим прежде, что $\bar{X} = (X, \sigma) = (x_1, x_2, ..., x_{|X|}, \sigma)$, и поставим в соответствие этому вектору вещественную мультихромосому $\bar{H}$, первая часть которой кодирует вектор X, а вторая – скаляр σ. Размерность представления индивидов в результате увеличивается на единицу и становится равной $|X| + 1$.

Генетические операторы в этом случае применяют как к компонентам вектора X, так и к скаляру σ, так что и шаг мутации проходит эволюционное развитие. Таким образом, компоненты вектора $\bar{X}_i$, $i \in [1:|S|]$ эволюционируют по правилам

$$x'_{i,j} = x_{i,j} + N_1(0, \sigma'),\ j \in [1:|X|],$$
$$x'_{i,|X|+1} = \sigma' = x_{i,|X|+1} \exp\left(N_1(0, b_\sigma)\right),$$

где b_σ – среднее квадратичное отклонение величины σ (рекомендуемое значение $b_\sigma = 1/\sqrt{|X|}$).

Несколько более сложным примером является использование рассмотренного механизма применительно к каждой из компонент вектора X в отдельности.

Пусть $\bar{X} = (x_1, x_2, ..., x_{|X|}, \sigma_1, \sigma_2, ..., \sigma_{|X|})$, и первая часть вещественной мультихромосомы $\bar{H}$ кодирует вектор X, а вторая – вектор шагов мутации $(\sigma_1, \sigma_2, ..., \sigma_{|X|})$, так что размерность представления индивидов равна $2|X|$. В этом случае эволюцию популяции определяют выражения

$$x'_{i,j} = x_{i,j} + N_1(0, \sigma'_j),$$
$$x'_{i,|X|+j} = \sigma'_i = x_{i,|X|+j} \exp\left(N_1(0, b_\sigma)\right),$$

где $i \in [1:|S|]$, $j \in [1:|X|]$. В результате оказывается, что каждая компонента вектора X имеет свой шаг мутации, и потенциал адаптации повышается.

Еще один пример самоадаптивной метаоптимизации на основе расширения вектора варьируемых параметров представляет собой самоадаптирующийся мультимемный алгоритм (п. 5.7.3).

7.6.2. Коэволюция типа соперничества

Данный подход к конструированию самоадаптивных популяционных алгоритмов предполагает формирование достаточно представительного набора субпопуляций, имеющих различные стратегии. В идеале число субпопуляций равно числу элементов множества D_B (или его дискретной аппроксимации). В случае параметрической метаоптимизации это может привести к слишком большому числу субпопуляций и к неприемлемым вычислительным затратам. Поэтому коэволюционную самоадаптацию в основном используют для структурной метаоптимизации. Например, известен коэволюционный подход к адаптивному выбору мемов в меметическом алгоритме.

На разных стадиях решения базовой задачи оптимальными в общем случае являются разные субпопуляции, то есть различные стратегии рассматриваемого популяционного алгоритма. В процессе самоадаптации коалгоритм находит эти стратегии или, по крайней мере, приближается к ним. Таким образом, можно сказать, что коалгоритм реализует поиск приближенно оптимальных законов изменения свободных параметров базового алгоритма, то есть поиск его динамической стратегии.

7.7. СТРУКТУРНАЯ МЕТАОПТИМИЗАЦИЯ

По аналогии с параметрической метаоптимизацией выше были выделены статическая и динамическая структурная и структурно-параметрическая мета-

оптимизация. Ограничимся рассмотрением структурной метаоптимизации. Структурно-параметрическую метаоптимизацию можно реализовать посредством комбинирования подходов, представленных в п. 7.1.1, и результатов, полученных в параграфах 7.3–7.6.

Статическая структурная метаоптимизация

С формальной точки зрения специфика метазадачи статической структурной оптимизации по сравнению с такой же задачей параметрической оптимизации состоит лишь в том, что множество допустимых стратегий D_O, в отличие от множества D_B, является дискретным. Поэтому, вообще говоря, задача статической структурной метаоптимизации может быть решена методами, аналогичными методам однократной и перманентной настройки параметров. Так, задача (7.12) сводится к решению $|F|$ задач вида (7.11) и выбору на этой основе стратегии O^*, доставляющей минимум целевой метафункции $\mu(\Phi, O)$. При этом вместо перебора допустимых стратегий, соответствующих узлам Ω_l сетки Ω, необходим перебор всех элементов множества D_O.

Динамическая структурная метаоптимизация

Аналогично параметрической метаоптимизации, методы решения динамической метазадачи структурной оптимизации существенно зависят от того, идет ли речь о решении задачи вида (7.9) или задачи вида (7.10). В первом случае решение возможно только методами адаптации или самоадаптации, а во втором случае – методами настройки параметров.

Рассмотрим для примера задачу динамической структурной метаоптимизации генетического алгоритма.

Пусть операторы $o_i, i \in [1:3]$ представляют собой операторы мутации, кроссовера и отбора, то есть $|O| = 3$.

В качестве вариантов оператора мутации o_1 рассмотрим стационарный оператор мутации $o_{1,1}$, оператор случайной мутации $o_{1,2}$, гауссов мутатор $o_{1,3}$, арифметический оператор вещественного сдвига $o_{1,4}$, геометрический оператор вещественного сдвига $o_{1,5}$, *BGA*-оператор мутации и степенной оператор мутации $o_{1,6}$ (см. 2.3).

Аналогично в качестве вариантов оператора скрещивания o_2 используем следующие кроссоверы с одной особью-потомком: плоский кроссовер $o_{2,1}$; эвристический кроссовер $o_{2,2}$; смешанный кроссовер $o_{2,3}$; расширенный линейный кроссовер $o_{2,4}$; дискретный кроссовер $o_{2,5}$ (см. 2.4).

Положим, наконец, что выбору подлежат следующие варианты оператора отбора o_3 на основе близости приспособленностей индивидов: метод пропорционального отбора $o_{3,1}$; метод турнирного отбора $o_{3,2}$; метод рангового отбора $o_{3,3}$; метод отбора на основе элитизма $o_{3,4}$ (см. 2.5).

Таким образом, в данном случае $|o_1| = 6$, $|o_2| = 5$, $|o_3| = 4$ и мощность множества допустимых стратегий $|D_O|$ равна 120.

В качестве первой целевой метафункции $\mu_1(\phi, O)$ применим процент неулучшенных агентов (поскольку задача (7.10) представляет собой задачу минимизации), полученных с помощью рассматриваемой комбинации указан-

ных операторов. Вычислительная сложность различных вариантов операторов может весьма существенно отличаться. Пусть поэтому вторая метафункция $\mu_2(\phi, O)$ представляет собой число арифметических операций, реализующих рассматриваемую комбинацию этих вариантов.

Решение задачи (7.10) может быть выполнено методом адаптации или самоадаптации на множестве стратегий D_O.

Метод *адаптации*. В простейшем случае адаптационную стратегию можно построить с помощью метода рулетки по следующей схеме.

1) Поставим в соответствие каждой из $|D_O|$ допустимых комбинаций O_i рассматриваемых вариантов операторов некоторую начальную вероятность их применения ξ_i, так что имеет место равенство

$$\sum_{i \in [1:|D_O|]} \xi_i = 1.$$

2) По правилу рулетки в соответствии с вероятностью ξ_i выбираем комбинацию O_i.

3) В основном цикле генетического алгоритма определяем значения величин $\mu_1(\phi, O_i)$, $\mu_2(\phi, O_i)$, а также их скалярной свертки $\mu(\phi, O_i)$.

4) Изменяем распределение вероятностей ξ_i так, чтобы вероятность ξ_i была обратно пропорциональна значению целевой метафункции $\mu(\phi, O_i)$, и продолжаем итерационный процесс путем перехода к шагу 2.

5) Вычисления заканчиваем при выполнении условий окончания итераций.

Очевидным вариантом такого подхода к решению метазадачи динамической структурной оптимизации является вариант, в котором по рассмотренной схеме задают вероятности применения не наборов O_i, а непосредственно каждого из допустимых операторов $o_{i,j}$.

Метод *самоадаптации*. Введем в рассмотрение $((|X|+1)\times 1)$-вектор

$$\bar{X} = (X, n_o) = (x_1, x_2, ..., x_{|X|}, n_o),$$

где $n_o \in [1:|D_O|]$ – номер набора операторов из множества D_O. Экзон мультихромосомы $\bar{H}$ в этом случае представляет собой вектор варьируемых параметров базовой задачи, а интрон – номер набора n_o. Указанные части мультихромосомы $\bar{H}$ (экзон и интрон) эволюционируют в соответствии с функциями приспособленности $\varphi(X)$, $\mu(\phi, B)$ соответственно.

Генетические операторы, применяемые к экзону и интрону, различны. В первом случае речь идет о непрерывных генетических операторах, а во втором случае – о дискретных операторах.

Вопросы для самопроверки

1. Приведите постановку статической задачи параметрической метаоптимизации. Представьте общие постановки задач неадаптивной и адаптивной динамической метаоптимизации.

2. Приведите постановки задач структурной и структурно-параметрической метаоптимизации.

3. Перечислите основные классификационные признаки методов метаоптимизации по Эйбену и Смиту – Фогарти.

4. Изложите суть следующих методов параметрической метаоптимизации: однократная настройка параметров, перманентная настройка параметров, адаптивное управление параметрами, самоадаптивное управление параметрами.

5. Назовите возможные объекты, уровни и основания метаоптимизации.

6. Дайте общие схемы статической параметрической метаоптимизации, динамической неадаптивной и динамической адаптивной метаоптимизации.

7. Изложите схему перманентной настройки параметров.

8. Раскройте суть гомогенных и гетерогенных коэволюционных методов адаптивного управления параметрами.

9. Приведите схемы методов самоадаптивного управления параметрами на основе расширения вектора варьируемых параметров исходной задачи, а также на основе коэволюционного подхода.

10. Определите схемы статической и динамической структурной метаоптимизации.

Глава 8

ПОПУЛЯЦИОННЫЕ АЛГОРИТМЫ МНОГОЦЕЛЕВОЙ ОПТИМИЗАЦИИ

В русскоязычной литературе задачу поиска экстремумов одновременно нескольких функций обычно называют задачей многокритериальной оптимизации. Точнее будет использовать термин *многоцелевая оптимизация*.

Классические алгоритмы решения задачи многоцелевой оптимизации (МЦО-задачи) основаны на сведении ее к совокупности задач глобальной одноцелевой оптимизации. Относительно новый и быстро развивающийся класс алгоритмов многоцелевой оптимизации образуют *алгоритмы Парето-аппроксимации*, предполагающие предварительное построение некоторой конечномерной аппроксимации множества Парето, а тем самым и фронта Парето.

Обычно алгоритмы Парето-аппроксимации строят на основе эволюционных и, чаще всего, генетических алгоритмов. Такие алгоритмы Парето-аппроксимации называют эволюционными. Принципиальными в этом случае являются правила формирования фитнес-функции, обеспечивающей в конечном счете перемещение индивидов популяции в направлении множества Парето. Эволюция же этих индивидов может протекать по законам не только эволюционных, но и других популяционных алгоритмов. Поэтому в качестве общего названия рассматриваемых в данной главе алгоритмов используем термин *популяционные алгоритмы Парето-аппроксимации.*

В первом параграфе приведена постановка МЦО-задачи, даны определения множества и фронта Парето, классификация алгоритмов решения задачи. Для полноты картины сначала рассматриваем наиболее известные непопуляционные алгоритмы Парето-аппроксимации (второй параграф). Обзор популяционных алгоритмов Парето-аппроксимации представлен в третьем параграфе. При этом внимание сконцентрировано преимущественно на правилах формирования фитнес-функций, используемых в этих алгоритмах. Самостоятельную проблему в задаче Парето-аппроксимации представляет разработка критериев оценки качества получаемой аппроксимации. Основные из этих критериев рассматриваем в четвертом параграфе. Пятый параграф посвящен методам обеспечения качества Парето-аппроксимации. Последний (шестой) параграф представляет примеры решения задачи Парето-аппроксимации с помощью генетического алгоритма, а также с помощью алгоритмов роя частиц, колонии муравьев и колонии пчел.

8.1. ЗАДАЧА МНОГОЦЕЛЕВОЙ ОПТИМИЗАЦИИ (МЦО-ЗАДАЧА) И АЛГОРИТМЫ ЕЕ РЕШЕНИЯ

В данном разделе приводим постановку МЦО-задачи, даем определение множества и фронта Парето, формулируем основные свойства этих объектов, представляем классификацию алгоритмов решения МЦО-задачи.

8.1.1. Постановка задачи

Пусть множеством допустимых значений вектора варьируемых параметров X является ограниченное и замкнутое множество $D_X = \{X \mid G(X) \geq 0\} \subset \{X\} = R^{|X|}$. Положим, что *целевая вектор-функция* $F(X) = \left(f_1(X), f_2(X), ..., f_{|F|}(X)\right)$ со значениями в *целевом пространстве* $\{F\} = R^{|F|}$ определена в области D_X. ЛПР стремится минимизировать в этой области каждую из *частных целевых функций* $f_1(X), f_2(X), ..., f_{|F|}(X)$, что *условно* записываем в виде

$$\min_{X \in D_X} F(X) = F(X^*) = F^*, \tag{8.1}$$

где векторы X^*, F^* – искомое решение задачи многоцелевой оптимизации.

Полагаем, что частные целевые функции нормализованы по формуле

$$f_k(X) = \frac{f_k(X) - f_k^*}{f_k^{**} - f_k^*}, k \in [1:|F|],$$

где $f_k^* = \min_{X \in D_X} f_k(X)$, $f_k^{**} = \max_{X \in D_X} f_k(X)$.

При более широком взгляде на задачу (8.1) ее следует рассматривать как *задачу многоцелевого принятия решений*. В зависимости от мощности множества D_X можно выделить два класса таких задач. Если мощность множества D_X велика (например, как в рассматриваемом случае, когда это множество является континуальным), то задачу (8.1) называют *задачей многоцелевой оптимизации* (*multi-criteria* или *multi-objective optimization*). Если число альтернатив во множестве D_X невелико, то эта задача называется *задачей многоцелевого анализа* (*multi-criteria* или *multi-objective analysis*).

Целевая вектор-функция $F(X)$ выполняет отображение множества D_X в некоторое множество D_F целевого пространства, которое называется *множеством достижимости* (рис. 8.1). Введем на множествах D_F, D_X *отношения доминирования* и *строгого доминирования*.

Определение 8.1. Вектор $F_1 = F(X_1) \in D_F$ доминирует вектор $F_2 = F(X_2) \in D_F$, что записываем в виде $F_1 \succeq F_2$, если среди равенств и неравенств $f_k(X_1) \leq f_k(X_2)$, $k \in [1:|F|]$ имеется хотя бы одно строгое.

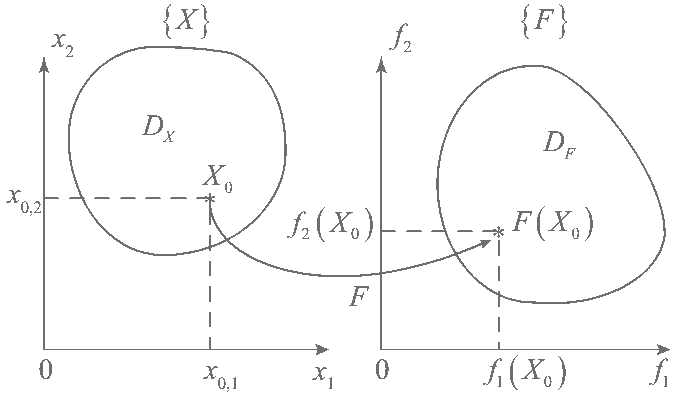


Рис. 8.1. К определению множества достижимости $D_F : |X| = |F| = 2$

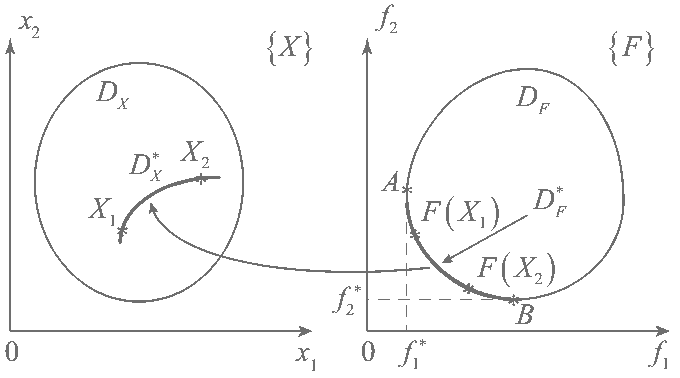


Рис. 8.2. К определению фронта Парето D_F^* и множества Парето D_X^* : $|X|=|F|=2$

Определение 8.2. Вектор X_1 доминирует вектор X_2, то есть $X_1 \overline{\rhd} X_2$, если $F(X_1) \overline{\succ} F(X_2)$.

Выделим из множества D_F подмножество точек D_F^* – *фронт Парето* МЦО-задачи (8.1), которые не доминируются другими точками множества D_F и среди которых нет доминирующих друг друга. Множество $D_X^* \in D_X$, соответствующее множеству D_F^*, называют *множеством Парето* указанной МЦО-задачи. Таким образом, если $X \in D_X^*$, то $F(X) \in D_F^*$ (рис. 8.2).

Наряду с отношениями доминирования определим аналогичные *отношения строгого доминирования*.

Определение 8.3. Вектор $F_1 = F(X_1) \in D_F$ строго доминирует вектор $F_2 = F(X_2) \in D_F$ ($F_1 \succ F_2$), если для всех $k \in [1:|F|]$ справедливы неравенства $f_k(X_1) < f_k(X_2)$.

Определение 8.4. Вектор X_1 строго доминирует вектор X_2 ($X_1 \rhd X_2$), если $F(X_1) \succ F(X_2)$.

Пример 8.1. Рассмотрим задачу двухмерной ($|X|=2$) двухцелевой ($|F|=2$) оптимизации (8.1) в ситуации, когда

$$D_X = \{X \mid x_1, x_2 \in [-1;1]\}, \tag{8.2}$$

$$f_1(X) = x_1^2 + x_2^2,\ f_2(X) = (x_1-1)^2 + (x_2-1)^2. \tag{8.3}$$

Можно показать, что множество D_F при этом имеет вид, представленный на рис. 8.3, *б*, а множество Парето D_X^* есть не что иное, как отрезок прямой, соединяющий точки (0;0), (1;1), – рис. 8.3, *а*. Отметим, что первая из указанных точек представляет собой точку минимума целевой функции $f_1(X)$, а вторая – аналогичную точку для целевой функции $f_2(X)$ •

Роль множества Парето при решении МЦО-задачи определяет следующая теорема.

Теорема 8.1. Если для некоторых весовых множителей $\lambda_k > 0$, $k \in [1:|F|]$, имеет место равенство

$$\min_{X \in D_X} \sum_{k=1}^{|F|} \lambda_k f_k(X) = \sum_{k=1}^{F|} \lambda_k f_k(X^*), \tag{8.4}$$

где $X^* \in D_X$, то вектор X^* оптимален по Парето, то есть $X^* \in D_X^*$.

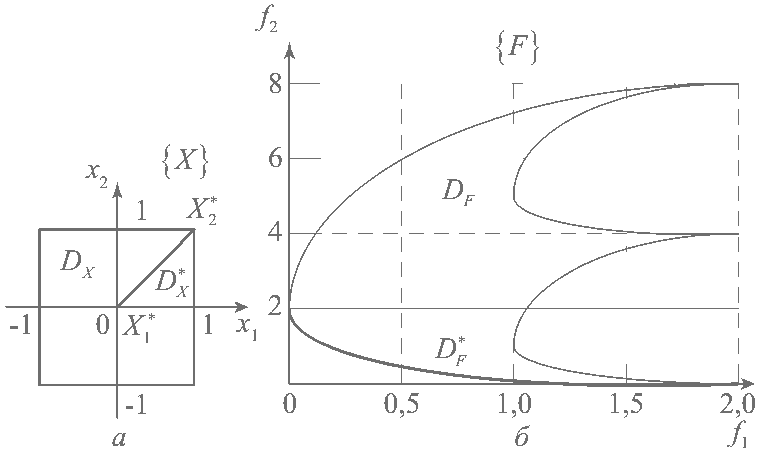


Рис. 8.3. Множество Парето (*а*) и фронт Парето (*б*) для задачи (8.2), (8.3)

Доказательство. Пусть вектор X^* не оптимален по Парето. Тогда существует такой вектор $X \in D_X$, что

$$f_k X) \le f_k(X^*),\ k \in [1:|F|],$$

причем хотя бы одно из этих неравенств является строгим. Умножая каждое из последних неравенств на $\lambda_k > 0$ и складывая, получим

$$\sum_{k=1}^{|F|} \lambda_k f_k(X) < \sum_{k=1}^{F} \lambda_k f_k(X^*),$$

что противоречит условию теоремы •

Теорема 8.1 показывает, что выбор определенной точки из множества Парето эквивалентен указанию весов каждой из частных целевых функций. На этом факте основано большое число приближенных алгоритмов решения МЦО-задачи.

Заметим, что теорема 8.1 задает лишь *необходимое условие* оптимальности по Парето вектора $X^* \in D_X$. Другими словами, из того факта, что точка X^* принадлежит множеству Парето, не следует, что эта точка обязательно удовлетворяет условию (8.4).

Постановка МЦО-задачи (8.1) фиксирует лишь множество допустимых значений D_X и вектор целевых функций $F(X) = (f_1(X), f_2(X), ..., f_{|F|}(X))$. Как правило, этой информации недостаточно для однозначного решения указанной задачи. Данная информация позволяет лишь выделить соответствующее множество Парето. Поэтому часто говорят, что решением МЦО-задачи в постановке (8.1) является множество Парето этой задачи. Для однозначного решения задачи нужна дополнительная информация.

Множество Парето и фронт Парето занимают в теории многоцелевой оптимизации исключительное место. Это обусловлено тем обстоятельством, что, согласно известному *принципу Эджворта – Парето (F. Edgeworth, V. Pareto),* при «разумном» поведении ЛПР выбор решения следует производить на множестве Парето.

8.1.2. Классификация алгоритмов решения МЦО-задачи

Противоречивый характер целей ЛПР обуславливает, как правило, противоречивость частных целевых функций, то есть своих минимальных значений эти функции достигают в различных точках множества D_X. В результате ЛПР, принимая некоторый вектор $\tilde{X}^* \in D_X$ в качестве решения МЦО-задачи, всегда должно идти на компромисс – допускать ухудшение значений одних частных функций ради улучшения значений других функций. Вследствие этого обстоятельства деятельность ЛПР при решении МЦО-задачи является плохо формализуемой и требует привлечения его предыдущего опыта и интуиции.

Алгоритмы решения МЦО-задачи чрезвычайно разнообразны (что является, в конечном счете, следствием отмеченной плохой формализуемости этой задачи). Существует несколько способов классификации указанных алгоритмов. Используем в качестве основы классификацию, предложенную А. В. Лотовым, и выделим следующие классы алгоритмов решения МЦО-задачи: алгоритмы зондирования; априорные алгоритмы; апостериорные алгоритмы; адаптивные алгоритмы; алгоритмы Парето-аппроксимации.

Алгоритмы каждого из указанных классов имеют свои достоинства и ни один из них не свободен от недостатков, наличие которых не позволяет признать алгоритмы этого класса универсальными. Эти же классы алгоритмов в значительной мере переплетаются друг с другом так, что не всегда МЦО-алгоритмы удается однозначно отнести к тому или иному классу. Так, в настоящее время развивается класс адаптивных эволюционных алгоритмов, которые основаны на многократном построении некоторых фрагментов множества Парето. Примером алгоритмов этого класса служит алгоритм *MOEA/D* (*Multi-Objective Evolutionary Algorithm based on Decomposition*), предложенный Чжаном (*Q. Zhang*) и Ли (*H. Li*) в 2007 г.

Широко известна более грубая классификация алгоритмов решения МЦО-задачи, в соответствии с которой выделяют алгоритмы априорного предпочтения, прогрессивного предпочтения, апостериорного предпочтения.

Первый класс алгоритмов в этой классификации совпадает с классом априорных алгоритмов.

Второй класс предполагает в той или иной форме взаимодействие ЛПР с программной системой многоцелевой оптимизации (МЦО-системой), так что его предпочтения формируются в процессе решения задачи. Данный класс включает в себя апостериорные и адаптивные алгоритмы предыдущей классификации.

Третий класс не предполагает установления каких-либо предпочтений перед или в процессе решения МЦО-задачи. МЦО-система в этом случае только поставляет ЛПР набор кандидатов, из которых ЛПР неформальными методами выбирает окончательное решение. Данный класс объединяет алгоритмы зондирования и алгоритмы Парето-аппроксимации.

Общей идеей решения МЦО-задачи является сужение множества D_X вплоть до одной или немногих точек. Построение множества Парето или его некоторой аппроксимации можно интерпретировать как часть этого пути.

Алгоритмы зондирования

Алгоритмы зондирования не предполагают формализации в той или иной форме дополнительной информации о предпочтениях ЛПР. Схема решения МЦО-задачи в этом случае имеет следующий вид.

1) Покрываем множества D_X некоторой сеткой $\Omega_X = (X_j,\ j \in [1:|\Omega_X|])$, число узлов $|\Omega_X|$ в которой определяет требуемая точность решения.

2) Вычисляем в узлах сетки Ω_X значения всех частных целевых функций $F_j = (f_1(X_j), f_2(X_j), ..., f_{|F|}(X_j)) = F(X_j)$, $j \in [1:|\Omega_X|]$.

3) Предъявляем полученные результаты ЛПР, которое выполняет их неформальный анализ и выбирает одно из решений либо требует от МЦО-системы уточнить его.

Основной проблемой рассмотренной схемы является возможность получения нерепрезентативных аппроксимаций множеств D_X^*, D_F^*, то есть возможная сильная неравномерность распределения точек X_j^*, F_j^* во множествах D_X^*, D_F^* соответственно. Кроме того, при высокой вычислительной сложности целевых и ограничивающих функций и/или высокой требуемой точности аппроксимации множеств D_X^*, D_F^* метод зондирования может требовать неприемлемо больших затрат вычислительных ресурсов.

При высокой размерности векторов X, F и сложной топологии множеств D_X, D_F, D_X^*, D_F^* важной проблемой в алгоритме зондирования является проблема предоставления ЛПР полученных результатов в форме, удобной для анализа и принятия решения. Поскольку решение МЦО-задачи в данном случае, собственно, не производится, алгоритмы зондирования можно отнести к алгоритмам многоцелевого анализа.

Вариантом алгоритмов зондирования можно считать алгоритмы, в которых узлы сетки Ω_X генерируют случайным образом.

Априорные алгоритмы

Априорные алгоритмы предполагают формализацию дополнительной информации о предпочтениях ЛПР до начала решения задачи, то есть априори. Чаще всего эту информацию представляют в форме, позволяющей свести многоцелевую задачу к одноцелевой задаче оптимизации некоторой скалярной функции. Наиболее известным алгоритмом этого класса является *алгоритм скалярной свертки*. В данном случае указанную скалярную функцию образует, например взвешенная сумма частных целевых функций (аддитивная скалярная свертка)

$$F(X) = \sum_{k=1}^{|F|} \lambda_k f_k(X),\ \sum_{k=1}^{|F|} \lambda_k = 1,\ \lambda_k > 0, \tag{8.5}$$

где λ_k – априори назначаемые ЛПР весовые коэффициенты частных целевых функций.

В соответствии с теоремой 8.1, отыскание точки X^*, принадлежащей множеству Парето, эквивалентно отысканию соответствующих весовых коэффициентов λ_k (для выпуклого множества Парето). Если вместо свертки (8.5) использовать известную *свертку Джоффриона* на основе *лексикографического упорядочения*, то этот подход можно реализовать и для невыпуклого множества Парето.

Апостериорные алгоритмы

Апостериорные алгоритмы предполагают внесение ЛПР в МЦО-систему дополнительной информации о своих предпочтениях в ходе решения задачи, то есть апостериори. При этом обычно дополнительную информацию, как и в априорных алгоритмах, в конечном счете формализуют в виде некоторой скалярной целевой функции.

В качестве примера апостериорного алгоритма рассмотрим *алгоритм последовательных уступок*, схема которого имеет следующий вид.

1) ЛПР ранжирует частные целевые функции по важности. Для простоты записи примем, что первой по важности является функция $f_1(X)$, второй – функция $f_2(X)$ и т. д.

2) МЦО-система отыскивает на множестве D_X минимум функции $f_1(X)$. ЛПР выполняет анализ полученного значения целевой функции f_1^* и назначает максимально допустимую уступку по этой функции δ_1.

3) МЦО-система отыскивает на множестве D_X минимум функции $f_2(X)$ с ограничением $f_1(X) \le f_1^* + \delta_1$. ЛПР производит анализ полученной величины f_2^* и назначает максимально допустимую уступку по этой функции δ_2.

4) МЦО-система отыскивает на множестве D_X минимум функции $f_3(X)$ с ограничениями $f_1(X) \le f_1^* + \delta_1$, $f_2(X) \le f_2^* + \delta_2$. ЛПР производит анализ полученной величины f_3^* и назначает максимально допустимую уступку по этой функции δ_3.

5) И так далее для целевых функций $f_4(X),\ldots, f_{|F|-1}(X)$.

6) МЦО-система отыскивает на множестве D_X минимум $f_{|F|}^*$ функции $f_{|F|}(X)$ с ограничениями $f_i(X) \le f_i^* + \delta_i, i \in [1:|F|-1]$. Совокупность величины $f_{|F|}^*$ и соответствующих величин f_i^*, $i \in [1:|F|-1]$, объявляем решением МЦО-задачи.

Адаптивные алгоритмы

Адаптивные алгоритмы включают в себя некоторую совокупность итераций, каждая из которых содержит фазу анализа, выполняемого ЛПР, и фазу расчетов, выполняемых МЦО-системой. По характеру информации, получаемой этой системой от ЛПР на фазе анализа, можно выделить три класса адаптивных алгоритмов: алгоритмы, в которых ЛПР непосредственно назначает весовые коэффициенты частных целевых функций; алгоритмы, в которых ЛПР накладывает ограничения на значения этих функций; алгоритмы, в которых ЛПР выполняет только сравнение (например, в терминах «лучше», «хуже», «одинаково») альтернатив, предлагаемых МЦО-системой.

Схему адаптивного алгоритма последнего класса представляет следующая последовательность шагов.

1) ЛПР оценивает предлагаемые МЦО-системой решения в терминах «лучше», «хуже», «одинаково» и предоставляет их системе.

2) МЦО-система организует на этой основе с помощью какого-либо алгоритма скалярной оптимизации один шаг поиска минимума неявно заданной функции предпочтений ЛПР.

3) ЛПР по схеме шага 1 оценивает полученные результаты, и т. д.

Как отмечалось выше, априорные, апостериорные и адаптивные алгоритмы в конечном счете сводят решение МЦО-задачи к одноцелевой задаче глобальной оптимизации, которая может быть решена всеми рассмотренными популяционными алгоритмами.

Алгоритмы Парето-аппроксимации

Выделяют четыре класса задач Парето-аппроксимации, соответствующих четырем типам фронта Парето – выпуклые, вогнутые, выпукло-вогнутые и разрывные задачи (рис. 8.4).

Выделяем непопуляционные и *популяционные алгоритмы Парето-аппроксимации*. В отличие от непопуляционных алгоритмов, в популяционных алгоритмах реализуется одновременный поиск не одной, а большого числа недоминируемых точек, близких к множеству или к фронту Парето.

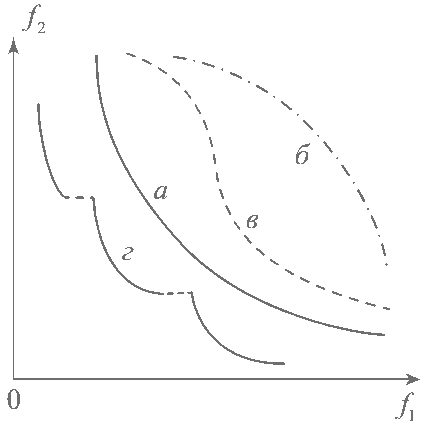


Рис. 8.4. Типы фронтов Парето ($|F| = 2$): *а* – выпуклый; *б* – вогнутый; *в* – выпукло-вогнутый фронт; *г* – разрывный

Среди непопуляционных выделяем сеточные алгоритмы и алгоритмы на основе скалярной свертки частных целевых функций.

Обозначим Θ^F, Θ^X *архивные множества*, содержащие недоминируемые точки F_j^{Θ} и соответствующие им точки X_j^{Θ}; $j \in [1:|\Theta|]$. Большинство популяционных алгоритмов Парето-аппроксимации использует итерационное уточнение множеств точек в архивах Θ^F, Θ^X. Если при этом на итерации t появляется новая точка F_i, доминирующая некоторые точки из архива Θ^F, то все доминируемые точки, а также соответствующие им точки из архива Θ^X удаляем. При удовлетворении некоторого критерия останова текущее содержимое архивов Θ^F, Θ^X полагаем искомой аппроксимацией фронта Парето D_F^* и множества Парето D_X^* соответственно.

В популяционных алгоритмах Парето-аппроксимации новые точки для архивов Θ^F, Θ^X «поставляет» популяция S агентов s_i, текущие координаты которых в пространстве поиска равны X_i, а в целевом пространстве – $F_i = F(X_i)$; $i \in [1:|S|]$.

В популяционных алгоритмах одноцелевой оптимизации миграция агентов в пространстве поиска подчинена задаче минимизации (для определенности) значений фитнес-функции. Основной проблемой построения популяционных алгоритмов Парето-аппроксимации является построение фитнес-функции, обеспечивающей перемещение агентов популяции s_i в направлении множества Парето D_X^*, а соответствующих точек F_i – в направлении фронта Парето D_F^*.

В силу, как правило, меньшей размерности целевого пространства $\{F\}$ по сравнению с размерностью пространства параметров $\{X\}$ ответ на вопрос о направлении и шаге перемещения агентов обычно отыскивают в терминах целевого пространства, а не пространства параметров. Важно также, что относительно фронта Парето, в отличие от множества Парето, имеется некоторая

априорная информация. Например, с учетом того, что частные целевые функции нормированы и поэтому $f_k(X) \in [0;1]$ для всех $X \in D_X$ и всех $k \in [1:|F|]$ имеет место утверждение 8.1.

Утверждение 8.1. Любой луч, проведенный из начала системы координат $0f_1\ f_2 \ldots f_{|F|}$ в неотрицательном направлении осей $0f_1,\ 0f_2, \ldots, 0f_{|F|}$, пересекает фронт Парето не более чем в одной точке.

Доказательство проведем от противного. Допустим, что утверждение неверно, и указанный луч пересекает фронт Парето в точках F_1^*, F_2^*. В этом случае по крайней мере для одного $k \in [1:|F|]$ должно быть справедливо неравенство $f_{1,k}^* < f_{2,k}^*$ или неравенство $f_{1,k}^* > f_{2,k}^*$, что противоречит условию принадлежности точек F_1^*, F_2^* фронту Парето •

По способу определения приспособленности агентов разделяем популяционные алгоритмы Парето-аппроксимации на алгоритмы на основе лексикографической селекции, алгоритмы чередующихся целевых функций и алгоритмы, непосредственно использующие концепцию доминирования по Парето.

Иногда выделяют еще класс популяционных алгоритмов, основанных на использовании скалярной свертки частных целевых функций, когда каждый из агентов популяции определяется своим набором весовых коэффициентов, значения которых могут изменяться в процессе эволюции популяции. Алгоритмы данного класса не получили заметного распространения и далее не рассматриваются.

Отметим также класс алгоритмов Парето-аппроксимации на основе покрытия множества D_X шарами, радиусы которых определяются константами Липшица целевых функций. Алгоритмы этого класса представляют собой обобщение ставшего классическим алгоритма покрытий, предложенного Ю. Г. Евтушенко в 60-е гг. прошлого века. Важно, что алгоритмы данного класса порождают внутреннюю и внешнюю аппроксимации множества Парето.

8.2. НЕПОПУЛЯЦИОННЫЕ АЛГОРИТМЫ ПАРЕТО-АППРОКСИМАЦИИ

Рассмотрим сеточные алгоритмы, а также алгоритмы, основанные на агрегировании (сворачивании) частных целевых функций в одну параметризованную скалярную целевую функцию, выступающую в роли фитнес-функции.

8.2.1. Сеточные алгоритмы

Схема простейшего сеточного алгоритма близка к схеме рассмотренных выше алгоритмов зондирования и имеет следующий вид.

1) Покрываем область D_X достаточно плотной сеткой Ω_X с узлами X_j, $j \in [1:|\Omega_X|]$.

2) В каждом из этих узлов вычисляем значение вектор-функции $F_j = F(X_j)$.

3) Среди векторов F_j выбираем недоминируемые векторы $\{F_{j_l}^*\}$ и находим соответствующее им множество точек $\{X_{j_l}^*\}$, где $F_{j_l}^* = F(X_{j_l}^*)$, $j_l \in [1:|\Omega_X|]$.

Множества $\left\{X_{j_l}^*\right\}$, $\left\{F_{j_l}^*\right\}$ представляют собой искомые дискретные аппроксимации множества и фронта Парето МЦО-задачи (8.1).

Важным достоинством сеточного алгоритма является простота учета ограничений на значения частных целевых функций. Эти ограничения отсутствуют в приведенной выше постановке МЦО-задачи, но часто возникают в прикладных задачах. Отметим также в качестве достоинств сеточного алгоритма простоту и возможную высокую эффективность его распараллеливания.

Известным вариантом сеточного алгоритма является *алгоритм исследования пространства параметров* (И. М. Соболь, Р. Б. Статников). Особенностью алгоритма является использование специальных сеток, построенных на основе так называемых $ЛП_\tau$ -последовательностей, которые могут значительно повысить эффективность алгоритма.

8.2.2. Алгоритмы на основе свертки целевых функций

Выделим следующие алгоритмы данного класса: алгоритм взвешенных целевых функций, алгоритм идеальной точки, а также новый перспективный адаптивный алгоритм взвешенных сумм.

Алгоритм взвешенных целевых функций

Теоретической основой алгоритма взвешенных целевых функций (*Sum of Weighted Objectives*, *SWO*) является теорема 8.1, которая показывает, что выбор определенной точки из множества Парето эквивалентен указанию весов для каждой из частных целевых функций. Еще раз подчеркнем, что теорема задает лишь *необходимое условие* оптимальности по Парето вектора $X^* \in D_X$. То есть из того факта, что точка X^* принадлежит множеству Парето не следует, что эта точка обязательно удовлетворяет условию (8.4) – случай невыпуклого множества D_F^* (рис. 8.5).

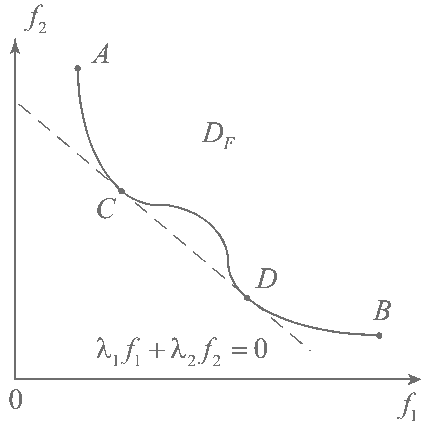


Рис. 8.5. Невыпуклый фронт Парето: $|F| = 2$; дуга AB – фронт Парето D_F^*

Фитнес-функция в алгоритме взвешенных целевых функций имеет вид

$$\varphi(X) = \varphi(\Lambda, F) = \sum_{k=1}^{|F|} \lambda_k f_k(X) = (\Lambda, F),$$

где вектор $\Lambda = (\lambda_1, \lambda_2, ..., \lambda_{|F|})$ представляет собой $|F|$-мерный вектор весовых коэффициентов. Основным недостатком алгоритма является невозможность с его помощью локализовать точки фронта Парето, принадлежащие невыпуклой части фронта (дуга CD на рис. 8.5).

Схема алгоритма взвешенных сумм включает в себя следующие основные шаги.

1) Множество $D_\Lambda = \{\Lambda \mid 0 \le \lambda_k \le 1, k \in [1:|F|]\}$ допустимых значений вектора весовых множителей Λ покрываем некоторой сеткой Ω_Λ с узлами $\Lambda_j = (\lambda_{1,j}, \lambda_{2,j}, ..., \lambda_{|F|,j})$, $j \in [1:|\Omega_\Lambda|]$.

2) При каждом $\Lambda_j \in \Omega_\Lambda$ решаем задачу глобальной условной оптимизации (8.4) – получаем точку $X_j^* \in D_X^*$, а также соответствующую точку F_j^*.

Совокупности полученных точек $\{X_j^*\}, \{F_j^*\}$ образуют искомые аппроксимации множества и фронта Парето соответственно.

Алгоритм идеальной точки

Идеальной называют точку $F^{**} = (f_1^*, f_2^*, ..., f_{|F|}^*)$ в целевом пространстве такую, что

$$f_k^* = f_k(X_k^*) = \min_{X \in D_X} f_k(X),\ k \in [1:|F|].$$

В качестве фитнес-функции в алгоритме идеальной точки используем функцию

$$\varphi(X) = \rho(F^{**}, F(X)),$$

где $\rho(\cdot,\cdot)$ – некоторая метрика пространства $\{F\}$, например чебышевская метрика

$$\rho(F^{**}, F) = \max_{k \in [1:|F|]} \left(\lambda_k abs(f_k^* - f_k)\right),\ \lambda_k > 0. \qquad (8.6)$$

Алгоритм идеальной точки сводит задачу Парето-аппроксимации к многократному решению задачи глобальной условной оптимизации

$$\min_{X \in D_X} \rho(F^{**}, F(X)) = \rho(F^{**}, F(X^*))$$

при различных допустимых значениях компонент вектора весов $\Lambda = (\lambda_1, \lambda_2, ..., \lambda_{|F|})$ по схеме, близкой к схеме алгоритма взвешенных целевых функций.

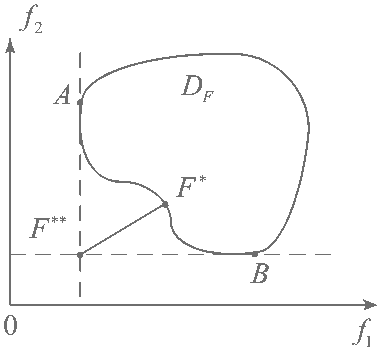


Рис. 8.6. К алгоритму идеальной точки: $|F| = 2$; дуга AB – фронт Парето

Алгоритм идеальной точки иллюстрирует рис. 8.6, показывающий, что в отличие от алгоритма взвешенных целевых функций данный алгоритм позволяет отыскивать точки, лежащие на невыпуклой части фронта Парето. Заметим, что свертка на основе идеальной точки (8.6) близка к известной свертке Гермейера.

Адаптивный алгоритм взвешенных сумм

Адаптивный алгоритм взвешенных сумм (*Adaptive Weighted Sum algorithm*, *AWS*) предложили Рю (*J-H. Ryu*), Ким (*S. Kim*) и Ван (*H. Wan*) в 2009 г. Целью разработки алгоритма было преодоление указанного выше ограничения алгоритма взвешенных целевых функций, а именно невозможности отыскать точки, принадлежащие невыпуклым частям фронта Парето.

Алгоритм разработан для двухцелевой задачи ($|F| = 2$) и включает в себя три следующие основные процедуры: определение центральной точки; формирование метамоделей частных целевых функций; решение полученных оптимизационных задач. Рассмотрим суть указанных процедур.

Определение центральной точки. На этапе инициализации центральную точку X_C^0 выбираем случайным образом в области D_X. На этом же этапе должны быть определены следующие свободные параметры алгоритма: $\delta \geq \delta_{\min}$ – начальный *радиус области доверия* (*trust region radius*), где $\delta_{\min}$ – минимальная величина радиуса; $\rho \in (0;1)$ – коэффициент сужения указанной области.

На итерации $(t+1)$ центральную точку X'_C отыскиваем среди точек текущей Парето-аппроксимации Θ^X, построенной на предыдущей итерации t.

Отсортируем элементы архивных множеств Θ^X, Θ^F по возрастанию первой частной целевой функции $f_1(X)$ и представим в виде линейных списков с прежними наименованиями Θ^X, Θ^F. Определим расстояние d_j архивной точки F_j^Θ до ближайшей к ней в списке Θ^F точки формулой

$$d_j = \| F_{j-1}^\Theta - F_j^\Theta \|_E + \| F_j^\Theta - F_{j+1}^\Theta \|_E, \; j \in [2 : |\Theta| - 1]. \tag{8.7}$$

Алгоритм использует следующее правило определения центральной точки X'_C.

1) Если $|\Theta| > 2$, то полагаем $X'_C = X_{j^*}^\Theta$, где

$$j^* = \arg\left(\max_{j \in [2:\, |\Theta|-1]} d_j \mid X_j^\Theta \notin X_T \right).$$

Здесь X_T – множество точек, использованных в качестве центральных на всех предыдущих итерациях $[0:\, t]$. Иными словами, за центральную точку принимаем точку, во-первых, наиболее удаленную от других точек множества Θ^X в смысле расстояния (8.7) и, во-вторых, не использованную на предшествующих итерациях.

2) Если $|\Theta| = 2$, то с равной вероятностью полагаем $X'_C = X_1^\Theta$ или $X'_C = X_2^\Theta$.

3) Если $|\Theta| = 1$, то принимаем $X'_C = X_1^\Theta$.

Формирование метамоделей. Метамодели представляют собой квадратичные аппроксимации $m'_1(X)$, $m'_2(X)$ функций $f_1(X)$, $f_2(X)$ в окрестности точки X'_C:

$$m'_1(X) = f_1(X'_C) + \nabla^T f_1(X'_C)(X - X'_C) + \frac{1}{2}(X - X'_C)^T H_1(X'_C)(X - X'_C);$$

$$m'_2(X) = f_2(X'_C) + \nabla^T f_2 X'_C)(X - X'_C) + \frac{1}{2}(X - X'_C)^T H_2(X'_C)(X - X'_C).$$

Здесь $H_1(X'_C), H_2(X'_C)$ – матрицы Гессе функций $f_1(X)$, $f_2(X)$ в точке X'_C.

Если $|\Theta| = 1$ или $|\Theta| = 2$, дополнительно строим метамодель

$$m'_p(X) = \lambda_1^p \, m'_1(X) + \lambda_2^p \, m'_2(X),$$

а если $|\Theta| > 2$ – то еще и метамодель

$$m'_q(X) = \lambda_1^q \, m'_1(X) + \lambda_2^q \, m'_2(X).$$

В случае $|\Theta| = 1$ весовые множители λ_1^p, λ_2^p определяет равенство $\Lambda^p = (0{,}5,\; 0{,}5)$. При $|\Theta| = 2$ эти множители вычисляем по правилу

$$\Lambda^p = c^p \left((-f_2(X_2^\Theta) + f_2(X_1^\Theta)),\; (f_1(X_2^\Theta) - f_1(X_1^\Theta)) \right).$$

Если $|\Theta| > 2$, то полагаем

$$\Lambda^p = (\lambda_1^p, \lambda_2^p) = c^p \left((-f_2(X'_C) + f_2(X^{\Theta}_{j^*-1})),\ (f_1(X'_C) - f_1(X^{\Theta}_{j^*-1})) \right),$$

$$\Lambda^q = (\lambda_1^q, \lambda_2^q) = c^q \left((-f_2(X^{\Theta}_{j^*+1}) + f_2(X'_C)), (f_1(X^{\Theta}_{j^*+1}) - f_1(X'_C)) \right).$$

Константы c^p, c^q в приведенных формулах выбираем таким образом, чтобы обеспечить выполнение условий нормировки $\lambda_1^p + \lambda_2^p = \lambda_1^q + \lambda_2^q = 1$.

При построении метамоделей $m'_1(X)$, $m'_2(X)$ речь может идти не о градиенте и матрице Гессе функций $f_1(X)$, $f_2(X)$, а об их оценках, полученных, например, численными методами (путем соответствующих конечно-разностных аппроксимаций указанных функций).

Решение оптимизационных задач. Данная процедура предполагает решение задач оптимизации

$$\min_{X \in D'_C} m'_1(X) = m'_1(\hat{X}'_1),\ \ \min_{X \in D'_C} m'_2(X) = m'_2(\hat{X}'_2), \tag{8.8}$$

где текущую *область доверия* D'_C определяет формула

$$D'_C = \{X \mid X \in D_X, \left\| X - X'_C \right\| \le \delta\}.$$

Если $|\Theta| > 2$, то решения $\hat{X}'_1, \hat{X}'_2$ задач (8.8) позволяют отыскать приближенно оптимальные по Парето точки $\hat{X}'_p, \hat{X}'_q$, принадлежащие области доверия D'_C, путем решения оптимизационных задач

$$\min_{X \in D'_C} m'_p(X) = m'_p(\hat{X}'_p),\ \ \min_{X \in D'_C} m'_q(X) = m'_q(\hat{X}'_q). \tag{8.9}$$

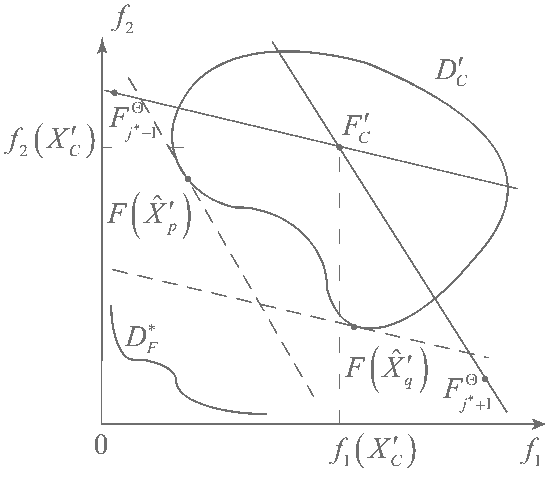


Рис. 8.7. К схеме адаптивного метода взвешенных сумм: результаты решения задач (8.9)

Данный этап алгоритма иллюстрирует рис. 8.7, на котором принято, что $F'_C = F(X'_C)$, $F^{\Theta}_{j^*-1} = F(X^{\Theta}_{j^*-1})$, $F^{\Theta}_{j^*+1} = F(X^{\Theta}_{j^*+1})$.

Задачи (8.8), (8.9) представляют собой задачи оптимизации квадратичных функций, для решения которых известны высокоэффективные методы, алгоритмы и соответствующее программное обеспечение.

В процессе итераций текущий радиус области доверия уменьшают по правилу $\delta = \rho\delta$ до достижения минимально допустимой величины $\delta_{\min}$. Новое состояние архивного множества Θ^X получаем путем добавления в него точек $\hat{X}'_1$, $\hat{X}'_2$, $\hat{X}'_p$, $\hat{X}'_q$ и исключения из полученного набора доминируемых решений. Аналогично, множество Θ^F образуем путем добавления в него точек $F(\hat{X}'_1)$, $F(\hat{X}'_2)$, $F(\hat{X}'_p)$, $F(\hat{X}'_q)$.

8.3. ПОПУЛЯЦИОННЫЕ АЛГОРИТМЫ ПАРЕТО-АППРОКСИМАЦИИ

В первом пункте этого параграфа представляем алгоритмы на основе лексикографической селекции. Второй пункт посвящен алгоритмам чередующихся целевых функций. Здесь же рассматриваем алгоритм динамических соседей и алгоритм «хищник-жертва». Два последних пункта содержат обзор алгоритмов, в той или иной форме непосредственно использующих концепцию доминирования по Парето: алгоритмы на основе ранжирования агентов популяции и алгоритмы указанного класса, не использующие ранжирование агентов.

8.3.1. Лексикографическая турнирная селекция

Алгоритм на основе лексикографической турнирной селекции (*lexicographic tournament selection*) является представителем класса алгоритмов, который Люк (*S. Luke*) в своей известной книге, посвященной метаэвристическим алгоритмам, называет «наивными». Алгоритм исходит из того, что частные целевые функции $f_k(X), k \in [1:|F|]$ упорядочены по важности так, что самой важной является функция $f_1(X)$, следующей по важности – функция $f_2(X)$ и так далее. Рассмотрим правило сравнения приспособленностей агентов, используемое данным алгоритмом.

Пусть $s_i, s_j, i, j \in [1:|S|], i \neq j$ – два сравниваемых агента в текущих положениях X_i, X_j. Лучшего из этих агентов определяем путем последовательного сравнения пар величин $\left(f_k(X_i), f_k(X_j)\right)$ до тех пор, пока не будет установлено, что, например, $f_{k_l}(X_i) < f_{k_l}(X_j)$; $k, k_l \in [1:|F|]$. В таком случае полагаем, что агент s_i имеет лучшую приспособленность по сравнению с агентом s_j.

Лучшего агента $s_{i_b} \in S$ популяции в ее текущем положении $X_i, i \in [1:|S|]$ определяем в результате следующей последовательности шагов.

1) Выбираем случайного агента s_{i_1} и полагаем его лучшим агентом, то есть полагаем $X_{i_b} = X_{i_1}$. Выполняем присваивание $j = 2$.

2) Выбираем другого случайного агента s_{i_j}.

3) По указанному выше правилу определяем лучшего из агентов s_{i_b}, s_{i_j} и объявляем его новым агентом s_{i_b}.

4) Если $j < m$ (размер турнира m не исчерпан), то полагаем $j = j + 1$ и возвращаемся к шагу 2.

Известен ряд модификаций рассмотренного алгоритма. Так, для выбора лучшего агента можно использовать случайно выбранную целевую функцию. Можно использовать процедуру голосования, когда лучшим считается агент, которому соответствуют наименьшие значения наибольшего числа целевых функций. Наконец, можно использовать многоуровневую турнирную селекцию (в случае $|F| > 2$), когда основной турнир проводим для одной целевой функции, однако агентов для этого турнира отбираем с использованием турнира по другой целевой функции, агентов для этого турнира отбираем с использованием турнира по третьей функции и т. д.

8.3.2. Алгоритмы чередующихся целевых функций

К классу алгоритмов чередующихся целевых функций относим алгоритм переключающихся функций и алгоритм динамического соседства. В плане логики использования целевых функций к этим алгоритмам близок также алгоритм «хищник-жертва».

Алгоритм переключающихся функций

Как и в «наивных» алгоритмах, в алгоритме переключающихся целевых функций (*altering objective functions*) оценку приспособленности агентов популяции производим на основе соответствующих значений различных целевых функций. Наиболее известным алгоритмом данного класса является генетический алгоритм *VEGA* (*Vector Evaluated Genetic Algorithm*), предложенный Шаффером (*D. Shaffer*) в 1985 г.

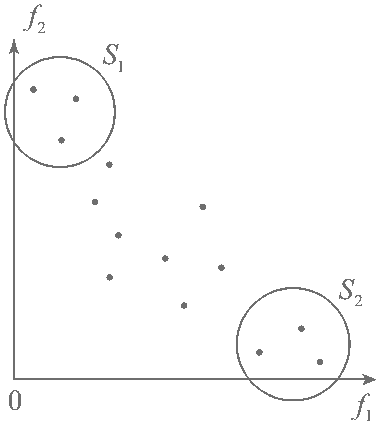


Рис. 8.8. К схеме отбора агентов в алгоритме *VEGA*

Пусть имеет место неравенство $|S| > |F|$ и указанные величины кратны. Приспособленность агентов в алгоритме *VEGA* определяем по следующей схеме.

1. Случайным образом разбиваем популяцию S на $|F|$ субпопуляций $S_1, S_2, ..., S_{|F|}$, каждая из которых включает в себя $|S|/|F|$ агентов.

2. Полагаем, что фитнес-функция всех агентов субпопуляции S_j совпадает с целевой функцией $f_j(X)$; $j \in [1:|F|]$.

Далее алгоритм *VEGA* в каждой субпопуляции $S_1, S_2, ..., S_{|F|}$ реализует пропорциональную селекцию на основе соответствующей фитнес-функции, чтобы затем по общим правилам генетического алгоритма применить к отобранным агентам операторы скрещивания и мутации.

Рассмотренную схему отбора иллюстрирует рис. 8.8, на котором представлены проекции точек F_i, $i \in [1:|S|]$ на плоскость $0 f_1 f_2$ целевого пространства $\{F\}$. Показаны множества агентов S_1, S_2, отобранных на основе целевых функций $f_1(X)$, $f_2(X)$ соответственно (имеется в виду задача многоцелевой минимизации).

Алгоритм динамического соседства

Алгоритм динамического соседства (*dynamic neighborhood*) предложили Хью (*X. Hu*) и Эберхарт (*R. Eberhart*) в 2002 г. Рассмотрим агента $s_i \in S, i \in [1:|S|]$. Схему выбора с помощью данного алгоритма лучшего для агента s_i решения можно представить в следующем виде.

1) Фиксируем все целевые функции, кроме функции $f_1(X)$ – полагаем эту функцию *динамической*.

2) Вычисляем в пространстве целевых функций $f_2, f_3, ..., f_{|F|}$ расстояния (обычно евклидовы) от точки F_i до всех архивных точек F_j^{Θ}, $j \in [1:|\Theta|]$.

3) Выбираем на этой основе m ближайших архивных соседей точки F_i, где m – свободный параметр алгоритма.

4) Из числа выделенных точек выбираем точку $F_{j_b}^{\Theta}$, лучшую с точки зрения целевой функции $f_1(X)$.

5) На следующей итерации принимаем в качестве динамической функцию $f_2(X)$, а все остальные функции фиксируем. И так далее.

Авторы алгоритма отмечают, что остается открытым вопрос об оптимальной величине числа m.

Алгоритм «хищник-жертва»

Алгоритм «хищник-жертва» (*predator-prey*) предложил Лоуманс (*M. Lautanns*) с соавторами в 1998 г. Рассмотрим оригинальный алгоритм Лоуманса и его основные модификации последних лет. Полагаем, что областью допустимых значений вектора варьируемых параметров является гиперпараллелепипед *П*.

Каждая из жертв представляет собой один вектор решений, а каждый хищник – одну целевую функцию. Популяция эволюционирует на тороидальной сетке (рис. 8.9), в узлах которой случайным образом инициализируем жертв и хищников.

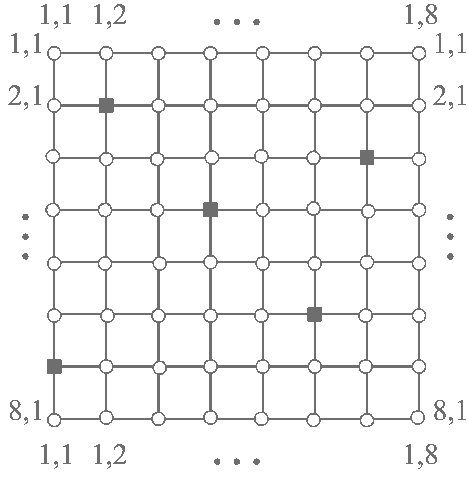


Рис. 8.9. Пример размещения хищников (квадратики) и жертв (кружки) на тороидальной сетке

Для каждого хищника рассматриваем все жертвы в его окрестности и удаляем жертву с худшим значением соответствующей целевой функции. Затем в той же окрестности выбираем и модифицируем случайную жертву. Измененное решение помещаем на место удаленной жертвы. Хищника случайным образом перемещаем в один из соседних с ним узлов сетки. Процесс итерационно повторяем до выполнения условий окончания.

Приведем схему алгоритма в более формализованном виде.

1) В гиперпараллелепипеде *П* случайным образом инициализируем популяцию жертв, то есть решений $X_i, i \in [1:|S|]$.

2) Ставим в соответствие жертвам случайные вершины ненаправленного связного графа в виде тороидальной решетки.

3) Случайным образом помещаем хищников в вершины указанного графа.

4) Ставим в соответствие каждому из хищников одну из целевых функций $f_k(X)$, $k \in [1:|F|]$ таким образом, чтобы каждая из функций была представлена по меньшей мере одним хищником.

5) Вычисляем значения соответствующей целевой функции для всех жертв в окрестности каждого хищника и выбираем худшие жертвы.

6) Полагаем, что выбранные жертвы поглощены хищниками и удаляем их из популяции.

7) Взамен каждого из удаленных агентов создаем новую жертву путем изменения случайно выбранной жертвы в окрестности удаленного агента.

8) Случайным образом перемещаем хищников из их текущих положений в свободные соседние вершины графа.

9) Если условие окончания итераций не выполнено, возвращаемся к шагу 5.

Экспериментально показано, что с ростом числа поколений популяция жертв стремится к фронту Парето, то есть представляет собой искомую Парето-аппроксимацию. Недостатком рассмотренного канонического алгоритма «хищник-жертва» является возможная потеря лучших (ближайших к множеству Парето) решений.

Деб (*K. Deb*) в 2001 г. предложил улучшения алгоритма, суть которых заключается в следующем:

– хищникам ставим в соответствие не частные целевые функции, а векторы их весов;

– вместо удаленной жертвы новую жертву создаем путем модификации не случайного, но лучшего агента в окрестности удаленной жертвы;

– хищника перемещаем не случайным образом, а в направлении позиции лучшей жертвы из его окрестности.

Исследования Деба показывают, что его модифицированный алгоритм обеспечивает по сравнению с каноническим более высокую скорость сходимости, однако, как и канонический алгоритм, он может терять найденные лучшие решения.

Ли (*X. Li*) выполнил модификацию канонического алгоритма, в которой жертв и хищников помещают не во все узлы сетки. Жертвы и хищники выполняют случайные перемещения в направлении соседних пустых узлов сетки, причем хищники движутся быстрее жертв. При каждом перемещении жертвы создаем ее потомка. Вычислительные эксперименты показали недостаточно высокую скорость сходимости такого алгоритма.

Деб (*K. Deb*) и Рао (*U. Rao*) в 2005 г. предложили глубокую модификацию канонического алгоритма «хищник-жертва», основу которой составляют механизм сохранения лучших решений, механизм рекомбинации решений и механизм получения равномерной Парето-аппроксимации.

Механизм сохранения лучших решений (элитизм) предполагает сравнения приспособленности худших жертв с приспособленностью созданных агентов (потомков). Меру приспособленности агентов строим на основе Парето-доминирования. Если данный потомок слабо доминирует всех жертв популяции (лучше их по крайней мере по одной из частных целевых функций), то заменяем этим потомком худшую жертву. В противном случае потомка объявляем неподходящим, худшую жертву оставляем в популяции, а хищника для обеспечения разнообразия популяции случайным образом перемещаем в направлении худшей жертвы.

Механизм рекомбинации решений использует в качестве основы операторы скрещивания и мутации агентов (так что, в целом, алгоритм можно считать вариантом генетического алгоритма). В окрестности худшей жертвы создаем двух потомков путем скрещивания лучшего и второго лучшего решений. Затем случайным образом выбираем одного из этих потомков и выполняем его мутацию. Если полученный таким образом потомок оказывается хуже худшей

жертвы, то по рассмотренной схеме создаем нового потомка и опять сравниваем его приспособленность с приспособленностью худшей жертвы. Процесс повторяем не более некоторого фиксированного числа раз.

Механизм обеспечения равномерности покрытия. Полагаем, что каждая жертва имеет *область влияния* (*influencing region*) в виде гиперкуба в целевом пространстве, в центре которого находится данный агент. Потомка полагаем неподходящим, если он создан в области влияния любой из существующих жертв. Заметим, что аналогичный механизм использует известный алгоритм ε -доминирования (*K. Deb, M. Mohan, S. Mishra*). Размеры области влияния назначаем исходя из требуемой плотности покрытия множества и/или фронта Парето.

Для того чтобы предотвратить ситуацию, когда хищники полностью уничтожают популяцию жертв, используем следующий прием. Число перемещений t^p, которые выполняют хищники перед тем, как жертвы могут сделать свои перемещения, объявляем динамической переменной, определяемой формулой

$$t^p = \left\lfloor \frac{\left|S^o\right| - \left|S_*^o\right|}{\left|S^p\right|} \right\rfloor,$$

где $\left|S^o\right|$, $\left|S_*^o\right|$ – текущее и желаемое числа жертв соответственно, $\left|S^p\right|$ – текущее число хищников. В результате при уменьшении числа жертв уменьшается величина t^p, то есть хищники начинают двигаться медленнее, шансы жертв на выживание повышаются, и популяция жертв расширяется. При этом хищники становятся более быстрыми, и популяция жертв начинает снова сокращаться. И так далее.

Авторы рассматриваемой модификации алгоритма выполнили широкое экспериментальное исследование его эффективности на ряде трудных задач тестового набора многоцелевых задач оптимизации. Показано, что алгоритм обеспечивает высокие скорость сходимости и равномерность покрытия фронта Парето.

Важным с практической точки зрения является класс задач Парето-аппроксимации, в которых требуется построить аппроксимацию не всего фронта Парето, но лишь той части его, которая ближе всего к заданной ЛПР «предпочтительной» точке пространства целей. Известна модификация алгоритма «хищник-жертва», предназначенная для решения таких задач.

8.3.3. Алгоритмы на основе ранжирования агентов

Основным понятием алгоритмов данного класса является понятие *ранга агента популяции*. Известно несколько правил вычисления рангов, основные из которых представлены в этом пункте.

Алгоритм недоминируемой сортировки

Алгоритм недоминируемой сортировки (*Non-Dominated Sorting*, *NDS*) впервые был опубликован в работе Шриниваса (*N. Srinivas*) и Деба (*K. Deb*) в 1994 г. Метод лежит в основе широко известного генетического алгоритма Парето-аппроксимации *NSGA* (*Non-dominated Sorting Genetic Algorithm*).

Положим, что все частные целевые функции являются одинаково важными. Ранг агента s_i, $i \in [1:|S|]$ в его текущем положении X_i обозначаем r_i. Алгоритм *NDS* использует простейшее из правил вычисления рангов, имеющее следующий вид (рис. 8.10).

1) Выбираем среди всех агентов популяции недоминируемых, присваиваем им ранг, равный единице, и исключаем из дальнейшего рассмотрения.

2) Среди оставшихся агентов выбираем недоминируемых, присваиваем им ранг, равный двум, и исключаем из дальнейшего рассмотрения. И так далее до исчерпания популяции.

Ранг индивида используют для вычисления его приспособленности, например, по формуле вида

$$\varphi(X_i) = \frac{1}{1+r_i}, \; i \in [1:|S|].$$

Здесь имеется в виду, что фитнес-функция $\varphi(X)$ подлежит минимизации.

Предшественником метода недоминируемой сортировки можно считать *алгоритм MOGA* (*Multi-Objective Genetic Algorithm*), называемый еще алгоритмом *FFGA (Fonseca and Fleming's multi-objective Genetic Algorithm),* который предложили Фонсека (*C. M. Fonseca*) и Флеминг (*P. J. Fleming*) в 1993 г. В алгоритме предполагается, что ранг r_i агента s_i равен числу агентов в популяции, которые его доминируют, плюс единица, то есть

$$r_i = 1 + \left|\{s_j \mid X_j \trianglerighteq X_i, s_j \in S\}\right|, \; i \in [1:|S|],$$

где, напомним, запись вида $|B|$ означает мощность множества B.

Заметим, что алгоритм *MOGA* фиксирует лишь правило определения пригодностей агентов популяции, так что для отбора агентов в следующее поколение могут быть использованы различные процедуры селекции.

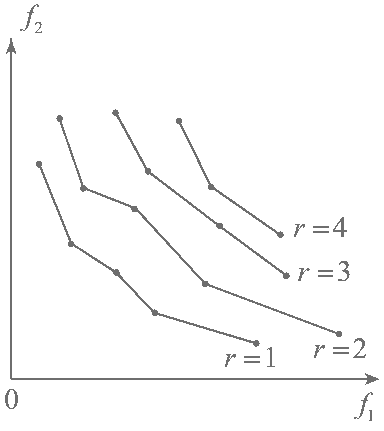


Рис. 8.10. К определению понятия ранга агента: $|F| = 2$

Заслуживает также упоминания *генетический алгоритм Парето-аппроксимации с нишеванием* (*Niched-Pareto Genetic Algorithm*, *NPGA*), предложенный Хорном (*J. Horn*) с соавторами в 1994 г. Принципиальное отличие данного алгоритма от двух предыдущих заключается в использовании механизма поддержания разнообразия популяции на основе нишевания (см. 2.6).

Важнейшим требованием к методам и алгоритмам Парето-аппроксимации является требование обеспечения равномерности покрытия множества и фронта Парето. Для оценки равномерности покрытия может быть использована величина, называемая *разреженностью покрытия* (*scarcity*), которая имеет смысл минимального расстояния между решениями, принадлежащими Парето-аппроксимации. Указанное расстояние может быть измерено с помощью различных метрик, например с помощью манхэттен-

ской метрики. Развитием алгоритма недоминируемой сортировки является алгоритм, в котором при формировании архивов Θ^X, Θ^F отвергают агентов, расстояние которых до других агентов в архиве не превышает заданной величины. Такой модифицированный алгоритм положен в основу генетического алгоритма Парето-аппроксимации *NSGA II* (*Non-dominated Sorting Genetic Algorithm II*).

Алгоритм Парето-силы

Иное правило ранжирования агентов популяции использует алгоритм Парето-силы (*Pareto strength*). Алгоритм исходит из того, что сила агента s_i равна числу особей, которых доминирует по Парето данный агент. Определенную указанным способом силу агентов можно использовать для вычисления их приспособленности. Однако такое определение пригодности обладает рядом недостатков. Поэтому в вычислительной практике используют альтернативное понятие *слабости (weakness) агента*, равной числу особей, его доминирующих. Точнее говоря, используют вариант слабости агента, названный Люком (*S. Luke*) его *хилостью* (*wimpiness*) w_i и равный суммарной силе всех особей, доминирующих данного агента. Фитнес-функция при этом имеет вид

$$\varphi_i = \frac{1}{1+w_i}, \; i \in [1:|S|].$$

Алгоритм Парето-силы реализует известный популяционный алгоритм Парето-аппроксимации *SPEA* (*Strength Pareto Evolutionary Algorithm*). Развитием алгоритма *SPEA* является алгоритм *SPEA 2*, который подобно алгоритму *NSGA II* использует средства обеспечения равномерности Парето-аппроксимации. Важной особенностью алгоритмов *SPEA* является возможность априорного задания числа итоговых точек Парето-аппроксимации.

Упрощенным вариантом алгоритма *SPEA 2* можно считать алгоритм *PAES* (*Pareto Achieved Evolution Strategy*), предложенный Ноулзом (*J. Knowles*) и Корном (*D. Corne*) в 1999 г. Ранг r_i агента s_i в алгоритме *PAES* представляет собой число архивных решений, доминируемых решением X_i. Схема алгоритма *PAES* для агента s_i, $i \in [1:|S|]$ имеет следующий вид.

1) По законам используемого популяционного алгоритма (в оригинале – генетического алгоритма) перемещаем агента s_i из его текущего положения X_i в новое положение X_i'.

2) Если одно из решений X_i и X_i' доминирует другое решение, то лучшее из них объявляем *кандидатом*. В противном случае кандидатом объявляем то из решений X_i, X_i', ранг которого выше.

3) Если решение-кандидат доминируется некоторым из архивных решений, то кандидата отвергаем. В противном случае кандидата записываем в архив, но только в том случае, если он находится в той части области поиска, в которой на текущий момент обнаружено мало решений. При записи кандидата в архив все доминируемые им решения из архива удаляем.

4) Если условие окончания итераций не выполнено, возвращаемся к шагу 1.

Другие алгоритмы на основе ранжирования

Алгоритм средневзвешенного ранжирования (*Weighted Average Ranking*, *WAR*) предложили Бентли (*P.J. Bentley*) и Вэйкфилд (*J.P. Wakefield*) в 1996 г. Пусть λ_k – относительная важность целевой функции $f_k; k \in [1:|F|]$. Алгоритм *WAR* предполагает вычисление текущего ранга агента $s_i, i \in [1:|S|]$ по следующей схеме.

1) Для каждой из функций f_k формируем список L_k, содержащий величины $f_k(X_i)$, упорядоченные по их возрастанию.

2) Ранг агента s_i полагаем равным величине

$$r_i = \sum_{k=1}^{|F|} \lambda_k n_k(X_i), \tag{8.10}$$

где $n_k(X_i)$ – порядковый номер агента s_i в списке L_k.

Алгоритм суммы взвешенных оценок (*Sum of Weighted Ratios*, *SWR*) можно считать развитием алгоритма *WAR*. При вычислении ранга агентов алгоритм использует свертку вида (8.10), в которой вместо относительной важности частных целевых функций λ_k входят их особым образом нормализованные значения. Нормализация основана на использовании наихудших и наилучших текущих значений целевых функций

$$f_{k_w}(X_i) = \max_{k \in [1:|F|]} f_k(X_i), \; f_{k_b}(X_i) = \min_{k \in [1:|F|]} f_k(X_i)$$

и определяется формулой

$$\overline{f}_k(X_i) = \frac{f_k(X_i) - f_{k_w}(X_i)}{f_{k_b}(X_i) - f_{k_w}(X_i)}, k \in [1:|F|].$$

Заметим, что поскольку числитель и знаменатель этой формулы неположительны, величина $\overline{f}_k$ имеет знак плюс.

Алгоритм суммы взвешенных глобальных оценок (*Sum of Weighted Global Ratios*, *SWGR*) близок к предыдущему алгоритму с тем отличием, что наихудшие и наилучшие значения целевых функций определяют на основе всей предыстории поиска, то есть на основе всех итераций $\tau \in [1:t]$:

$$f_{k_w}(X_i) = \max_{\tau \in [1:t]} \max_{k \in [1:|F|]} f_k(X_i(\tau)); \; f_{k_b}(X_i) = \min_{\tau \in [1:t]} \min_{k \in [1:|F|]} f_k(X_i(\tau)).$$

Алгоритм максимального взвешенного ранжирования (*Weighted Maximum Ranking*, *WMR*) исходит из предположения, что все целевые функции нормализованы и заданы их относительные важности λ_k, $k \in [1:|F|]$. Схему алгоритма для агента $s_i \in S$ определяет следующая последовательность шагов.

1) Аналогично алгоритму *WAR* формируем списки L_k и определяем величины $n_k(X_i)$; $k \in [1:|F|]$.

2) Вычисляем ранги агента s_i по всем $|F|$ целевым функциям

$$r_k(X_i) = \frac{\lambda_k}{n_k(X_i)}.$$

3) В качестве итогового ранга агента s_i в его текущем положении X_i принимаем величину

$$r_i = \max_{k \in [1:|F|]} r_k(X_i).$$

8.3.4. Алгоритмы, не использующие ранжирование агентов

Данный класс алгоритмов представляет сигма-алгоритм, а также алгоритмы композитных точек и гиперкубов.

Сигма-алгоритм

Сигма-алгоритм (*sigma algorithm*) предложили Мостагхим (*S. Mostaghim*) и Тейч (*J. Teich*) в 2003 г.

Для двухцелевой задачи ($|F| = 2$) сигма-параметром точки X_i или соответствующей ей точки $F_i = (f_{i,1}, f_{i,2})$ называется величина

$$\sigma_i = \sigma(F_i) = \frac{f_{i,1}^2 - f_{i,2}^2}{f_{i,1}^2 + f_{i,2}^2}, \; i \in [1:|S|].$$

Легко видеть, что точки F_i, имеющие одно и то же значение сигма-параметра, лежат на одной прямой, проходящей через начало системы координат $0 f_1 f_2$. Определение сигма-параметра иллюстрирует рис. 8.11, на котором одинаковые значения сигма-параметра имеют пары точек $(F_1, F_2), (F_3, F_4), (F_5, F_6)$.

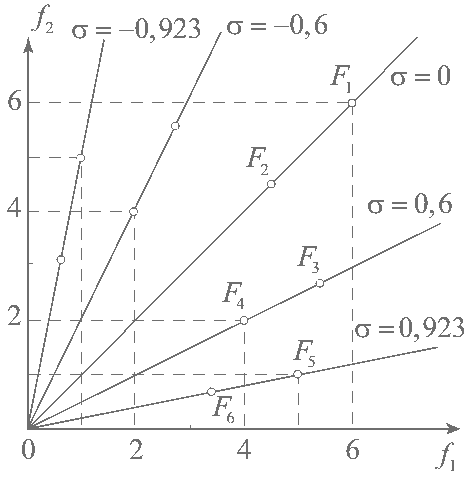


Рис. 8.11. К определению сигма-параметра, $|F| = 2$

В случае, когда размерность целевого пространства больше двух ($|F| > 2$), сигма-алгоритм использует $|F|$-мерный сигма-вектор параметров $\Sigma = (\sigma_1, \sigma_2, \ldots, \sigma_{|F|})$. Для агента s_i этот вектор определяет выражение

$$\Sigma_i = \Sigma(F_i) = \begin{pmatrix} f_{i,1}^2 - f_{i,2}^2 \\ f_{i,2}^2 - f_{i,3}^2 \\ \ldots \\ f_{i,|F|}^2 - f_{i,1}^2 \end{pmatrix} \cdot (f_{i,1}^2 + f_{i,2}^2 + \cdots + f_{i,F|}^2)^{-1}, \; i \in [1:|S|].$$

В сигма-алгоритме для точки X_i лучшую архивную точку X_j^{Θ} определяем по следующей схеме (полагаем, что $i \in [1:|S|]$, $j \in [1:|\Theta|]$).

1) Определяем сигма-вектор Σ_i точки $F(X_i)$.

2) Находим в архиве Θ^F точку F_j^{Θ}, у которой сигма-вектор Σ_j наиболее близок к вектору Σ_i:

$$\min_{l \in [1:|\Theta|]} \|\Sigma_l - \Sigma_i\|_E = \|\Sigma_j - \Sigma_i\|_E. \qquad (8.11)$$

3) В качестве лучшей для точки X_i принимаем точку X_j^{Θ} такую, что $F(X_j^{\Theta}) = F_j^{\Theta}$.

Как видим, в качестве метрики близости точек F_i, F_j^{Θ} в данном случае используется евклидова норма в сигма-пространстве $\{\Sigma\}$.

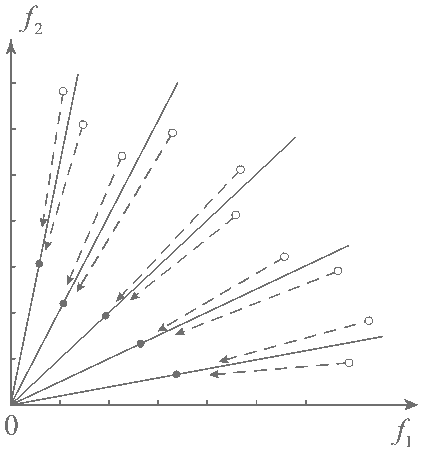


Рис. 8.12. Схема сигма-метода для двухцелевой задачи:
• – архивные точки F_j^{Θ}; ∘ – точки F_i

Утверждение 8.2. Предположим, что для каждой из архивных частиц X_i^{Θ} определен сигма-вектор Σ_i; $i \in [1:|\Theta|]$. Тогда отыскание для точки X_i лучшей архивной точки X_j^{Θ} сигма-алгоритмом требует $O(|F||\Theta|)$ арифметических операций.

Справедливость утверждения вытекает из того факта, что для отыскания точки, которая является лучшей для точки X_i, требуется $|\Theta|$ раз вычислить значение нормы вида (8.11) и для каждого такого вычисления необходимо $O(|F|)$ арифметических операций •

Для двухцелевой задачи изложенную схему сигма-алгоритма иллюстрирует рис. 8.12. Стрелками на рисунке для каждой из точек F_i показаны «притягивающие» архивные точки F_j^{Θ}.

Алгоритм композитных точек

Алгоритм композитных точек предложили Филдсенд (*J. E. Fieldsend*) и Сингх (*S. Singh*) в 2002 г. Алгоритм использует архив Θ^F недоминируемых решений в виде так называемого дерева доминирования (*dominated tree*), представляющего собой список *композитных точек* (*composite points*) $C_j = (c_{1,j}, c_{2,j}, \ldots, c_{|F|,j})$, $j \in [1:|\Theta|]$. Для двухцелевой задачи пример дерева доминирования приведен на рис. 8.13.

Координаты первой композитной точки C_1 определяем по следующему правилу.

1) В качестве первой координаты $c_{1,1}$ этой точки принимаем первую координату той архивной точки $F_{j_1}^{\Theta}$, у которой значение этой координаты наибольшее, то есть полагаем

$$c_{1,1} = \max_{j \in [1:|\Theta|]} f_{1,j}^{\Theta} = f_{1,j_1}^{\Theta}.$$

Исключаем точку $F_{j_1}^{\Theta}$ из дальнейшего рассмотрения.

2) Вторую координату $c_{1,2}$ точки C_1 полагаем равной второй координате той из оставшихся архивных точек F_j^{Θ}, у которой эта координата имеет наибольшее значение:

$$c_{1,2} = \max_{\substack{j \in [1:|\Theta|] \\ j \neq j_1}} f_{2,j}^{\Theta} = f_{2,j_2}^{\Theta}.$$

Точку $F_{j_2}^{\Theta}$ исключаем из дальнейшего рассмотрения.

3) Если $|F|>2$, то поступаем аналогичным образом для всех остальных координат, так что

$$c_{1,|F|}=\max_{\substack{j\in[1:|\Theta|]\\ j\notin\{j_1,j_2,\ldots,j_{|F|-1}\}}} f^{\Theta}_{|F|,j}=f^{\Theta}_{|F|,j_{|F|}}.$$

4) По аналогичной схеме вычисляем координаты второй композитной точки C_2 и так далее до исчерпания всех точек архива Θ^F.

Рассмотрим схему алгоритма быстрого построения дерева доминирования. Положим, что сформированы списки $L_1,L_2,\ldots,L_{|F|}$ из $|\Theta|$ элементов каждый, такие, что в списке L_k содержатся точки архива Θ^F, отсортированные в порядке убывания целевой функции $f_k(X)$; $k\in[1:|F|]$. При использовании известного алгоритма быстрой сортировки построение таких списков требует, в худшем случае, $O(|\Theta|^2)$ операций сравнения. Элементы списка L_k обозначим $L_{k,j}$, $k\in[1:|F|]$, $j\in[1:|\Theta|]$.

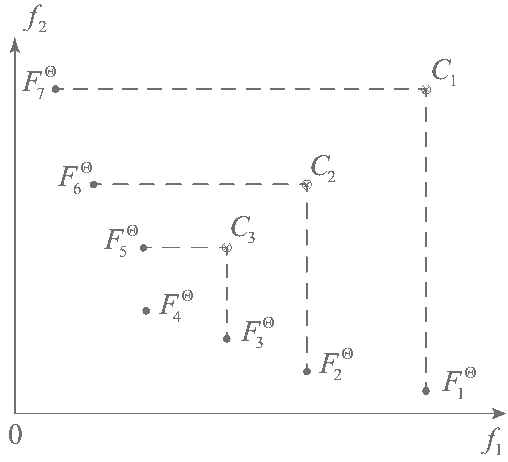


Рис. 8.13. Дерево доминирования для двухцелевой задачи: • – архивные точки; ⊗ – композитные точки

1) По очереди для $k=1,2,\ldots,|F|$ в качестве координаты $c_{1,k}$ точки C_1 принимаем элемент $L_{k,1}$ списка L_k и исключаем этот элемент из всех списков.

2) По очереди для $k=1,2,\ldots,|F|$ в качестве координаты $c_{2,k}$ точки C_2 принимаем элемент $L_{k,1}$ (то есть элемент $L_{k,2}$ исходного списка L_k) и исключаем этот элемент из всех списков.

3) И так далее до исчерпания всех списков L_k.

Легко видеть, что архиву Θ^F соответствует общее число композитных точек, равное $|C|=\left\lfloor\frac{|\Theta|}{|F|}\right\rfloor$. Архивные точки $F^{\Theta}_{j_1},F^{\Theta}_{j_2},\ldots,F^{\Theta}_{j_{|F|}}$, соответствующие композитной точке C_j, называем *вершинами* этой точки; $j\in[1:|C|]$.

После того как дерево доминирования построено, лучшую точку для точки X_i определяем по следующему правилу.

1) В списке композитных точек находим точку C_j такую, что имеет место хотя бы одно из неравенств

$$c_{j,k}\le f_{i,k}(X_i)<c_{j+1,k},\ \ j\in[1:|C|-1],\ \ k\in[1:|F|].$$

2) Из числа вершин $F^{\Theta}_{j_1},F^{\Theta}_{j_2},\ldots,F^{\Theta}_{j_{|F|}}$ композитной точки C_j находим точку $F^{\Theta}_{j_p}$, которая строго доминирует точку F_i, то есть такую, что $F^{\Theta}_{j_p}\succ F_i$.

3) В качестве лучшей для точки X_i принимаем точку $X^{\Theta}_{j_p}$, соответствующую вершине $F^{\Theta}_{j_p}$.

4) Если среди вершин $F^{\Theta}_{j_1},F^{\Theta}_{j_2},\ldots,F^{\Theta}_{j_{|F|}}$ имеется несколько вершин (более одной), строго доминирующих точку F_i, то из них случайным образом

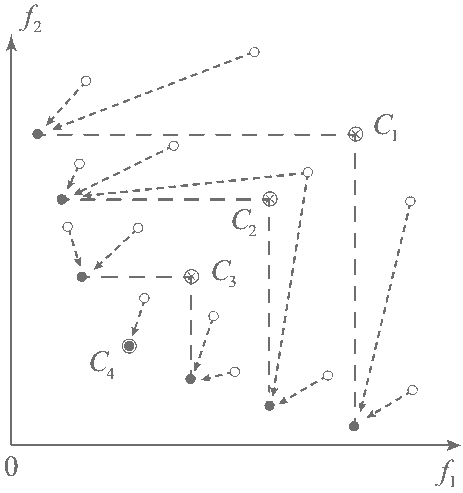


Рис. 8.14. К определению лучшей точки в методе композитных точек: • – архивные точки; ⊗ – композитные точки; ∘ – точки F_i

выбираем вершину $F_{j_p}^{\Theta}$ и в качестве лучшей для точки X_i принимаем соответствующую точку $X_{j_p}^{\Theta}$.

Рассмотренное правило определения лучшей точки иллюстрирует рис. 8.14, на котором стрелками показаны соответствующие лучшие точки.

Утверждение 8.3. Предположим, что дерево доминирования предварительно тем или иным образом построено. Тогда поиск в архиве Θ^X точки, которая является лучшей для точки X_i, требует $O(|C|\cdot|F|^2)$ сравнений.

Справедливость утверждения вытекает из того факта, что в изложенной схеме определения лучшей точки поиск композитной точки C_j требует $O(|C|\cdot|F|)$, а выбор точки $F_{j_p}^{\Theta}$ – не более $O(|F|)$ сравнений •

Гиперкубовый алгоритм

В основе гиперкубового алгоритма, который предложили Коэлло (*C. A. Coello*) и Лечунга (*M. S. Lechunga*) в 2002 г., лежит процедура покрытия области достижимости D_F некоторой совокупностью гиперкубов и процедура назначения каждому из этих гиперкубов коэффициента пригодности. Рассмотрим данные процедуры.

Напомним, что частные целевые функции $f_1(X), f_2(X),\ldots,f_{|F|}(X)$ полагаются нормированными, так что $f_k(X)\in[0;1]$, $k\in[1:|F|]$ для любого $X\in D_X$ и $D_F\subset H^0$, где $H^0=[0;1]\times[0;1]\times\ldots\times[0;1]$ – $|F|$-мерный единичный гиперкуб.

Искомое покрытие строим путем разделения гиперкуба H^0 на $n^{|F|}$ гиперкубов H_j^1 с длиной ребра, равной $1/n$ (рис. 8.15), где натуральное число n – свободный параметр алгоритма.

Пусть некоторый гиперкуб H_j^1, $j\in[1:n^{|F|}]$ включает в себя m_j точек архива Θ^F. Пригодность этого гиперкуба принимаем равной величине $\varphi_j=10/m_j$. Если выбранный гиперкуб вообще не содержит точек архива, то производим расширение области поиска, то есть вводим в рассмотрение гиперкуб, который включает в себя гиперкуб H_j^1.

Таким образом, пригодность каждого из построенных гиперкубов полагаем обратно пропорциональной числу архивных точек, которые находятся в нем, и гиперкуб, содержащий меньшее число этих точек, оказывается более пригодным (рис. 8.15). Такое определение пригодности направлено на обеспечение более равномерной Парето-аппроксимации.

Если по рассмотренному правилу множество достижимости D_F покрыто гиперкубами и для каждого из гиперкубов определены их пригодности, то отыскание в архиве Θ^F точки F_j^{Θ}, которая является лучшей для точки F_i, производим по следующей схеме.

1) Если в гиперкубе H_j^1, в котором находится точка F_i, имеются архивные точки, то случайным образом с равной вероятностью выбираем одну из этих точек $F_{j_i}^{\Theta}$.

2) В противном случае выполняем следующие действия.

2.1) Расширяем зону поиска – рассматриваем все возможные гиперкубы H_j^2 с длиной ребра равной $2/n$, которые включают в себя гиперкуб H_j^1 и содержат хотя бы одну архивную точку.

2.2) Определяем пригодность гиперкубов H_j^2 как сумму пригодностей входящих в них гиперкубов H_j^1.

2.3) Из числа всех найденных гиперкубов H_j^2 по правилу рулетки выбираем один из гиперкубов $H_{j_*}^2$.

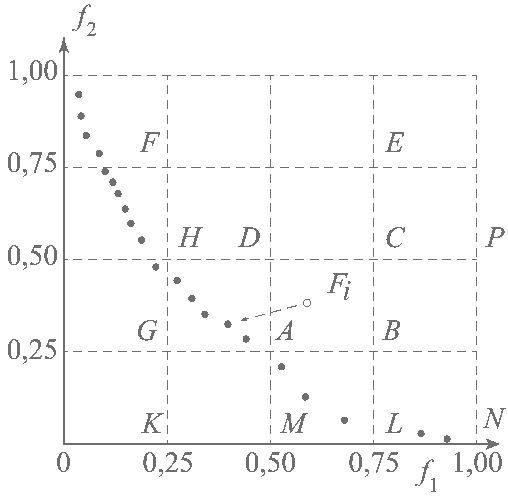


Рис. 8.15. Покрытие области D_F гиперкубами, $|F| = 2$, $n = 4$: • – архивные точки F_j^{Θ}

2.4) Случайным образом с равной вероятностью выбираем одну из точек $F_{j_i}^{\Theta}$ этого гиперкуба.

Приведенную схему выбора лучшей точки иллюстрирует рис. 8.15. Точка F_i на рис. 8.15 находится в гиперкубе $ABCD$. В этом гиперкубе нет ни одной архивной точки. Поэтому в рассмотрение вводим гиперкубы $BEFG$, $CHKL$, $DMNP$. Пригодности этих гиперкубов равны 2; 1,25; 2,5 соответственно. По правилу рулетки выбираем, например, гиперкуб $BEFG$. В этом гиперкубе с равной вероятностью выбираем одну из архивных точек, например точку, на которую на рисунке указывает стрелка.

8.4. КРИТЕРИИ ОЦЕНКИ КАЧЕСТВА ПАРЕТО-АППРОКСИМАЦИИ

Оценку эффективности алгоритмов Парето-аппроксимации естественно производить на основе качества полученной Парето-аппроксимации и суммарных вычислительных затрат. Таким образом, задача оценки эффективности алгоритмов Парето-аппроксимации сама является как минимум двухцелевой.

В содержательных терминах качество Парето-аппроксимации может быть оценено с помощью следующих характеристик:

– близость найденных решений к точному множеству Парето рассматриваемой МЦО-задачи;

– равномерность распределения решений в полученной Парето-аппроксимации;

– мощность найденного множества решений.

В практически значимых МЦО-задачах, в которых, как правило, используется более двух целевых функций, визуальный анализ качества соответствующей Парето-аппроксимации затруднен. Поэтому актуальной является задача разработки критериев качества Парето-аппроксимации, формализующих

указанные характеристики. Подобно тому, как это сделано в работе 2002 г. Цицлера (*E. Zitzler*) с соавторами, назовем критерии такого сорта *унарными* (*unary quality indicators*).

Качество Парето-аппроксимации можно оценивать также путем сравнения найденных недоминируемых решений с некоторыми другими, например, *эталонными решениями*. Критерии такого сравнения вслед за указанной работой называем *бинарными* (*binary quality indicators*).

В первом пункте данного параграфа представлены наиболее известные унарные критерии, а во втором – бинарные критерии. Ограничимся рассмотрением критериев, определенных в целевом пространстве, хотя известны их варианты, определенные в пространстве параметров. Если не оговорено противное, для простоты записи в обозначениях архивного множества Θ^F решений, недоминируемых в пространстве $\{F\}$, опускаем индекс F, и для элементов множества Θ используем обозначения вида F_j, $j \in [1:|\Theta|]$. Для критериев качества Парето-аппроксимации используем авторские обозначения Цицлера или близкие к ним.

8.4.1. Унарные критерии

Среднее расстояние до точного фронта (*Generational Distance*) $I_{GD}(\Theta)$ определяет формула

$$I_{GD}(\Theta) = \frac{\left(\sum_{j=1}^{|\Theta|} \left\| F_j, F_j^* \right\|_E^{|F|}\right)^{\frac{1}{|F|}}}{|\Theta|} \to \min,$$

где F_j^* – ближайшая к точке F_j точка множества D_F^*. Критерий характеризует близость найденных решений к точному фронту Парето D_F^*.

Для критерия $I_{GD}(\Theta)$ и других рассматриваемых ниже критериев можно, очевидно, предложить большое число вариантов, используя другие фигурирующие в них метрики.

Среднее рассеяние (*Spacing*) $I_S(\Theta)$ представляет собой меру равномерности распределения решений, полученных в результате Парето-аппроксимации. Критерий задает формула

$$I_S(\Theta) = \sqrt{\frac{1}{|\Theta|-1} \sum_{j=1}^{|\Theta|} abs\left(\bar{d} - d_j\right)} \to \min, \tag{8.12}$$

где $d_j = \min\limits_{k \in [1:|\Theta|],\, k \neq j} \left\| F_j, F_k \right\|_M$ – минимальное манхэттенское расстояние между архивным решением F_j и остальными архивными решениями; $\bar{d}$ – среднее всех этих величин.

Максимальное рассеяние (*Maximum Spread*) $I_{MS}(\Theta)$ определяет формула, аналогичная формуле (8.12):

$$I_{MS}(\Theta) = \sqrt{\sum_{j=1}^{|\Theta|} \max_{k \in [1:|\Theta|]} \left\| F_j, F_k \right\|_M} \to \min.$$

Отклонение от равномерного распределения (*Deviation from Uniform distribution*) $I_{DU}(\Theta)$, аналогично критериям $I_S(\Theta)$, $I_{MS}(\Theta)$, определяет равномерность распределения архивных недоминируемых решений Θ в целевом пространстве. Критерий задает формула

$$I_{DU}(\Theta) = \frac{1}{|\Theta|}\sum_{j=1}^{|\Theta|} abs\left(d_j - \bar{d}\right),$$

где d_j – евклидово расстояние решения F_j до ближайшего из решений множества Θ; $\bar{d}$ – среднее этих величин.

Мощность множества решений (*Overall Nondominated Vector Generation*) $I_{ONVG}(\Theta)$ есть не что иное, как число элементов множества Θ, то есть

$$I_{ONVG}(\Theta) = |\Theta| \to \max.$$

Объем объемлющего гиперкуба (*Hypercube enClosing indicator*) $I_{HC}(\Theta)$ определяют величины (рис. 8.16)

$$I_{HC_1}(\Theta) = \max_{P \in R^{|F|}}\left(P \overline{\succ} F_j, \forall j \in [1:|\Theta|]\right) = p,\ P = (p, p, ..., p),$$
$$I_{HC_2}(\Theta) = \min_{Q \in R^{|F|}}\left(Q \overline{\prec} F_j, \forall j \in [1:|\Theta|]\right) = q,\ Q = (q, q, ..., q).$$

Относительное число ошибочных решений (*Error Ratio*) $I_{ER}(\Theta)$ – относительное число решений архивного множества Θ, не принадлежащих точному фронту Парето D_F^* рассматриваемой МЦО-задачи:

$$I_{ER}(\Theta) = \frac{1}{|\Theta|}\sum_{j=1}^{|\Theta|} e_j \to \min,\ e_j = \begin{cases} 1, F_j \notin D_F^*, \\ 0, F_j \in D_F^*. \end{cases}$$

Можно предложить вариант критерия $I_{ER}(\Theta)$, представляющий собой число решений множества Θ, не принадлежащих точному фронту Парето с заданной точностью.

Максимальная ошибка аппроксимации (*Maximum Pareto front Error*) $I_{MPE}(\Theta)$ также для своего вычисления требует знания точного фронта Парето. Критерий определяет формула

$$I_{MPE}(\Theta) = \max_{j \in [1:|\Theta|]} \left\| F_j, F_j^* \right\|_M^{|F|} \to \min,$$

где, как и ранее, F_j^* – ближайшая к точке F_j точка множества D_F^*.

Нам понадобятся далее понятия ε-доминирования и ε_+-доминирования.

Определение 8.4. Вектор $F_1 \in D_F$ ε-доминирует вектор $F_2 \in D_F$, что записываем в виде $F_1 \overline{\succ}_\varepsilon F_2$, если для всех $j \in [1:|F|]$ имеет место неравенство

$$f_{1,j} \le \varepsilon\, f_{2,j},$$

где ε – заданная положительная константа.

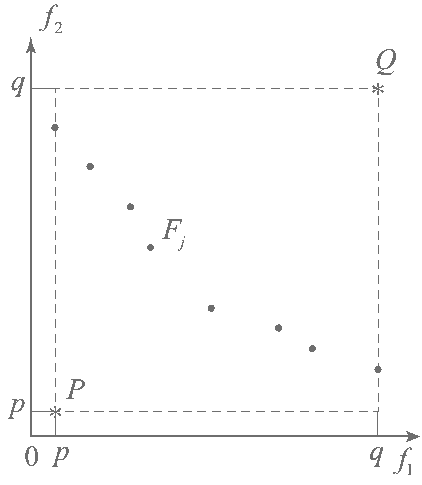


Рис. 8.16. К определению критерия $I_{HC}(\Theta)$, $|F| = 2$: точки – недоминируемые архивные решения Θ; $I_{HC_1}(\Theta) = p$; $I_{HC_2}(\Theta) = q$

Определение 8.5. Вектор $F_1 \in D_F$ ε_+-доминирует вектор $F_2 \in D_F$, то есть $F_1 \overset{-}{\succ}_{\varepsilon_+} F_2$, если для всех $j \in [1:|F|]$ имеет место неравенство

$$f_{1,j} \le \varepsilon + f_{2,j}\,,\ \varepsilon > 0.$$

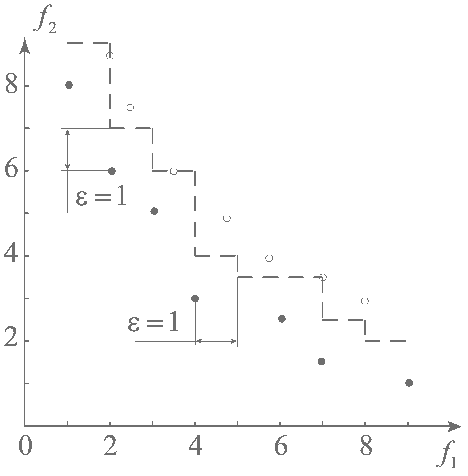


Рис. 8.17. К определению критерия $I_{\varepsilon_+}(\Theta_1, \Theta_2)$, $|F| = 2$: • – решения множества Θ_1; ∘ – решения множества Θ_2; $I_{\varepsilon_+}(\Theta_1, \Theta_2) = 1$

Определим бинарный критерий $I_\varepsilon(\Theta_1, \Theta_2)$ $\left(I_{\varepsilon_+}(\Theta_1, \Theta_2)\right)$, формализующий близость множеств недоминируемых решений Θ_1, Θ_2 как минимальную величину ε, при которой для любого вектора $F_2 \in \Theta_2$ существует вектор $F_1 \in \Theta_1$, который ε- (ε_+)-доминирует вектор F_2, то есть такой вектор, что $F_1 \overset{-}{\succ}_\varepsilon F_2$ $\left(F_1 \overset{-}{\succ}_{\varepsilon_+} F_2\right)$.

Определение критерия $I_{\varepsilon_+}(\Theta_1, \Theta_2)$ иллюстрирует рис. 8.17.

Минимальная ошибка аппроксимации (*Spacing Distribution in* ε *-area*)

$$I_{SD(\varepsilon)}(\Theta) = I_\varepsilon(D_F^*, \Theta) \to \min$$

представляет собой минимальную величину ε, для которой справедливо отношение $F_j \overset{-}{\succ}_\varepsilon F_k^*$ для любых $F_j \in \Theta$ и любых $F_k^* \in D_F^*$. Для определения значения критерия $I_{SD(\varepsilon)}(\Theta)$ необходимо знание точного фронта Парето D_F^*.

Аналогично определяется критерий $I_{SD(\varepsilon_+)}(\Theta)$.

Критерий *протяженности* (*Dimensions Extent*) $I_{DE}(\Theta)$ позволяет оценить суммарную протяженность Парето-аппроксимации в целевом пространстве по всем $|F|$ измерениям. Значения критерия вычисляют по формуле

$$I_{DE}(\Theta) = \sqrt{\sum_{j=1}^{|F|} \max_{k,l \in [1:|\Theta|]} \left| f_j(X_k^\Theta) - f_j(X_l^\Theta) \right|} \to \max.$$

8.4.2. Бинарные критерии

Положим, что множество $\Theta_*^F = \Theta_*$ недоминируемых решений в целевом пространстве D_F является «эталонным», а множество $\Theta^F = \Theta$ – аналогичным оцениваемым множеством.

ε *-критерий* $I_\varepsilon(\Theta_*, \Theta)$ строят на основе критерия $I_\varepsilon(\Theta_1, \Theta_2)$. Величина $I_\varepsilon(\Theta_*, \Theta)$ формализует близость множества Θ к множеству Θ_* как минимальную величину ε, при которой для любых векторов $F_j \in \Theta$, $F_k \in \Theta_*$ имеет место отношение $F_j \overset{-}{\succ}_\varepsilon F_k$.

ε_+ *-критерий* $I_{\varepsilon_+}(\Theta_*, \Theta)$ определяем аналогичным образом.

Критерий $I_C(\Theta_*, \Theta)$, называемый *покрытием* (*Coverage*), может быть использован в том случае, когда множество Θ_* является счетным. Величина $I_C(\Theta_*, \Theta)$ имеет смысл относительного числа решений, принадлежащих множеству Θ_*, которые доминируются некоторыми из решений множества Θ:

$$I_C(\Theta_*,\Theta)=\frac{\left|\left\{F_k\in\Theta_*\mid\exists F_j\in\Theta:F_j\succ F_k\right\}\right|}{\left|\Theta_*\right|}\to\max.$$

Здесь числитель представляет собой число указанных решений. Легко видеть, что $I_C(\Theta_*,\Theta)\in[0;1]$. Случай $I_C(\Theta_*,\Theta)=1$ соответствует ситуации, когда все решения из множества Θ_* слабо доминируются решениями из множества Θ, а случай $I_C(\Theta_*,\Theta)=0$ – ситуации, когда число первых решений равно нулю.

Следующий бинарный критерий строят на основе величины, которая называется *разностью покрытий* (*coverage difference*).

Определение 8.6. Пусть Θ_1,Θ_2 – множества недоминируемых векторов данной МЦО-задачи. Величина

$$\delta(\Theta_1,\Theta_2)=\varsigma(\Theta_1+\Theta_2)-\varsigma(\Theta_2)$$

называется разностью покрытий и представляет собой некоторую меру $\varsigma(\cdot)$ части множества достижимости, слабо доминируемой множеством Θ_1, но не слабо доминируемой множеством Θ_2.

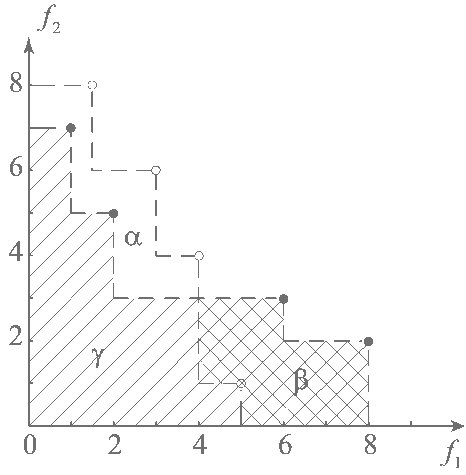


Рис. 8.18. К определению разности покрытий $\delta(\Theta_1,\Theta_2)$, $|F|=2$: • – решения множества Θ_1; ∘ – решения множества Θ_2; область α покрывает только множество Θ_2, область β – множество Θ_1, область γ – оба указанных множества

Определение иллюстрирует рис. 8.18, где $\delta(\Theta_1,\Theta_2)=\varsigma(\beta)$ и $\delta(\Theta_2,\Theta_1)=\varsigma(\alpha)$.

Критерий *гиперобъема разности покрытий* (*HyperVolume of coverage difference*) $I_{HV}(\Theta_*,\Theta)$ в качестве меры $\varsigma(\cdot)$ использует объем соответствующей части множества достижимости. Большим значениям критерия соответствует более высокое качество Парето-аппроксимации. Так, если $I_{HV}(\Theta_1,\Theta_2)>I_{HV}(\Theta_2,\Theta_1)$, то аппроксимация Θ_1 лучше аппроксимации Θ_2. Легко видеть, что в примере, представленном на рис. 8.18, $I_{HV}(\Theta_2,\Theta_1)=7$, $I_{HV}(\Theta_1,\Theta_2)=9{,}0$ и поэтому аппроксимация Θ_1 является лучшей.

Ряд критериев качества Парето-аппроксимации построен Хансеном (*M.P. Hansen*) и Яшкевичем (*A. Jaszkiewicz*) в 1998 г. на основе *функции предпочтений* ЛПР (*decision-maker's utility function*) $u(F)$. Функция $u(F)$ осуществляет отображение множества достижимости D_F во множество вещественных чисел, то есть $u:R^{|F|}\to R^1$, и формализует предпочтения ЛПР на множестве D_F, так что большим значениям этой функции соответствуют лучшие с его точки зрения решения.

Определение 8.7. Функция предпочтений $u(F)$ называется *согласованной* (*Compatible*) с отношением доминирования $\succ$, если для любых $F_j,F_k\in D_F$ таких, что $F_j\succ F_k$, справедливо неравенство $u(F_j)\ge u(F_k)$.

Определение 8.8. Функция предпочтений $u(F)$ называется *строго согласованной* (*Strictly Compatible*) с отношением доминирования $\succ$, если для любых $F_j,F_k\in D_F$ таких, что $F_j\succ F_k$, справедливо неравенство $u(F_j)>u(F_k)$.

Наборы всех функций предпочтения, согласованных и строго согласованных с отношением доминирования $\succ$, обозначаем D_C, D_{SC} соответственно.

Функцию предпочтений можно задать, например, с помощью нормы вида

$$u(F) = u(F, F^{**}) = \left\| F - F^{**} \right\|,$$

где $F^{**} = (f_1^*, f_2^*, ..., f_{|F|}^*)$ – идеальное решение. К примеру, для взвешенной L_p -манхэттенской нормы имеем

$$u(F, F^{**}) = u(F, F^{**}, \Lambda) = -\left(\sum_{k=1}^{|F|} \lambda_k abs\left(f_k - f_k^* \right)^{|F|} \right)^{\frac{1}{|F|}}, \qquad (8.13)$$

где $\Lambda = (\lambda_1, \lambda_2, ..., \lambda_{|F|})$ – вектор неотрицательных вещественных весов. Знак минус перед нормой в формуле (8.13) обеспечивает то, что решениям, более близким к идеальному решению (в смысле указанной нормы), соответствуют большие значения функции предпочтений.

Пусть тем или иным образом задан набор функций предпочтения D_u. Обозначаем $D_u(\Theta_1 > \Theta_2)$ подмножество функций предпочтения этого набора, для которых Парето-аппроксимация Θ_1 данной МЦО-задачи лучше такой же аппроксимации Θ_2.

Определение 8.9. Парето-аппроксимация Θ_1 данной МЦО-задачи *превосходит* (*outperforms*) такую же аппроксимацию Θ_2 в смысле набора функций предпочтения D_u, что обозначаем в виде $\Theta_1 \succ_{D_u}^{o} \Theta_2$, если $D_u(\Theta_1 > \Theta_2) \neq Null$ и $D_u(\Theta_1 < \Theta_2) = Null$.

Другими словами, $\Theta_1 \succ_{D_u}^{o} \Theta_2$, если в наборе D_u существует хотя бы одна функция предпочтений u, относительно которой аппроксимация Θ_1 лучше аппроксимации Θ_2, и противоположное неверно.

Определение 8.10. Парето-аппроксимация Θ_1 данной МЦО-задачи *слабо превосходит* (*weakly outperforms*) аппроксимацию Θ_2 в смысле набора функ-

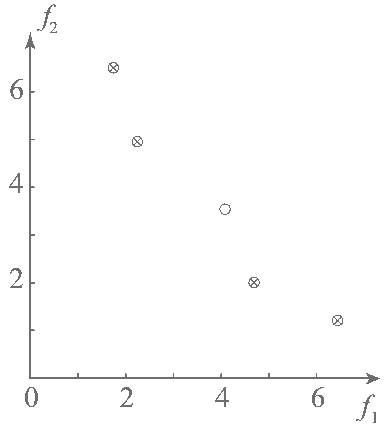


Рис. 8.19. К определению слабого превосходства $\succ_{D_u}^{w}$: ○ – решения множества Θ_1; ⊗ – решения одновременно множеств Θ_1 и Θ_2; $\Theta_1 \succ_{D_u}^{w} \Theta_2$

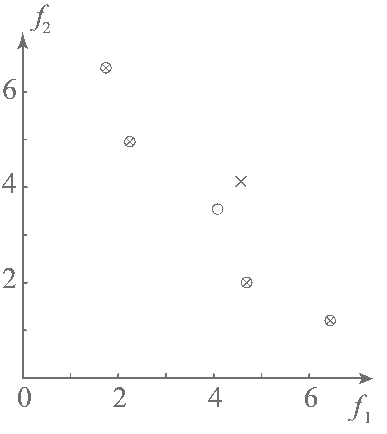


Рис. 8.20. К определению сильного превосходства $\succ_{D_u}^{s}$: ○ – решения множества Θ_1; × – решения множества Θ_2; ⊗ – решения обоих множеств; $\Theta_1 \succ_{D_u}^{s} \Theta_2$

ций предпочтения D_u, то есть $\Theta_1 \succ^{w}_{D_u} \Theta_2$, если $\Theta_1 \neq \Theta_2$ и для множества недоминируемых решений ND объединения $\Theta_1 \bigcup \Theta_2$ имеет место равенство $ND\left(\Theta_1 \bigcup \Theta_2\right) = \Theta_1$.

Определение 8.10 означает, что аппроксимация Θ_1 слабо превосходит аппроксимацию Θ_2, если для каждого решения $F_k \in \Theta_2$ существует некоторое решение $F_j \in \Theta_1$, которое либо эквивалентно решению F_k, либо доминирует его, и хотя бы одно решение аппроксимации Θ_1 не содержится в аппроксимации Θ_2. Определение иллюстрирует рис. 8.19.

Определение 8.11. Парето-аппроксимация Θ_1 данной МЦО-задачи *сильно превосходит* (*strongly outperforms*) аппроксимацию Θ_2 в смысле набора функций предпочтения D_u, то есть $\Theta_1 \succ^{s}_{D_u} \Theta_2$, если $\Theta_1 \neq \Theta_2$, $\Theta_1 \bigcup \Theta_2 = \Theta_1$ и $\Theta_2 \setminus ND\left(\Theta_1 \bigcup \Theta_2\right) \neq Null$.

Таким образом, аппроксимация Θ_1 сильно превосходит аппроксимацию Θ_2, если для каждого решения $F_k \in \Theta_2$ существует решение $F_j \in \Theta_1$, которое либо эквивалентно решению F_k, либо доминирует его, и хотя бы одно решение из Θ_2 доминируется некоторым решением из Θ_1. Определение 8.11 иллюстрирует рис. 8.20.

Определение 8.12. Парето-аппроксимация Θ_1 данной МЦО-задачи *совершенно превосходит* (*completely outperforms*) аппроксимацию Θ_2 в смысле набора функций предпочтения D_u, то есть $\Theta_1 \succ^{c}_{D_u} \Theta_2$, если $ND\left(\Theta_1 \bigcup \Theta_2\right) = \Theta_1$ и $\Theta_2 \bigcap ND\left(\Theta_1 \bigcup \Theta_2\right) \neq Null$.

Иначе говоря, $\Theta_1 \succ^{c}_{D_u} \Theta_2$, если каждое из решений $F_k \in \Theta_2$ доминируется некоторым решением $F_j \in \Theta_1$ (рис. 8.21).

Очевидно, возможны ситуации, когда аппроксимации Θ_1, Θ_2 не связаны ни одним из рассмотренных отношений слабого, сильного или совершенного превосходства, то есть являются несравнимыми (*incomparable*) в смысле этих отношений. Пример ситуации, когда указанные аппроксимации несравнимы в смысле слабого превосходства, иллюстрирует рис. 8.22.

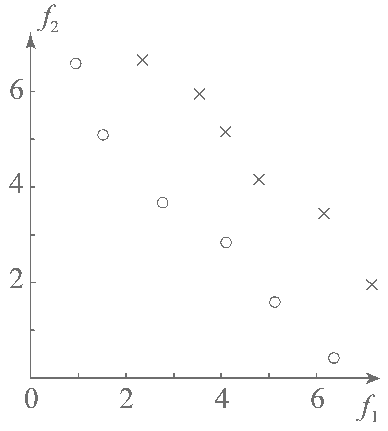


Рис. 8.21. К определению совершенного превосходства $\succ^{c}_{D_u}$: ○ – решения множества Θ_1; × – решения множества Θ_2; $\Theta_1 \succ^{c}_{D_u} \Theta_2$

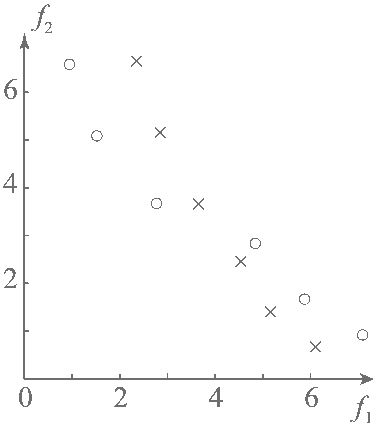


Рис. 8.22. Пример несравнимости Парето-аппроксимаций в смысле слабого отношения превосходства $\succ^{w}_{D_u}$: ○ – решения множества Θ_1; × – решения множества Θ_2

В подобной ситуации могут быть сделаны более сильные предположения о функции предпочтений ЛПР и на этой основе сформирован новый набор D_u этих функций.

Важно, что рассмотренные отношения определяют только факт превосходства одной Парето-аппроксимации над другой, но не являются мерами этого превосходства. Однако на основе данных отношений могут быть сконструированы бинарные критерии, позволяющие количественно оценить превосходство одной аппроксимации над другой.

Определение 8.13. Бинарный критерий качества Парето-аппроксимации $I_\alpha(\Theta_1, \Theta_2)$ называется *слабо совместимым* (*weakly compatible*) с отношением превосходства $\succ_{D_u}^{\alpha} \in \{\succ_{D_u}^{o}, \succ_{D_u}^{w}, \succ_{D_u}^{s}, \succ_{D_u}^{c}\}$, если для всякой пары Парето-аппроксимаций Θ_1, Θ_2 данной МЦО-задачи такой, что $\Theta_1 \succ_{D_u}^{\alpha} \Theta_2$, этот критерий показывает, что аппроксимация Θ_1 не хуже аппроксимации Θ_2.

По аналогии с определением слабо совместимого бинарного критерия вводится понятие *совместимого* (*compatible*) критерия, когда в условиях определения 8.13 этот критерий показывает, что аппроксимация Θ_1 лучше аппроксимации Θ_2.

Положим, что для каждой из функций предпочтений u множества D_u с помощью функции плотности вероятности $\xi(u)$ задана вероятность ее использования лицом, принимающим решения. Введем также в рассмотрение *функцию сравнения* (*comparison*)

$$c\left(\Theta_1, \Theta_2, u\right) = \begin{cases} 1, & u^*(\Theta_1) > u^*(\Theta_2), \\ \dfrac{1}{2}, & u^*(\Theta_1) = u^*(\Theta_2), \\ 0, & u^*(\Theta_1) < u^*(\Theta_2), \end{cases}$$

где $u^*(\Theta) = \max\limits_{F \in \Theta} u(F)$ – максимальное значение функции предпочтений на соответствующей Парето-аппроксимации.

С помощью функций $\xi(u)$, $c\left(\Theta_1, \Theta_2, u\right)$ Хансен и Яшкевич построили следующие бинарные критерии качества Парето-аппроксимации.

Критерий на основе функций предпочтения

$$I_{R_1}(\Theta_*, \Theta) = \int\limits_{u \in D_u} c(\Theta_*, \Theta, u)\xi(u)du. \tag{8.14}$$

В соответствии с данным критерием аппроксимация Θ_* лучше аппроксимации Θ, если $I_{R_1}(\Theta_*, \Theta) > 1/2$, и тем лучше, чем ближе величина $I_{R_1}(\Theta_*, \Theta)$ к единице. Аппроксимация Θ_* не хуже аппроксимации Θ, если $I_{R_1}(\Theta_*, \Theta) \geq 1/2$. Заметим, что, как следует из выражения (8.14), справедливо равенство $I_{R_1}(\Theta_*, \Theta) = 1 - I_{R_1}(\Theta, \Theta_*)$. Поэтому невозможна ситуация, когда в соответствии с данным критерием одна аппроксимация лучше другой и в то же время верно обратное утверждение.

Можно показать, что критерий $I_{R_1}(\Theta_*, \Theta)$ слабо совместим с любым набором функций предпочтений D_u, удовлетворяющим условию $D_u \subseteq D_C$.

Критерий на основе максимальных значений функции предпочтений

$$I_{R_2}(\Theta_*, \Theta) = \int\limits_{u \in D_u} \left(u^*(\Theta_*) - u^*(\Theta)\right)\xi(u)du.$$

В данном случае аппроксимация Θ_* лучше аппроксимации Θ, если $I_{R_2}(\Theta_*, \Theta) > 0$, и Θ_* не хуже Θ, если $I_{R_2}(\Theta_*, \Theta) \geq 0$. Большим значениям критерия $I_{R_2}(\Theta_*, \Theta)$ соответствует большее превосходство Θ_* над Θ. Легко видеть, что $I_{R_2}(\Theta_*, \Theta) = -I_{R_2}(\Theta, \Theta_*)$.

Критерий на основе отношения максимальных значений функции предпочтений

$$I_{R_3}(\Theta_*, \Theta) = \int\limits_{u \in D_u} \frac{u^*(\Theta_*) - u^*(\Theta)}{u^*(\Theta_*)} \xi(u) du.$$

В завершение отметим, что всесторонне оценить качество полученной Парето-аппроксимации можно только с помощью широкого набора рассмотренных и иных критериев. Поскольку эта задача является многоцелевой, ее, как и исходную МЦО-задачу, следует отнести к плохоформализованным задачам, решение которых является принципиально субъективным, полностью определяемым ЛПР. Анализ качества аппроксимации с помощью различных критериев позволяет ЛПР понять особенности построенного множества достижимости, а также особенности множества и фронта Парето МЦО-задачи, что в конечном счете обеспечивает формирование адекватных предпочтений ЛПР об этой задаче.

8.5. МЕТОДЫ ОБЕСПЕЧЕНИЯ КАЧЕСТВА ПАРЕТО-АППРОКСИМАЦИИ

Парето-аппроксимация с помощью популяционных алгоритмов предполагает получение большого числа оптимальных (в смысле близости к множеству Парето) решений с помощью одной популяции. Это противоречит сути популяционных алгоритмов, нацеленных на отыскание одного (глобального) оптимума целевой функции. Для преодоления этого противоречия требуется, часто глубокая, модификация известных популяционных алгоритмов одноцелевой оптимизации таким образом, чтобы эти алгоритмы могли отыскивать большое число локальных оптимумов. Основой такой модификации являются средства обеспечения разнообразия популяции на всех этапах решения задачи Парето-аппроксимации, наиболее известным из которых является нишевание (см. 2.6).

Представленные здесь методы обеспечения разнообразия популяции в своем большинстве разработаны для повышения эффективности эволюционных и прежде всего генетических алгоритмов одноцелевой оптимизации. В то же время данные методы могут быть использованы и используются для повышения разнообразия популяции и в других популяционных алгоритмах одноцелевой оптимизации. Например, известны работы по повышению эффективности алгоритма роя частиц на основе методов нишевания. Популяционные алгоритмы решения задачи Парето-аппроксимации в той или иной форме также используют методы такого сорта.

Выделяем следующие основные методы обеспечения разнообразия популяции: нишевание, уплотнение, дистанционирование, видообразование, пространственное структурирование. Существует точка зрения, в соответствии с которой все перечисленные методы представляют собой различные механизмы нишевания.

Еще одним известным методом повышения эффективности популяционных алгоритмов является элитизм, суть которого состоит в отборе в промежуточную популяцию лучших агентов, с тем чтобы избежать потери уже найденных хороших решений и возможного уменьшения максимальной пригодности популяции с ростом числа итераций (см. 2.5). В противоположность методам обеспечения разнообразия популяции элитизм в популяционных алгоритмах способствует интенсификации поиска. Примеры использования различных механизмов отбора на основе элитизма при решении задачи Парето-аппроксимации приведены в параграфе 8.3.

Всюду далее имеется в виду, что речь идет о максимизации фитнес-функции $\varphi(X)$.

8.5.1. Нишевание

Нишу можно определить как группу агентов, имеющих близкую приспособленность. Образование ниш, с одной стороны, снижает диверсификационные свойства популяции. С другой стороны, ниши позволяют организовать мультимодальный поиск. Методы управления нишами как с целью повышения диверсификации поиска, так и с целью обеспечения его мультимодальности называют *нишеванием* (*niching*). Некоторые такие методы применительно к генетическим алгоритмам рассмотрены в параграфе 2.6.

Идея нишевания (в генетических алгоритмах) предложена Голдбергом (*D. Goldberg*) и Ричардсоном (*J. Richardson*) в 1987 г. Идея основана на следующей природной аналогии. Ресурсы в природе ограничены, поэтому члены популяции, занимающей ту или иную экологическую нишу, должны разделять ресурсы этой ниши.

Первая проблема, которую необходимо решить в контексте нишевания, состоит в локализации ниш. Эта проблема представляет собой задачу кластеризации множеств и может быть решена с использованием классических методов *кластерного анализа* (*data clustering*), таких как *метод k-средних* (*k-means*), интеллектуальных методов (например, методы нечеткой кластеризации, методы на основе нейронных сетей Кохонена), а также на основе других методов.

В результате кластеризации текущей популяции S в пространстве параметров $\{X\}$ (в целевом пространстве $\{F\}$) получаем центры кластеров X_k^C (F_k^C) и величины $r_X(C_k)$ ($r_F(C_k)$), называемые *радиусами подобия* (*similarity radius*), или *радиусами ниш*. Здесь $k \in [1:|C|]$; $|C|$ – число выделенных кластеров. Еще одной величиной, определяющей указанные кластеры, является используемая при их формировании мера $\rho_{i,j} = \rho(s_i, s_j)$ расстояния между агентами s_i, s_j, $i, j \in [1:|S|]$. Функцию $\rho(s_i, s_j)$ называют *функцией подобия* (*similarity function*). В качестве этой функции может, например, выступать некоторая норма вида $\|X_i, X_j\|$ ($\|F_i, F_j\|$). В пространстве параметров $\{X\}$ функцию подобия обычно строят на основе евклидовой нормы, а в целевом пространстве $\{F\}$ – на основе манхэттенской нормы или $L_{|F|}$-манхэттенской нормы.

В процессе нишевания чаще всего кластеризацию в явном виде не производят, мотивируя это высокой вычислительной сложностью процедуры. Однако в практически значимых задачах, когда вычислительная сложность целевых функций очень высока, кластеризация может быть оправданной.

Второй проблемой, возникающей при нишевании, является выбор средств, обеспечивающих миграцию агентов из ниш и тем самым диссипацию ниш. Основным средством решения этой задачи является *разделение пригодности* (*fitness sharing*) агентов популяции, находящихся в одной нише, с помощью разделяющей функции (см. 2.6).

Разделение пригодности агентов, находящихся в нише C_k, осуществляют путем изменения их пригодности по формуле вида (2.5)

$$\varphi'(s_i) = \frac{\varphi(s_i)}{\sum_{j \in I_k} \phi(\rho(s_i, s_j))},$$

где I_k – набор номеров агентов, принадлежащих кластеру C_k; сумма в знаменателе представляет собой нишевый счет (*niche count*). Таким образом, текущее значение фитнес-функции уменьшается пропорционально числу и степени близости агентов в данном кластере.

Одним из методов нишевания можно считать механизм обеспечения равномерности покрытия, используемый алгоритмом «хищник-жертва». Механизм такого сорта использует также гиперкубовый алгоритм.

8.5.2. Другие методы

Уплотнение

Уплотнение (*crowding*) популяции предполагает замену близких агентов одним новым агентом. Уплотнение принято считать основной альтернативой нишеванию. Метод предложен Джонгом (*D. Jong*) в 1975 г.

Как и нишевание, уплотнение использует некоторую меру расстояния ρ между агентами популяции, чтобы определить их близость между собой. Однако уплотнение не использует понятие радиуса ниши, что снимает проблему его вычисления или априорного задания ЛПР. Важно также, что уплотнение, в отличие от нишевания, является элитарной операцией, предполагающей сохранение лучших агентов, полученных в результате ее выполнения.

Пусть s_i, s_j – агенты в своих текущих состояниях X_i, X_j, а s'_i, s'_j – эти же агенты в новых состояниях X'_i, X'_j соответственно, полученных в результате использования тех или иных миграционных операторов; $i, j \in [1:|S|]$. Суть операции уплотнения передает следующая последовательность ее основных шагов.

1) Если $\left(\rho(s_i, s'_i) + \rho(s_j, s'_j)\right) \le \left(\rho(s_i, s'_j) + \rho(s_j, s'_i)\right)$, то полагаем

$$X'_i = X_i, \text{ если } \varphi'_i > \varphi_i,$$
$$X'_j = X_j, \text{ если } \varphi'_j > \varphi_j.$$

2) В противном случае, принимаем

$$X'_j = X_i, \text{ если } \varphi'_j > \varphi_i,$$
$$X'_i = X_j, \text{ если } \varphi'_i > \varphi_j.$$

Здесь $\varphi_i = \varphi(X_i)$, $\varphi'_i = \varphi(X'_i)$ и так далее.

Известны исследования, показывающие невысокую эффективность метода уплотнения в представленном виде. Высокоэффективной модификацией простейшего метода уплотнения является метод, использующий *ограниченную турнирную селекцию* (*Restricted Tournament Selection, RTS*). Метод предложил Хэрик (*G. R. Harik*) в 1995 г. как средство повышения эффективности генетического алгоритма. Основную идею метода определяет следующая последовательность шагов.

1) Случайным образом выбираем из текущей популяции S агентов s_i, s_j.

2) Применяем к выбранным агентам операции скрещивания и мутации – получаем агентов s_i', s_j'.

3) Для каждого из агентов s_i', s_j' случайным образом сканируем популяцию S и выделяем из нее субпопуляции S_i', S_j', содержащие по $|S_i'| = |S_j'| \approx 4$ агента.

4) Выбираем в субпопуляциях S_i', S_j' агентов s_i'', s_j'', наиболее близких, соответственно, к агентам s_i', s_j' с точки зрения используемой метрики расстояний между агентами.

5) Вычисляем пригодности φ_i', φ_j агентов s_i', s_j' и пригодности φ_j'', φ_j'' агентов s_i'', s_j''. Если пригодность агента s_i' выше пригодности агента s_i'' $(\varphi_i' > \varphi_i'')$, то заменяем в популяции S агента s_i'' агентом s_i'. Аналогично поступаем с агентами s_j', s_j''.

Дистанционирование

Дистанционирование (*clearing*) имеет целью диверсификацию поиска. Главное отличие дистанционирования от нишевания состоит в том, что в данном случае отсутствует деградация пригодностей близких агентов (расположенных в одной нише). Вместо этого в каждой нише оставляют доминирующих агентов, а остальных агентов помещают вне ее. Рассматриваем схемы двух методов дистанционирования.

Положим, что агенты текущей популяции отсортированы в порядке убывания соответствующих значений фитнес-функции. Тогда основные шаги первого алгоритма передает приведенная ниже последовательность шагов.

1) По очереди для $i = 1, 2, ..., |S| - 1$ и $j = (i+1), (i+2), ..., |S|$ выполняем следующие действия.

2) Если $\rho(s_i, s_j) < r_c$, то удаляем агента s_j из текущей популяции.

3) После завершения указанного итерационного процесса взамен удаленных агентов создаем в области поиска D_X такое же число новых агентов.

Здесь r_c – *радиус подобия* (свободный параметр метода). Элитизм рассмотренной схемы обеспечивает процедура предварительной сортировки агентов. Недостатком метода является наличие указанного свободного параметра, содержательные рекомендации для задания значений которого затруднительны.

Схема второго метода дистанционирования имеет следующий вид.

1) По схеме двух первых шагов рассмотренной выше процедуры создаем промежуточную популяцию S'.

2) Путем удаления из популяции S' всех агентов, приспособленность которых ниже средней приспособленности агентов текущей популяции S, формируем основу популяции $S(t+1)$.

3) Пока число агентов популяции $S(t+1)$ не сравняется с размером популяции S, выполняем следующие действия: случайным образом выбираем из популяции S' агентов s'_i, s'_j; выполняем над выбранными агентами операцию репродукции; полученного в результате операции агента добавляем в популяцию $S(t+1)$.

Видообразование

Видообразование (*speciation*) преследует цель обеспечения мультимодальности поиска.

Нестрого вид можно определить как совокупность индивидов, которые в каком-то смысле ближе (подобны) друг к другу, чем индивиды другого вида. В популяционных алгоритмах искусственные виды (ниши) могут быть созданы путем разрешения скрещивания только подобных агентов. При этом в качестве меры близости агентов используют расстояния ρ, как в пространстве параметров $\{X\}$, так и в целевом пространстве $\{F\}$.

Метод видообразования включает в себя следующие основные шаги.

1) Случайным образом выбираем в окрестности радиуса r_c, центром которой является данный индивид s_i, индивида s_j.

2) Если $\rho(s_i, s_j) \le r_m$, то скрещиваем указанных агентов и продолжаем вычисления по общей схеме используемого популяционного алгоритма.

3) Если $\rho(s_i, s_j) > r_m$, то выбираем в указанной окрестности в качестве партнера агента s_i нового случайного агента и возвращаемся к шагу 2.

4) Если в течение фиксированного числа попыток подходящий партнер s_j не будет найден, то используем в качестве него случайного агента популяции.

Здесь $r_c > r_m > 0$ – свободные параметры алгоритма.

В случае если речь идет о популяционных алгоритмах, использующих скрещивание агентов, видообразование может иметь целью диверсификацию поиска. Разнообразие популяции в этом случае обеспечивают путем запрета скрещивания тех агентов $s_i, s_j \in S$, расстояние $\rho_{i,j}$ между которыми меньше заданной величины. Значения этой величины могут меняться по некоторому закону в процессе итераций. Иногда такой метод называют *методом ограничения скрещивания*.

Пространственное структурирование

Одной из альтернатив рассмотренным методам обеспечения разнообразия популяции является метод *пространственного структурирования* (*spatially-structured method*), предложенный Давидором (*Y. Davidor*) в 1991 г. для эволюционных алгоритмов. Метод основан на использовании так называемых *демов* (*demes*) – субпопуляций, образованных агентами, близко расположенными в пространстве поиска. Генетические операции селекции и скрещивания разрешены только между агентами, принадлежащими одному дему. Принципиальным является то обстоятельство, что каждый из агентов популяции может входить одновременно в несколько демов, что обеспечивает передачу генетической информации в пределах всей популяции.

Ряд исследований, выполненных в 90-х гг. прошлого столетия, показали невысокую эффективность данного метода, что надолго затормозило его разработку. Однако к настоящему времени на основе метода пространственного структурирования разработан ряд высокоэффективных алгоритмов обеспечения разнообразия, примерами которых являются *метод локального нишевания* (*localized niching*) и *метод локального дистанционирования* (*local clearing*), предложенные Диком (*G. Dick*) в 2005 г.

8.6. ПРИМЕРЫ ПАРЕТО-АППРОКСИМАЦИИ

Приведем примеры Парето-аппроксимации на основе генетического алгоритма, а также на основе алгоритмов роя частиц, колонии муравьев и роя пчел.

8.6.1. Парето-аппроксимация на основе генетического алгоритма

В данном пункте сначала рассматриваем примеры Парето-аппроксимации с помощью генетического алгоритма для ряда тестовых МЦО-задач. Затем приводим пример Парето-аппроксимации в задаче о спуске космического аппарата.

Тестовые задачи ZDT1 – ZDT4

Эффективность Парето-аппроксимации на основе генетического алгоритма иллюстрируем на примере широко известного элитарного генетического алгоритма недоминируемой сортировки *NSGA II*. Рассматриваем задачи двухцелевой оптимизации *ZDT1 – ZDT4* (Приложение В). Полученные Парето-аппроксимации представлены на рис. 8.23 – 8.26 соответственно. Всюду на рисунках сплошная линия показывает точный фронт Парето, а крестики – приближения к нему.

Задача *ZDT1* решена в параллелепипеде

$$\Pi = \left\{ X \mid x_1 \in [0;1],\, x_i \in [-1;1],\, i \in [2:|X|] \right\}, \tag{8.15}$$

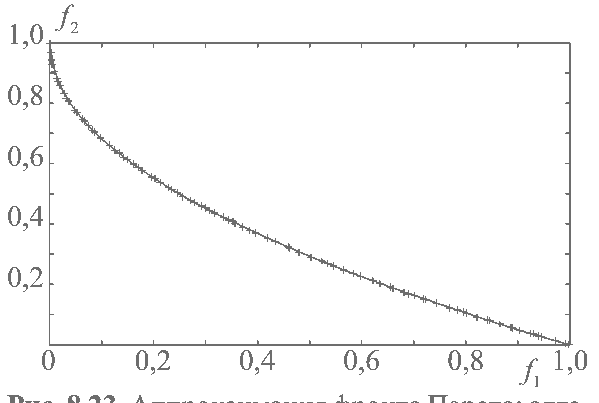


Рис. 8.23. Аппроксимация фронта Парето: алгоритм *NSGA II*; МЦО-задача *ZDT1*; $|X| = 30$; $\hat{t} = 500$

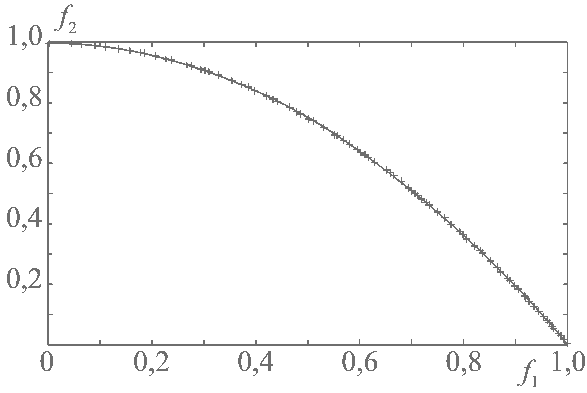


Рис. 8.24. Аппроксимация фронта Парето: алгоритм *NSGA II*; МЦО-задача *ZDT2*; $|X| = 30$; $\hat{t} = 500$

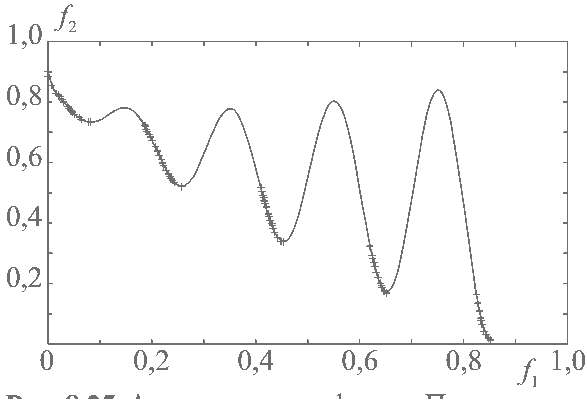


Рис. 8.25. Аппроксимация фронта Парето: алгоритм *NSGA II*; МЦО-задача *ZDT3*; $|X| = 30$; $\hat{t} = 500$

где размерность вектора варьируемых параметров $|X| = 30$. Задача имеет выпуклый фронт Парето и представляет собой простейшую задачу рассматриваемого тестового набора. Трудность задачи заключается лишь в высокой размерности пространства поиска. Рис. 8.23 показывает высокое качество полученной Парето-аппроксимации с точки зрения плотности и равномерности покрытия.

Задача *ZDT2* также решена в параллелепипеде (8.15) при размерности вектора варьируемых параметров $|X| = 30$. Особенностью задачи является невыпуклый фронт Парето. Рис. 8.24 показывает высокую эффективность алгоритма *NSGA II* и в данном случае.

Результаты решения задачи *ZDT3* представлены на рис. 8.25. Представленное решение соответствует области допустимых значений вектора варьируемых параметров в виде параллелепипеда (8.15) при $|X| = 30$. В силу разрывного, хотя и выпуклого, фронта Парето задача *ZDT3* представляет значительную сложность для многих алгоритмов Парето-аппроксимации. Рис. 8.25 показывает, что алгоритм *NSGA II* и в данном случае обеспечил достаточно высокое качество аппроксимации.

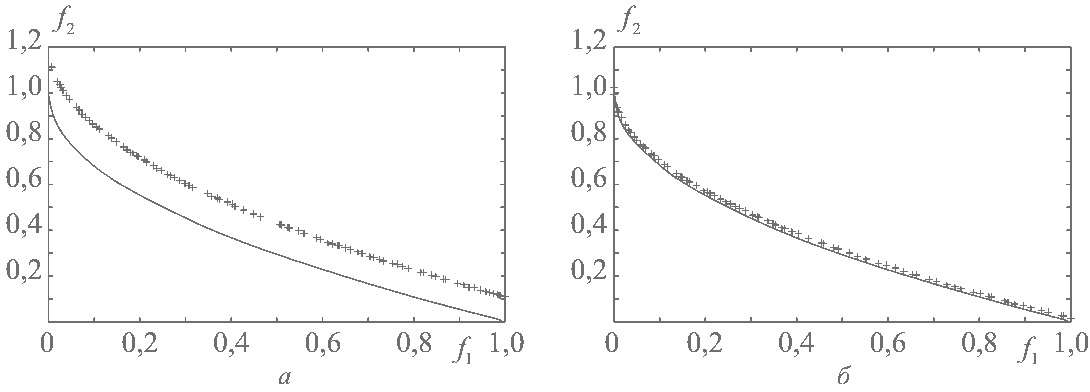


Рис. 8.26. Аппроксимация фронта Парето: алгоритм *NSGA II*; МЦО-задача *ZDT*4; $|X| = 10$: *а* – $\hat{t} = 500$; *б* – $\hat{t} = 800$

Задача *ZDT4* решена в параллелепипеде

$$\Pi = \left\{ X \mid x_1 \in [0;1],\ x_i \in [-5;5],\ i \in [2:|X|] \right\}$$

для десятимерного $(|X| = 10)$ вектора варьируемых параметров. Результаты решения задачи представлены на рис. 8.26, *а*, 8.26, *б*. Задача обладает той особенностью, что имеет множество локальных субоптимальных фронтов, способных остановить процесс Парето-аппроксимации вдали от точного фронта. Эту особенность задачи и иллюстрируют указанные рисунки.

После 500 итераций получена аппроксимация (рис. 8.26, *а*), в которой ни одно из решений не находится вблизи точного фронта Парето. Увеличение числа итераций до 800 значительно приближает аппроксимацию к точному фронту (рис. 8.26, *б*), однако по-прежнему ни одно из решений не лежит точно на фронте.

В целом приведенные результаты исследования эффективности алгоритма *NSGA II* показывают, что этот алгоритм, будучи относительно простым как в математическом плане, так и в плане реализации, обеспечивает решение даже сложных МЦО-задач с приемлемой точностью и относительно небольшими вычислительными затратами.

Задача о спуске космического аппарата

Эффективность Парето-аппроксимации на основе генетического алгоритма и алгоритма недоминируемой сортировки иллюстрируется примером (см. 2.13) двухцелевой задачи управления спуском космического аппарата в атмосфере Земли (рис. 8.27). Параметры используемого генетического алгоритма приведены в указанном параграфе.

Рис. 8.27 показывает, что прямое комбинирование генетического алгоритма и алгоритма недоминируемой сортировки не позволяет обеспечить высокие плотность покрытия фронта Парето и скорость сходимости. При числе итераций $\hat{t} = 100$ данная комбинация смогла отыскать 12 недоминируемых точек (рис. 8.27, *а*), а при увеличении числа итераций в 10 раз – всего лишь 14 таких точек (рис. 8.27, *б*). Отметим достаточно равномерное распределение найденных точек по фронту Парето.

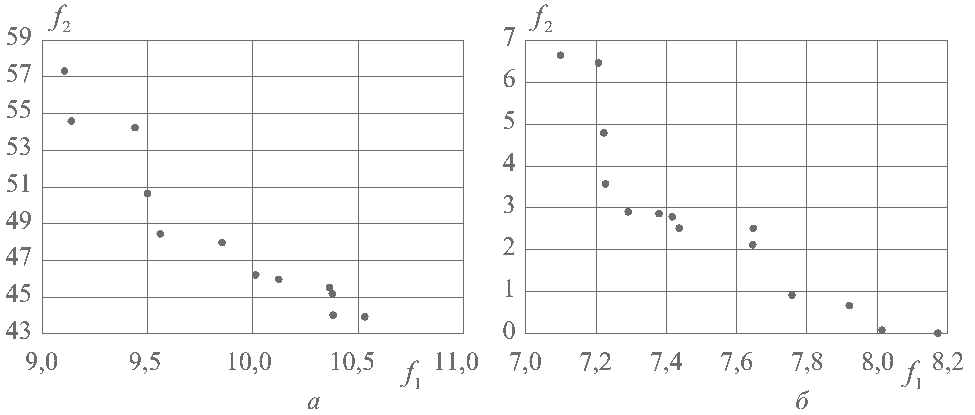


Рис. 8.27. Аппроксимация фронта Парето: *а* – $\hat{t} = 100$; *б* – $\hat{t} = 1000$

8.6.2. Аппроксимация с помощью алгоритма роя частиц

В данном пункте представлен известный алгоритм многоцелевой оптимизации роем частиц (*MOPSO*), а также его многороевая модификация *MS MOPSO*. Эффективность алгоритмов проиллюстрирована на примере простейшей задачи двухмерной двухцелевой оптимизации, а также на примере простой двухцелевой задачи оптимального управления.

Одно- и многороевый алгоритмы MOPSO

Алгоритм многоцелевой оптимизации роем частиц (*Multi-Objective Particle Swarm Optimization*, *MOPSO*) предложили Мостагхим (*S. Mostaghim*) и Тейч (*J. Teich*) в 2003 г. Алгоритм использует канонический алгоритм роя частиц и алгоритм динамического соседства (см. 8.3.2). Схема одной итерации алгоритма имеет следующий вид.

1) В соответствии с алгоритмом динамического соседства для каждой из частиц s_i популяции S находим глобально лучшую частицу s_i^{**}.

2) В соответствии с каноническим алгоритмом *PSO* перемещаем все частицы популяции на один шаг, используя в качестве локально лучшего вектора X_i^* последние сохраненные в архиве координаты частицы s_i, а в качестве глобально лучшего вектора X_i^{**} – координаты частицы s_i^{**}.

3) Если новое положение X_i' частицы s_i доминирует некоторое решение из архива Θ^X, то это решение заменяем решением X_i'. Если решение X_i' не сравнимо ни с одним из архивных решений, то добавляем его в архив и объявляем новым локально лучшим решением X_i^*. Соответствующим образом корректируем архив Θ^F.

Итерации в алгоритме *MOPSO* могут продолжаться до тех пор, пока архивы недоминируемых решений Θ^X, Θ^F не перестанут изменяться, либо до достижения заданного числа итераций.

Многороевый алгоритм MOPSO (*Multi-Swarm MOPSO*, *MS MOPSO*) предполагает параллельное (как минимум, на логическом уровне) функционирование $|\boldsymbol{S}|$ роев частиц S_j, каждый из которых имеет свои локальные архивы Θ_j^X, Θ_j^F. Кроме того, алгоритм использует глобальные архивы Θ^X, Θ^F.

Через фиксированное число итераций t_{mig} алгоритм *MS MOPSO* требует выполнения следующих основных действий.

1) Определяем лучшую s_{j_b} и худшую s_{j_w} (с точки зрения используемой фитнес-функции) частицы каждого из роев S_j и на этой основе выделяем наиболее и наименее успешные рои.

2) Худшие частицы последних роев передаем наиболее успешным роям.

3) Формируем глобальные архивы Θ^X, Θ^F, включающие в себя недоминируемые решения всех архивов Θ_j^X, Θ_j^F, и для каждого $j \in [1:|\boldsymbol{S}|]$ полагаем $\Theta_j^X = \Theta^X$, $\Theta_j^F = \Theta^F$.

Для повышения равномерности покрытия множества и фронта Парето алгоритм *MS MOPSO* использует декомпозицию области поиска D_X на $|\boldsymbol{S}|$ подобластей $D_X^j, j \in [1:|\boldsymbol{S}|]$. Начальную популяцию роя S_j полагаем равномерно распределенной в подобласти D_X^j. Для того чтобы в процессе итераций агенты популяции S_j не покидали данную подобласть, применяем метод штрафных функций.

Простейшая МЦО-задача

Эффективность Парето-аппроксимации с помощью алгоритмов *MOPSO* и *MS MOPSO* иллюстрируем двухмерной двухцелевой задачей из примера 8.1.

На рис. 8.28 представлены аппроксимации множества и фронта Парето указанной задачи, полученные с помощью алгоритма *MOPSO*. Использован рой из 10 частиц, итерации продолжались до достижения числа итераций $\hat{t} = 100$. В архивах Θ^X, Θ^F в результате оказалось по 315 решений.

Рис. 8.28 показывает, что даже для простейшей МЦО-задачи алгоритм *MOPSO* обеспечивает высокие плотности покрытия множества Парето и фронта Парето лишь в средних частях этих множеств. Гистограммы указанных плотностей представлены на рис. 8.29, где l_X, l_F – линейные размеры множества и фронта Парето соответственно.

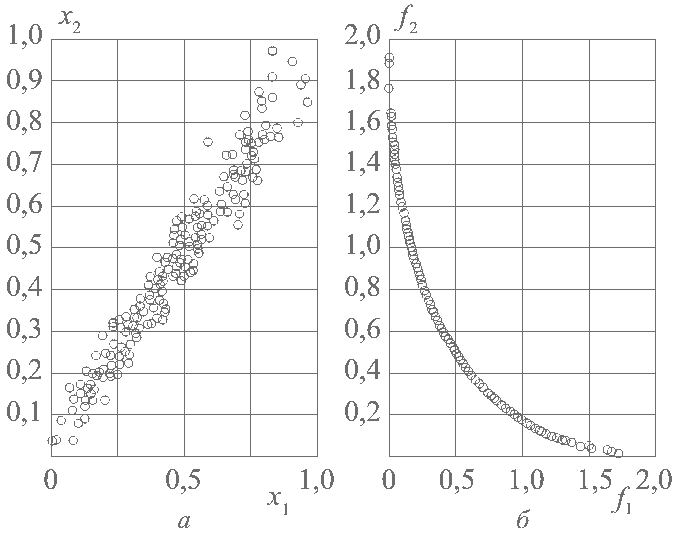


Рис. 8.28. Аппроксимация множества Парето (*а*) и фронта Парето (*б*), пример 8.1; алгоритм *MOPSO*

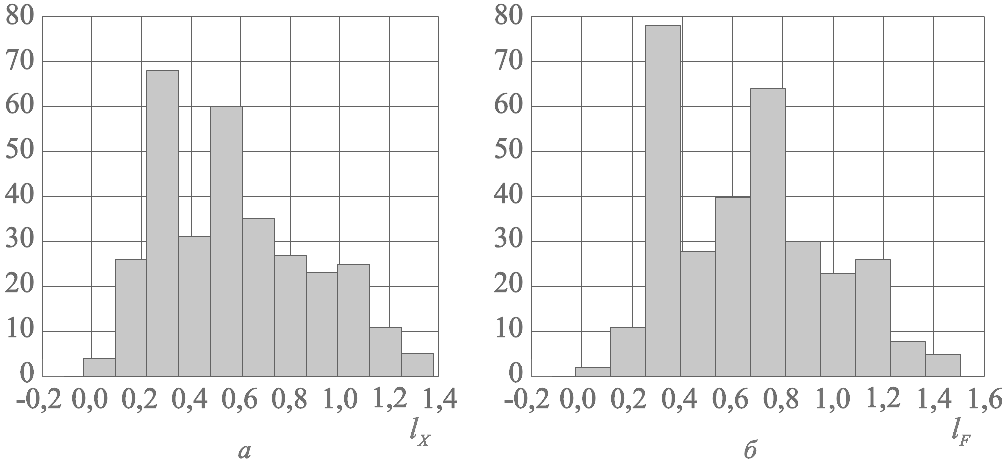


Рис. 8.29. Плотность покрытия множества Парето (*а*) и фронта Парето (*б*), пример 8.1; алгоритм *MOPSO*

Аналогичные результаты решения примера 8.1, полученные с помощью алгоритма *MS MOPSO*, иллюстрирует рис. 8.30.

Здесь использованы 100 роев из 10 частиц каждый; число итераций $\hat{t} = 100$; период миграции $t_{mig} = 1$. В архивах Θ^X, Θ^F в результате оказалось по 456 решений. Рис. 8.30 показывает, что алгоритм *MS MOPSO* может обеспечить высокую плотность и равномерность покрытия всех частей множества и фронта Парето рассматриваемой задачи, однако цена этого – весьма значительные вычислительные затраты.

Уменьшить суммарное число испытаний в рассматриваемых алгоритмах Парето-аппроксимации можно путем нишевания. Рассмотрим простейшую схему нишевания, когда недоминируемое решение добавляется в архивы Θ^X, Θ^F (если речь идет об алгоритме *MOPSO*) или в архивы Θ_j^X, Θ_j^F (в случае использования

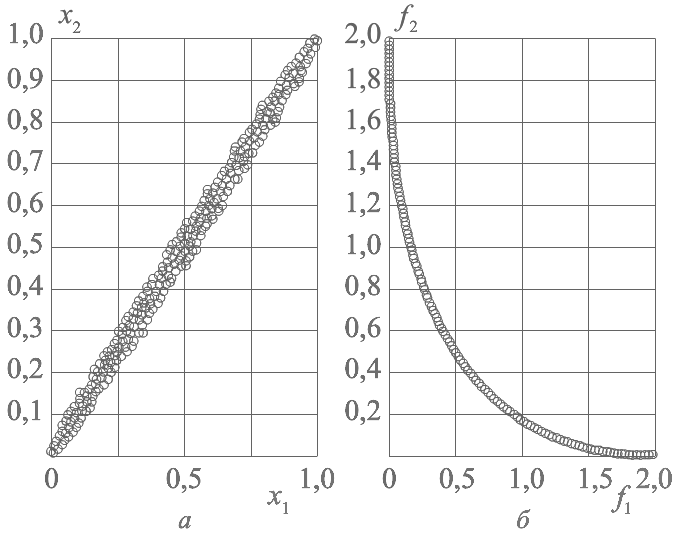


Рис. 8.30. Аппроксимация множества Парето (*а*) и фронта Парето (*б*), пример 8.1; алгоритм *MS MOPSO*

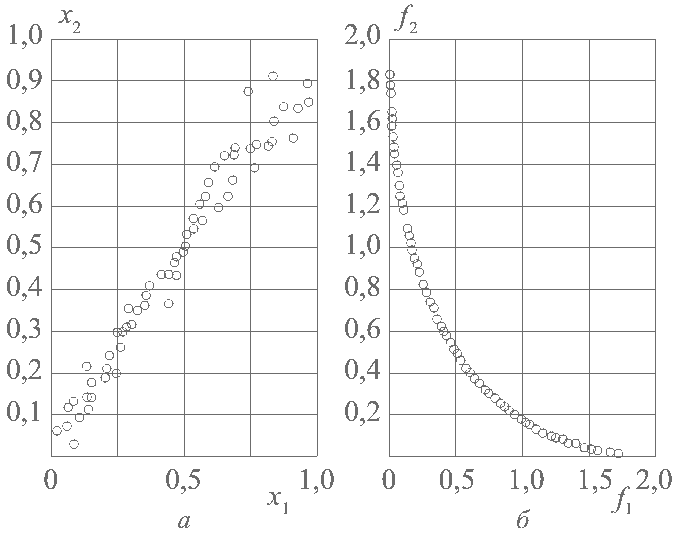


Рис. 8.31. Аппроксимация множества Парето (*а*) и фронта Парето (*б*), пример 8.1; алгоритм *MOPSO* с нишеванием

алгоритма *MS MOPSO*) только в том случае, если в окрестности радиуса r_X данного решения в соответствующем архиве отсутствуют другие решения.

Эффективность алгоритма *MOPSO* с нишеванием иллюстрирует рис. 8.31, а алгоритма *MS MOPSO* – рис. 8.32. Представленные Парето-аппроксимации получены при $r_X = 0{,}001$. Значения остальных параметров приведены выше. По завершении всех $\hat{t} = 100$ итераций число архивных решений для алгоритма *MOPSO* оказалось равным 62, а для алгоритма *MS MOPSO* – 172.

Рис. 8.31, 8.32 показывают, что нишевание обеспечивает в рассматриваемой задаче высокую равномерность покрытия множества и фронта Парето

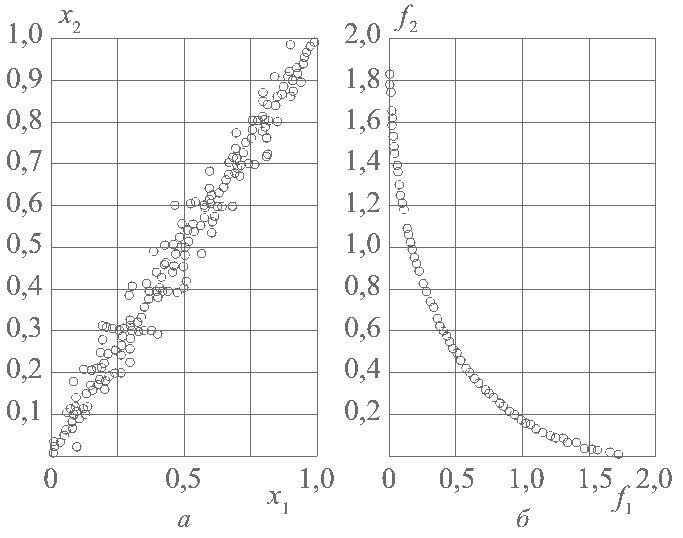


Рис. 8.32. Аппроксимация множества Парето (*а*) и фронта Парето (*б*), пример 8.1; алгоритм *MS MOPSO* с нишеванием

при использовании как алгоритма *MOPSO*, так и алгоритма *MS MOPSO*. При этом суммарное число испытаний уменьшается в несколько раз. Однако, как и следовало ожидать, одновременно понижается точность аппроксимации (что особенно заметно на аппроксимации множества Парето).

Задача оптимального управления

Тело массой m, находящееся в начальный момент времени $\tau_0 = 0$ в состоянии покоя в точке с координатой $x = 0$, может перемещаться вдоль оси $0x$ под действием силы тяги двигателя, которым оно снабжено (рис. 8.33). Максимальная по величине сила тяги u, которую может развить двигатель, равна a.

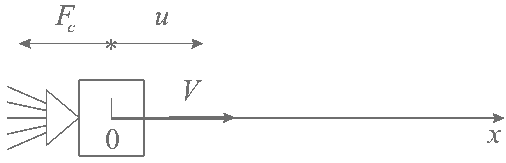


Рис. 8.33. Схема задачи

При движении тела на него действует сила сопротивления среды F_c, пропорциональная скорости движения тела V, то есть $F_c = -\kappa V$, где κ – заданный коэффициент пропорциональности. Расход горючего при работе двигателя пропорционален квадрату развиваемой им силы тяги и равен γu^2, где γ – заданная константа.

Необходимо управлять работой двигателя на промежутке времени $[0; \hat{\tau}]$ таким образом, чтобы при минимальном расходе горючего закон движения тела (то есть зависимость $x(\tau)$) как можно меньше отличался от требуемого закона движения $\phi(\tau)$.

Обозначим

$$x_1(\tau) = x(\tau), \;\; x_2(\tau) = \dot{x}_1(\tau) = V(\tau).$$

Тогда по второму закону Ньютона имеем

$$m\,\dot{x}_2(\tau) = u(\tau) + F_c = u(\tau) - \kappa x_2(\tau)\,,$$

где $\dot{x}_2(\tau) = \ddot{x}(\tau)$ – ускорение тела. Отсюда следует, что движение тела описывает система ОДУ в нормальной форме Коши

$$\begin{cases} \dot{x}_1(\tau) = x_2(\tau), \quad x_1(0) = 0, \\ \dot{x}_2(\tau) = -\dfrac{\kappa}{m} x_2(\tau) + \dfrac{1}{m} u(\tau), \quad x_2(0) = 0. \end{cases}$$

Поскольку величина силы тяги в любой момент времени не может превосходить a, имеет место ограничение $abs\big(u(\tau)\big) \le a$, то есть множество допустимых управлений имеет вид

$$D_u = \{u(\tau) \,|\, abs\,\big(u(\tau)\big) \le a\}.$$

Ограничения на вектор фазовых переменных $X(\tau) = \big(x_1(\tau), x_2(\tau)\big)$ отсутствуют.

Требование постановки задачи о минимальном отличии функций $x(\tau)$, $\phi(\tau)$ можно формализовать различными способами. Используем метрику функционального пространства $L_2[0; \hat{\tau}]$ (гильбертова пространства)

$$f_1\big(u\big) = \int_0^{\hat{\tau}} \big(x_1(\tau) - \phi(\tau)\big)^2 \, d\tau.$$

Для формализации требования минимума расхода горючего введем еще функционал

$$f_2(u) = \gamma \int_0^{\hat{\tau}} u^2(\tau)\, d\tau.$$

Таким образом, имеем двухцелевую задачу оптимального управления динамической системой

$$f_1(u) \to \min_{u(\tau)\in D_u}, \quad f_2(u) \to \min_{u(\tau)\in D_u}.$$

Подобно тому, как это сделано в примере п. 2.13, используем метод сведения поставленной задачи к задаче нелинейного программирования. Покроем интервал $[0; \hat{\tau}]$ равномерной сеткой с узлами τ_i, обозначим $U = (u_1, u_2, ..., u_{|U|})$ – вектор размерности $(|U| \times 1)$ и будем искать оптимальное управление $u^*(\tau)$ в классе кусочно-постоянных функций. Здесь $u_i = u(\tau_i)$, $i \in [0 : |U|]$.

Во введенных соглашениях и обозначениях рассматриваемая задача оптимального управления сводится к $|U|$-мерной двухцелевой задаче

$$\min_{U\in D_U} F(U) = F(U^*), \tag{8.16}$$

где вектор-функция $F(U) = (f_1(U), f_2(U))$ и множество D_U допустимых значений вектора варьируемых параметров имеет вид

$$D_U = \{u_i \mid abs(u_i) \le a, i \in [1 : |U|]\}. \tag{8.17}$$

Парето-аппроксимация поставленной задачи выполнена с помощью алгоритма *MOPSO* для следующих значений параметров задачи: $m = 1$; $\kappa = 1$; $\gamma = 1$; $a = 20$; $\hat{\tau} = 10$; $\phi(\tau) = \tau^2$; $|U| = 10$. Использован рой из 40 частиц.

Фрагмент полученной аппроксимации фронта Парето представлен на рис. 8.34. Рисунок показывает удовлетворительную равномерность покрытия (подчеркнем, что нишевание в данном случае не использовано).

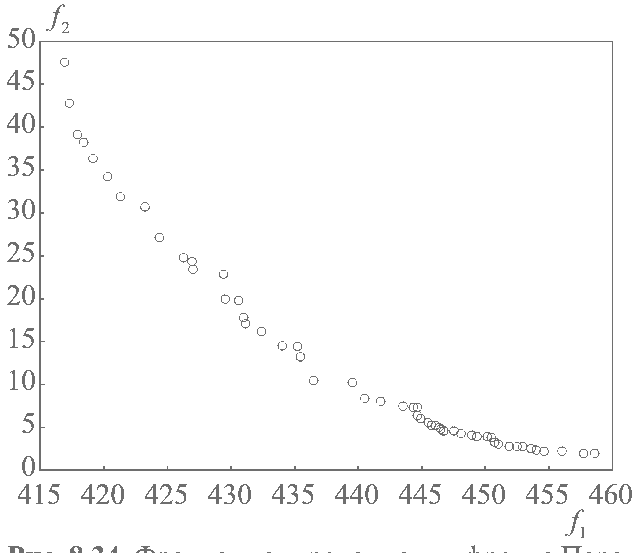


Рис. 8.34. Фрагмент аппроксимации фронта Парето задачи (8.16), (8.17): $\hat{t} = 800$

8.6.3. Аппроксимация на основе алгоритма колонии муравьев

Рассмотрим Парето-аппроксимацию в двухмерной двухцелевой задаче из примера 8.1 с помощью комбинации алгоритмов *CIAC* и динамического соседства. В случае, когда алгоритм *CIAC* использует как прямой канал, так и канал стигмергии, полученные аппроксимации множества и фронта Парето иллюстрирует рис. 8.35. Аналогичные результаты при использовании только канала стигмергии представлены на рис. 8.36.

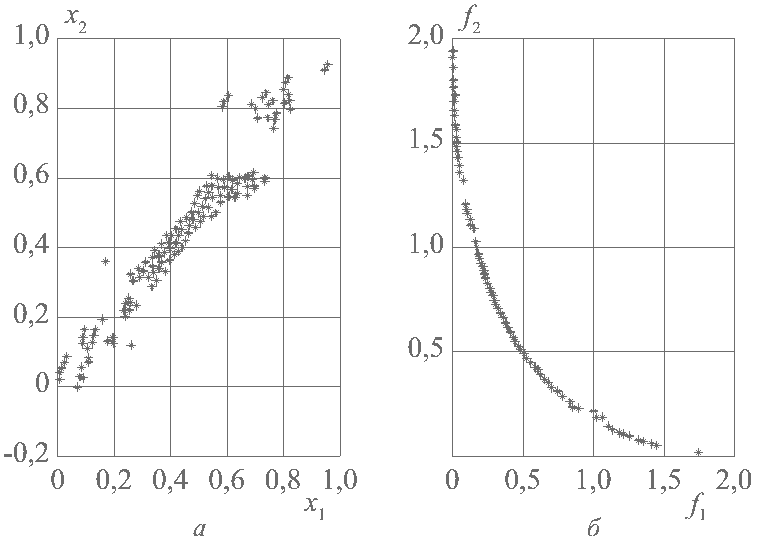


Рис. 8.35. Аппроксимация множества Парето (*а*) и фронта Парето (*б*), пример 8.1; алгоритм *CIAC* (прямой канал и канал стигмергии) и алгоритм динамического соседства

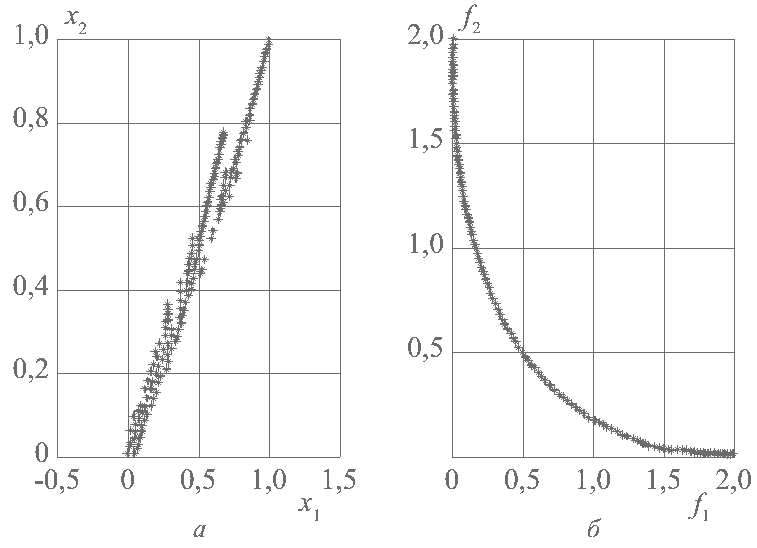


Рис. 8.36. Аппроксимация множества Парето (*а*) и фронта Парето (*б*), пример 8.1; алгоритмы *CIAC* (только канал стигмергии) и динамического соседства

Исследование выполнено при следующих значениях параметров алгоритма *CIAC*: число агентов в популяции 186; максимальное число сообщений в памяти муравья $m_{\max} = 5$; знаменатель размерного параметра $b_r = 50$; число итераций $\hat{t} = 200$.

Рис. 8.35, 8.36 показывают, что без дополнительных мер (нишевания, в первую очередь) рассмотренная комбинация алгоритмов *CIAC* и динамического соседства не обеспечивает приемлемую равномерность покрытия множества Парето и фронта Парето. Для получения высокой плотности покрытия необходима популяция агентов большого размера, а также значительное число итераций. Отметим неожиданно более высокую равномерность покрытия, которую обеспечил алгоритм *CIAC*, использующий только канал стигмергии, в сравнении с тем же алгоритмом при использовании как прямого канала, так и канала стигмергии.

8.6.4. Аппроксимация на основе пчелиного агоритма

Рассматриваемую модификацию базового алгоритма пчелиного роя, предназначенную для решения задачи Парето-аппроксимации, предложили Фам (*D. T. Pham*) и Ганбарзаде (*A. Ghanbarzadeh*) в 2007 г. Алгоритм основан на использовании недоминируемой сортировки. Шаги алгоритма, специфичные для задачи Парето-аппроксимации, имеют следующий вид.

1) Среди $|S^o|$ пчел-разведчиков определяем $|A^b|$ недоминируемых пчел и соответствующих элитных участков, где $|A^b|$ – свободный параметр алгоритма. Если число недоминируемых пчел в популяции превышает величину $|A^b|$, то выбираем первые $|A^b|$ из них. Если число недоминируемых пчел меньше $|A^b|$, то недостающих выбираем из доминируемых агентов. Называем $|A^b|$ выбранных пчел представителями соответствующих участков.

2) На каждый из $|A^b|$ элитных участков призываем n^b пчел-фуражиров (рекрутированных пчел) и вычисляем для них значения фитнес-функции.

3) Находим на каждом из участков недоминируемых пчел и, если соответствующая пчела-разведчик оказывается доминируемой, случайным образом выбираем нового представителя участка из числа недоминируемых рекрутированных пчел.

4) Среди всех пчел-представительниц выбираем недоминируемые, добавляем их в архивное множество Θ^X и редактируем это множество, исключая из него всех доминируемых пчел.

Простейшая МЦО-задача

Эффективность Парето-аппроксимации с помощью рассматриваемого алгоритма проиллюстрируем прежде всего на двухмерной двухцелевой задаче из примера 8.1. Представленные на рис. 8.37 результаты получены для случая, когда числа пчел-разведчиков, элитных участков и рекрутированных пчел равны соответственно $|S^o| = 200$, $|A^b| = 40$, $n^b = 60$. Рисунки показывают, что алгоритм обеспечивает высокую скорость сходимости – всего за $\hat{t} = 10$ итераций получено 73 недоминируемых решения, с достаточной точностью приближающих фронт Парето задачи.

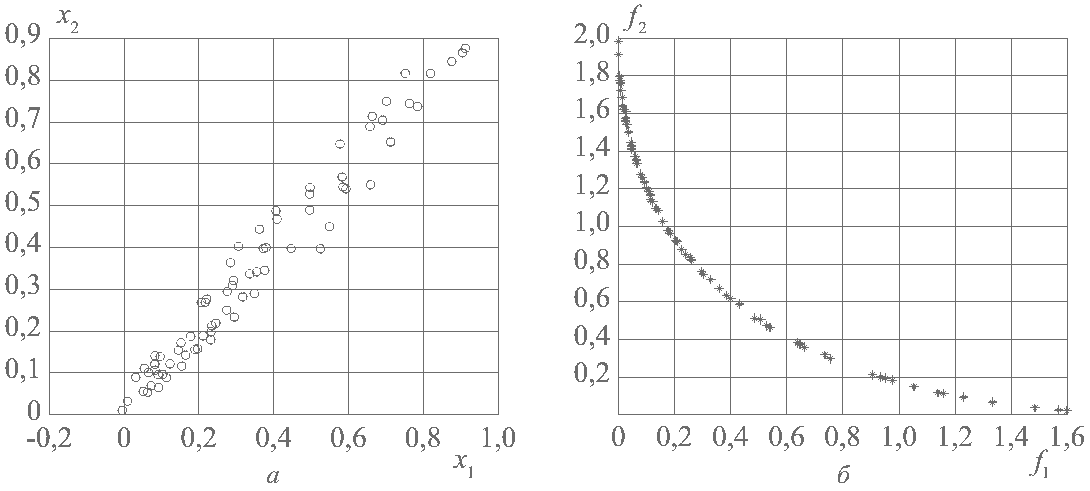


Рис. 8.37. Аппроксимация множества Парето (*а*) и фронта Парето (*б*), пример 8.1; модификация пчелиного алгоритма; $\hat{t} = 10$

Задача с невыпуклым фронтом Парето

Рассматриваем задачу минимизации целевых функций

$$f_1(X) = \left(x_1^2 + x_2^2\right)^{\frac{1}{8}}, \; f_2(X) = \left((x_1 - 0,5)^2 + (x_2 - 0,5)^2\right)^{\frac{1}{4}} \tag{8.18}$$

в области

$$D_X = \{X \mid x_i \in [-5; 10], \; i = 1, 2\}. \tag{8.19}$$

Парето-аппроксимация для данной задачи выполнена при следующих значениях свободных параметров алгоритма: $\left|S^o\right| = 100$; $\left|A^b\right| = 20$; $n^b = 30$; $\hat{t} = 200$. При этом получено 48 недоминируемых решений, хорошо аппроксимирующих фронт Парето задачи (рис. 8.38).

Заметим, что алгоритм *MOPSO* при том же числе итераций позволил отыскать для данной задачи 94 недоминируемых решения.

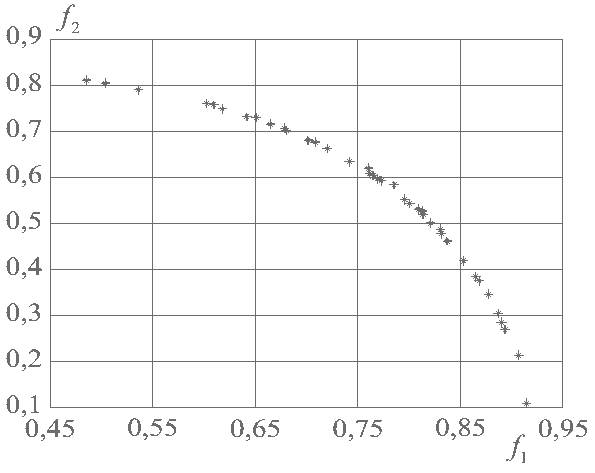


Рис. 8.38. Аппроксимация фронта Парето, задача (8.18), (8.19); модификация пчелиного алгоритма; $\hat{t} = 200$

Задача с разрывным фронтом Парето

Рассматриваемая задача является одномерной двухцелевой и имеет вид

$$f_1(x)=\begin{cases} -x, & x\le 1,\\ -2+x, & x\in(1;3],\\ 4-x, & x\in(3;4],\\ -4+x, & x>4,\end{cases}\quad f_2(X)=(x-5)^2, \tag{8.20}$$

$$D_X=\{x\,|\,x\in[-100;100]\}. \tag{8.21}$$

Исследование выполнено при тех же значениях свободных параметров алгоритма, что и в предыдущей задаче. Результаты решения задачи иллюстрирует рис. 8.39, содержащий 155 недоминируемых решений.

Алгоритм *MOPSO* в тех же условиях позволил найти всего 96 решений. Таким образом, по сравнению с предыдущей задачей рассматриваемый пчелиный алгоритм демонстрирует существенно более высокую эффективность, чем алгоритм *MOPSO*.

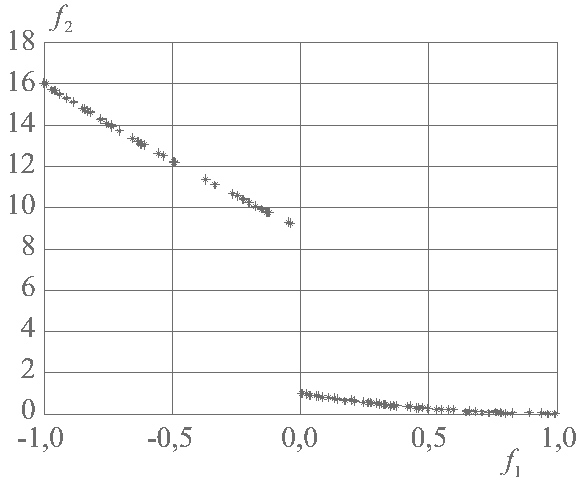


Рис. 8.39. Аппроксимация фронта Парето, задача (8.20), (8.21); модификация пчелиного алгоритма: $\hat{t}=200$

Задача с множественными разрывами фронта Парето

Рассматриваем двухмерную двухцелевую задачу вида

$$f_1(X)=x_1,\quad f_2(X)=a(x_2)b(X), \tag{8.22}$$

$$D_X=\{X\,|\,x_i\in[0;1],\,i=1,2\}, \tag{8.23}$$

где

$$a(x_2)=1+9\,x_2,$$

$$b(X)=1-\left(\frac{f_1(x_1)}{a(x_2)}\right)^{\frac{1}{2}}-\left(\frac{f_1(x_1)}{a(x_2)}\right)\sin\left(10\pi f_1(x_1)\right).$$

Используем следующие значения свободных параметров пчелиного алгоритма: $\left|S^o\right| = 200$; $\left|A^b\right| = 40$; $n^b = 60$; $\hat{t} = 100$. При этом получено 55 недоминируемых решений (рис. 8.40).

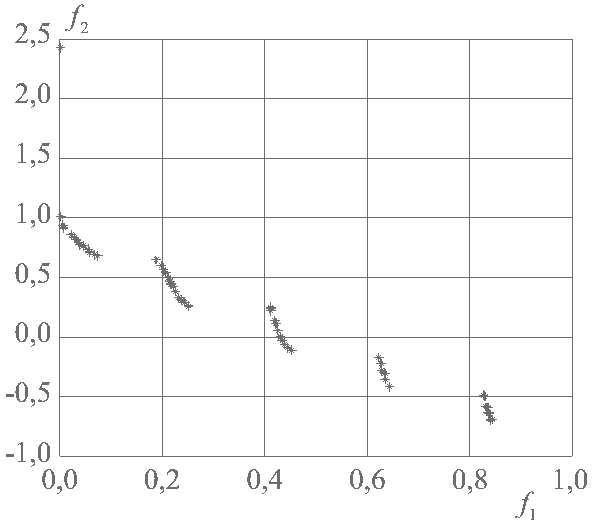


Рис. 8.40. Аппроксимация фронта Парето, задача (8.22), (8.23); модификация пчелиного алгоритма: $\hat{t} = 100$

Те же 100 итераций алгоритма *MOPSO* позволили найти 48 недоминируемых решений задачи (8.22), (8.23). Таким образом, и для сложной МЦО-задачи, имеющей множественные разрывы фронта Парето, рассматриваемая модификация пчелиного алгоритма показывает более высокую эффективность в сравнении с алгоритмом *MOPSO*.

Вопросы для самопроверки

1. Приведите постановку задачи многоцелевой оптимизации. Определите понятия: множество достижимости, отношение доминирования, отношение строгого доминирования, фронт Парето, множество Парето, Парето-аппроксимация.

2. Дайте общие схемы решения задачи многоцелевой оптимизации с помощью алгоритма зондирования, а также априорных, апостериорных и адаптивных алгоритмов.

3. Представьте схемы основных непопуляционных алгоритмов Парето-аппроксимации.

4. Опишите популяционные алгоритмы Парето-аппроксимации на основе лексикографической турнирной селекции.

5. Приведите схемы основных популяционных алгоритмов Парето-аппроксимации на основе чередующихся целевых функций.

6. Представьте схемы наиболее известных популяционных алгоритмов Парето-аппроксимации, использующих ранжирование агентов популяции.

7. Дайте схемы некоторых популяционных алгоритмов Парето-аппроксимации, не использующих ранжирование агентов популяции.

8. Назовите основные унарные и бинарные критерии оценки качества Парето-аппроксимации.

9. Перечислите основные средства обеспечения разнообразия популяции.

10. Определите метод обеспечения разнообразия популяции путем нишевания.

Глава 9

ПАРАЛЛЕЛЬНЫЕ ПОПУЛЯЦИОННЫЕ АЛГОРИТМЫ ПОИСКОВОЙ ОПТИМИЗАЦИИ

Популяционные алгоритмы поисковой оптимизации могут требовать больших вычислительных ресурсов даже при невысокой вычислительной сложности целевой функции (целевых функций) в задаче одноцелевой (многоцелевой) оптимизации. Например, обычной является ситуация, когда решение задачи оптимизации требует сотен тысяч и миллионов вычислений значений этой (этих) функций. С другой стороны, в практически значимых задачах одно испытание может требовать нескольких часов работы современной однопроцессорной ЭВМ даже в том случае, когда этот процессор является многоядерным. В силу данных обстоятельств весьма привлекательным выглядит использование параллельных ЭВМ для ускорения оптимизационных вычислений.

Некоторые параллельные алгоритмы сами по себе могут оказывать положительный эффект на процесс оптимизации. Так, Сколички (*Z. Skolicki*) показал, что для некоторых целевых функций параллельный алгоритм на основе островной модели параллелизма (см. ниже) более эффективен, чем соответствующий последовательный алгоритм, даже если не учитывать ускорение, полученное в результате использования параллельной ЭВМ.

Многие непопуляционные алгоритмы стохастической оптимизации могут быть распараллелены, но, как правило, весьма искусственным образом. Важным достоинством популяционных алгоритмов является их естественный параллелизм, поскольку в этих алгоритмах значительное число агентов на логическом уровне эволюционирует параллельно.

В разработку и развитие параллельных популяционных алгоритмов оптимизации внесли свой вклад многие исследователи, среди которых особо следует отметить Грефенстетта (*J. Grefenstette*). Технический отчет, выпущенный им в свет в 1981 г., предвосхитил очень многое в теории и практике этих алгоритмов.

Представляем классификацию и основные типы параллельных ЭВМ (первый параграф). Приводим постановку задачи балансировки загрузки этих ЭВМ, а также основные методы ее решения (второй параграф). Даем обзор известных методов распараллеливания, на основе которых могут быть построены параллельные популяционные алгоритмы (третий параграф). Рассматриваем примеры параллельных популяционных алгоритмов одно- и многоцелевой оптимизации (четвертый параграф).

9.1. КЛАССИФИКАЦИЯ И ОСНОВНЫЕ ТИПЫ ПАРАЛЛЕЛЬНЫХ ЭВМ

Нестрого *архитектуру* вычислительной системы можно определить как абстрактное представление ЭВМ, какой ее видит программист, занимающийся низкоуровневым программированием. Полное описание архитектуры сис-

темы включает в себя основные форматы представления данных и способы их адресации, состав аппаратных средств ЭВМ и их характеристики, принципы организации вычислительного процесса. *Структурой* вычислительной системы называют совокупность ее аппаратных средств и основных связей между ними.

9.1.1. Классификация параллельных вычислительных систем

Классификация Флинна. Наибольшее распространение получила классификация вычислительных систем, предложенная в 1966 г. профессором Стэнфордского университета Флинном (*M. J. Flynn*). Эта классификация охватывает только два классификационных признака – тип потока команд и тип потока данных (рис. 9.1, 9.2).

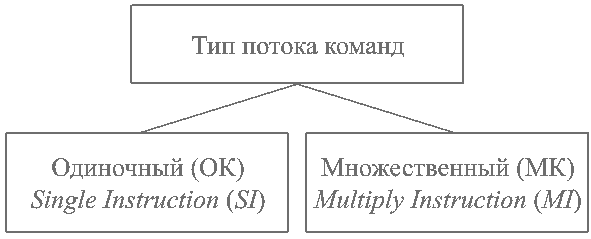


Рис. 9.1. К классификации Флинна: классификация по типу потока команд

В *одиночном потоке команд* в один момент времени может выполняться только одна команда, определяющая в этот момент времени работу всех или по крайней мере многих устройств вычислительной системы. Во *множественном потоке команд* в каждый момент времени может выполняться несколько команд, каждая из которых определяет работу только одного или лишь нескольких (но не всех) устройств этой системы. Можно сказать, что в одиночном потоке последовательно выполняются отдельные команды, а во множественном потоке – группы команд.

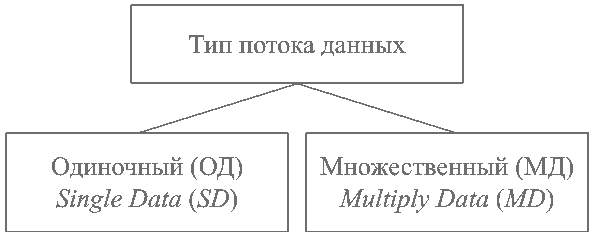


Рис. 9.2. К классификации Флинна: классификация по типу потока данных

Одиночный поток данных обязательно предполагает наличие в вычислительной системе только одного устройства оперативной памяти и одного процессора. Однако заметим, этот процессор может быть сколь угодно сложным, так что процесс обработки каждой единицы информации в потоке может тре-

бовать выполнения многих команд. *Множественный поток данных* состоит из многих зависимых или независимых одиночных потоков.

В соответствии с классификацией Флинна все вычислительные системы делят на четыре типа – *SISD* (ОКОД), *MISD* (МКОД), *SIMD* (ОКМД) и *MIMD* (МКМД).

SISD-система представляет собой классическую однопроцессорную ЭВМ фон-неймановской архитектуры.

На *MISD-систему* существуют различные точки зрения. По одной из них за всю историю развития вычислительной техники *MISD*-системы не были созданы. По другой (менее распространенной) точке зрения к *MISD*-системам относят векторно-конвейерные вычислительные системы. Будем придерживаться первой точки зрения.

SIMD-система содержит некоторое число процессоров, которые синхронно (как правило) выполняют одну и ту же команду над разными данными. *SIMD*-системы разделяют на классы векторно-конвейерных и векторно-параллельных (матричных) вычислительных систем.

MIMD-система содержит много процессоров, которые (как правило, асинхронно) выполняют разные команды над разными данными. Подавляющее большинство современных супер-ЭВМ имеют архитектуру *MIMD* (по крайней мере на верхнем уровне иерархии).

Классификации Флинна позволяет по принадлежности параллельной ЭВМ к классу *SIMD* или *MIMD* сделать сразу понятными базовые принципы ее функционирования. Известным недостатком классификации Флинна является «переполненность» класса *MIMD*.

Такие классификации архитектур параллельных ЭВМ, как классификация Фенга (*T. Feng*), Шора (*J. E. Shore*), Хэндлера (*W. Handler*), Шнайдера *(L. Snyder)* и других авторов, уточняют и детализируют различные аспекты представленных классов этих ЭВМ.

Классификация по типу строения оперативной памяти. По типу строения оперативной памяти параллельные ЭВМ разделяют на системы с общей памятью, системы с распределенной памятью и системы с физически распределенной, но логически общей памятью (гибридные системы).

В параллельных вычислительных системах *с общей* (*разделяемой*) памятью (*common memory systems*) значение, записанное в память одним из процессоров, напрямую доступно для другого процессора. Общая память обычно имеет высокую *пропускную способность* (*bandwidth*) и низкую *латентность* (*latency*) при передаче информации между процессорами, но при условии, что не происходит одновременного обращения нескольких процессоров к одному и тому же элементу памяти. К общей памяти доступ разных процессоров системы осуществляется, как правило, за одинаковое время. Поэтому такую память называют еще *UMA-памятью* (*Unified Memory Access*). Системы с общей памятью называются также *сильносвязанными* системами или *мультипроцессорами*.

В системах *с распределенной* памятью (*Distributed Shared Memory, DSM*) каждый процессор имеет свою локальную память с локальным адресным пространством. Для систем с распределенной памятью характерно наличие боль-

шого числа быстрых каналов передачи данных, которые связывают отдельные части этой памяти с отдельными процессорами. Обмен информацией между частями распределенной памяти осуществляется обычно относительно медленно. Системы с распределенной памятью называются также *слабосвязанными* системами или *мультикомпьютерами*.

Системы *с гибридной памятью* (*Non-Uniform Memory Access*, *NUMA*) имеют память, которая физически распределена по различным частям системы, но логически разделяема (образует единое адресное пространство). Такая память называется еще *логически общей* (*разделяемой*) памятью (*logically shared memory*).

Память современных параллельных вычислительных систем является многоуровневой, иерархической.

Классификация по типу коммуникационной сети. Нам понадобится далее понятие *диаметра d* коммуникационной сети, которым называют максимальное расстояние между двумя произвольными узлами этой сети. *Расстояние* между двумя узлами коммуникационной сети есть не что иное, как число ребер, которые образуют кратчайший путь между этими узлами в графе коммуникационной сети.

Топологически коммуникационные сети разделяют на сети *с фиксированной* топологией и на *реконфигурируемые* сети. С другой стороны, с этой же точки зрения коммуникационные сети разделяются на *регулярные* и *нерегулярные* сети.

Чаще всего в параллельных вычислительных системах используют регулярные сети с фиксированной топологией. Среди этих сетей наиболее распространены топологии, представленные на рис. 9.3 – 9.5, где $P = \left(p_i,\ i \in [1:|P|]\right)$ – процессоры системы, а $|P|$ – их число.

В сети с топологией линейки (рис. 9.3) каждый процессор, за исключением первого и последнего, связан непосредственно только с двумя соседними (с предыдущим и последующим) процессорами.

Сеть с топологией кольца получается из топологии «линейка» путем соединения между собой ее первого и последнего процессоров (рис. 9.4).

Можно считать, что сеть с топологией двумерной решетки (рис. 9.5) получается объединением нескольких сетей с топологией линейки.

Из сети с топологией двумерной решетки может быть получена сеть с топологией *тор* (*torus*). Для этого достаточно соединить между собой граничные процессоры по горизонтали $p_{1,1} - p_{1,n},\ p_{2,1} - p_{2,n}, \ldots,\ p_{m,1} - p_{m,n}$ и по вертикали $p_{1,1} - p_{m,1},\ p_{1,2} - p_{m,2}, \ldots, p_{1,n} - p_{m,n}$.

В полносвязной сети *клика* (*full connect*) между любой парой процессоров существует непосредственная связь. Такая сеть обеспечивает минимальное

Рис. 9.3. Некоммутируемая коммуникационная сеть с топологией *линейка* (*linear array*, *farm*): $d = O(|P|)$

Рис. 9.4. Некоммутируемая коммуникационная сеть с топологией *кольцо* (*ring*): $d = O(|P|)$

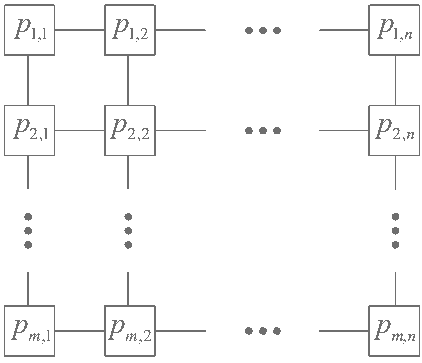


Рис. 9.5. Некоммутируемая коммуникационная сеть с топологией *двумерная решетка* (*mesh*): $d = O\left(\sqrt{|P|}\right)$; $|P| = m \cdot n$

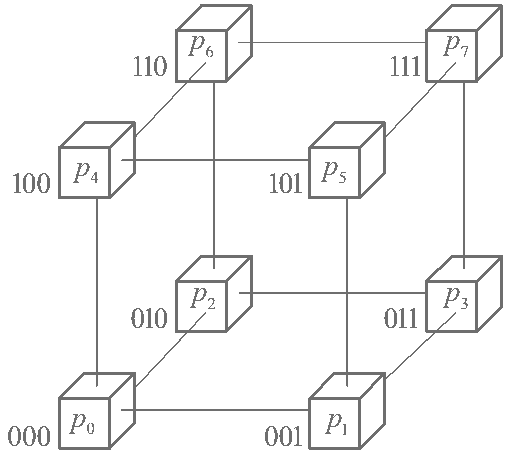


Рис. 9.6. Трехмерный гиперкуб: $|P| = 8$; $d = 3$

время обмена данными между любыми двумя процессорами системы, однако сложно реализуема и имеет высокую стоимость при большом числе процессоров. Диаметр сети равен, очевидно, единице.

Кроме того, известна коммуникационная сеть типа *бинарное дерево* (*binary tree*), диаметр которой равен $O(\log_2|P|)$; сеть, имеющая топологию *звезда* (*star*) и диаметр, равный двум; большое число других типов коммуникационных сетей.

Нельзя не упомянуть коммуникационные сети с топологией *гиперкуб* (*hypercube*), диаметр которых равен $d = \log_2|P|, |P| = 2^d$, где d одновременно – число измерений гиперкуба. Гиперкубовую топологию параллельной вычислительной системы иллюстрирует рис. 9.6, на котором бинарные номера процессоров даны в отраженном коде Грея, как это принято в системах с такой топологией. Диаметр данной сети медленно растет с ростом числа процессоров в системе, например, при $|P| = 2^6 = 64$ диаметр сети $d = 6$, а при $|P| = 2^{10} = 1024$ диаметр сети $d = 10$.

Большие параллельные вычислительные системы имеют обычно *иерархическую* коммуникационную сеть, в которой все процессоры разбиваются на группы с одинаковым и относительно небольшим числом процессоров (обычно несколько десятков) в каждой. Физически группу процессоров монтируют в один блок или стойку. Каждая из групп имеет коммуникационную сеть одной топологии. Группы процессоров объединяются между собой коммуникационной сетью другой топологии.

Классификация по степени однородности. По степени однородности различают однородные и неоднородные параллельные вычислительные системы. Обычно при этом имеется в виду тип используемых процессоров.

В *однородных* (*гомогенных*) вычислительных системах используют одинаковые, а в *неоднородных* (*гетерогенных*) вычислительных системах – различные процессоры.

В настоящее время большинство высокопроизводительных систем относится к классу однородных систем с общей памятью или к классу однородных систем с распределенной памятью. Однородные *MIMD*-системы с общей памятью принято называть системами с *симметричной* многопроцессорной архитектурой (*Symmetric Multi-Processing*, *SMP*). Все процессоры *SMP*-системы имеют равноправный (симметричный) доступ к общей оперативной памяти. Однородные *MIMD*-системы с распределенной памятью называются системами с *массивно-параллельной* архитектурой (*Massive Parallel Processing*, *MPP*).

Нечто среднее между *SMP*- и *MPP*-системами представляют *NUMA*-системы.

Рассмотренные классификационные признаки параллельных вычислительных систем не исчерпывают всех возможных их характеристик. Существует, например, еще разделение систем по степени согласованности режимов работы (*синхронные и асинхронные* вычислительные системы), по способу обработки (с пословной обработкой и *ассоциативные* вычислительные системы), по жесткости структуры (системы *с фиксированной структурой* и системы *с перестраиваемой структурой*), по управляющему потоку (системы *потока команд* и системы *потока данных*) и т. д.

Современные высокопроизводительные системы имеют, как правило, иерархическую структуру. Например, на верхнем уровне иерархии система является *MIMD*-системой, каждый процессорный узел которой представляет собой *MIMD*- или *SIMD*-систему.

9.1.2. Основные типы параллельных ЭВМ

Мультипроцессоры (*multi-processors*, *shared-memory systems*), как отмечалось выше, относятся к классу *MIMD*-систем, а значит содержат некоторое число процессоров, которые асинхронно выполняют разные команды над разными данными. Отличительным признаком мультипроцессоров является наличие общей памяти, разделяемой всеми процессорами системы (рис. 9.7, *а*).

В мультипроцессорах адресное пространство всех процессоров является единым. Это значит, что если в программах различных процессоров встречается одна и та же переменная, то для получения или изменения значения этой

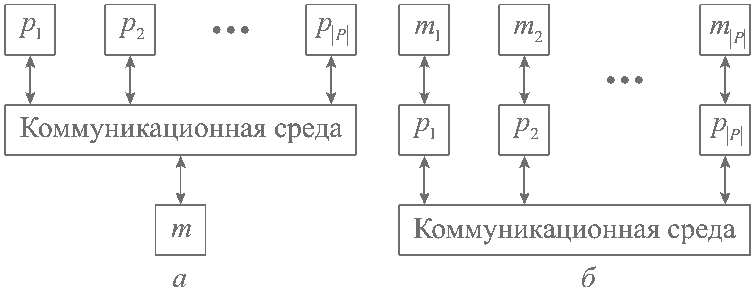


Рис. 9.7. Структура мультипроцессора (*а*) и мультикомпьютера (*б*): p_i – процессор; m_i – модуль памяти

переменной процессоры будут обращаться к одной физической ячейке общей памяти. Это обстоятельство имеет как положительные, так и отрицательные последствия.

Положительной стороной наличия общей памяти является отсутствие необходимости физически перемещать данные между коммутирующими программами, что исключает затраты времени на межпроцессорный обмен. Важно также, что мультипроцессорная система может при этом функционировать под управлением единственной копии операционной системы (обычно *UNIX*-подобной) и не требует индивидуальной настройки каждого процессорного узла.

Отрицательное последствие состоит в том, что поскольку одновременное обращение нескольких процессоров к общим данным может привести к получению неверных результатов, необходимы системы синхронизации параллельных процессов и обеспечения когерентности памяти. Кроме того, вследствие высокой частоты обращения процессоров к общей памяти в мультипроцессорах чрезвычайно повышаются требования к пропускной способности коммуникационной среды. Последнее обстоятельство ограничивает число процессоров в мультипроцессорах несколькими десятками.

В классе мультипроцессоров выделяют упомянутые выше *NUMA*-системы и *SMP*-системы.

Для программирования мультипроцессорных параллельных вычислений на языках *C*, *C*++ и Фортран широко используют программное средство *OpenMP* (*Open Multi-Processing*), включающее в себя совокупность директив компилятору, библиотечных процедур и так называемых переменных окружения.

Мультикомпьютеры (*multicomputers*, *distributed memory systems*) также как мультипроцессоры относятся к классу *MIMD*-систем. Отличительным признаком мультикомпьютеров является наличие распределенной памяти. Структура мультикомпьютерной системы приведена на рис. 9.7, *б*.

Вследствие простоты своей архитектуры наибольшее распространение в настоящее время получили именно мультикомпьютеры. Поскольку эти ЭВМ не имеют общей памяти, межпроцессорный обмен в них осуществляется только через коммуникационную сеть с помощью механизма передачи сообщений.

Каждый процессор в мультикомпьютере имеет независимое адресное пространство. Поэтому наличие переменной с одним и тем же именем в программах разных процессоров приводит к обращению к физически разным ячейкам собственной памяти этих процессоров. Это обстоятельство требует физического перемещения данных между коммутирующими программами в разных процессорах. Чаще всего основная часть обращений производится каждым процессором к собственной памяти. Поэтому требования к коммутационной среде ослабляются. В результате число процессоров в мультикомпьютерных системах может достигать десятков и сотен тысяч.

Пиковая производительность крупнейших параллельных вычислительных систем с общей памятью ниже пиковой производительности наиболее мощных систем с распределенной памятью. Стоимость систем с общей па-

мятью выше стоимости аналогичных по производительности систем с распределенной памятью.

Среди мультикомпьютеров выделяют системы с массивно-параллельной архитектурой (*MPP*-системы) и вычислительные кластеры.

Для программирования мультикомпьютеров в подавляющем большинстве случаев используют программный интерфейс *MPI* (*Message Passing Interface*), реализации которого существуют для языков Фортран 77/90, *C* и *C++*.

Графические процессорные устройства (*Graphics Processing Unit*, *GPU*) с некоторыми оговорками относят к классу *SIMD*-систем, точнее говоря – к классу векторно-параллельных вычислительных систем. Современные *GPU* являются мощными специализированными ЭВМ, которые успешно используются для решения целого ряда сложных вычислительных задач. В настоящее время одно *GPU* позволяет получить реальную производительность до нескольких Тфлопс и обеспечить многократное ускорение вычислений по сравнению с центральным процессором *host*-ЭВМ. Важно, что помимо высокой производительности *GPU* обладают низким энергопотреблением и низкой относительной стоимостью. Основная идея применения *GPU* в качестве универсального вычислителя состоит в использовании шейдерных процессоров как универсальных сопроцессоров для операций с плавающей точкой.

Известны два современных производителя *GPU* для высокопроизводительных вычислений – компании *AMD* и *NVIDIA*. При общности базовых принципов, архитектуры *GPU* этих производителей существенно отличаются.

Рассмотрим в качестве примера архитектуру *GPU AMD Radeon Cayman*, используемого в графических картах последнего поколения серии *Radeon* 6900. Наиболее производительная карта этой серии содержит 24 *SIMD*-блока, состоящих из 64 скалярных процессоров. Блоки могут коммутировать между собой через глобальную разделяемую память емкостью 64 Кб. *SIMD*-блок имеет 32 Кб локальной разделяемой и 8 Кб кэш-памяти. Каждый потоковый процессор содержит пять арифметико-логических устройств (АЛУ), четыре из которых являются АЛУ общего назначения, а один – специализированным АЛУ, позволяющим выполнять такие команды как, например, sin, cos, log, exp и т. д.

Итого, *GPU Radeon 6900* содержит 1536 суперскалярных потоковых процессоров. Такое число вычислительных устройств обеспечивает пиковую производительность *GPU*, равную 2,7 Тфлопс при вычислениях с одинарной точностью и 675 Гфлопс – с двойной точностью.

В качестве примера *GPU* компании *NVIDIA* приведем передовую архитектуру этой компании *Fermi*, которая лежит в основе семейства видеопроцессоров *GeForce 500* и видеопроцессоров для научных вычислений *Tesla*. Графическое процессорное устройство *NVIDIA GF110*, являющееся в настоящее время наиболее производительным из этой серии, обладает 16 потоковыми мультипроцессорами, каждый из которых содержит по 32 суперскалярных потоковых процессора (ядра), так что общее число потоковых процессоров достигает 512. Каждый мультипроцессор содержит четыре специализированных блока для вычисления

математических функций, 32 768 регистров, 64 Кб регистровой памяти, используемой для кэширования данных и коммуникации ядер. Видеопроцессор имеет общую память типа *GDDR* объемом до 4 Гб. Производительность видеокарт семейства *GeForce 500* достигает 1600 Гфлопс с одинарной точностью и 170 Гфлопс с двойной точностью. Производительность видеопроцессоров *Tesla* при вычислениях с двойной точностью достигает 500 Гфлопс.

С позиций программирования вычислительная система на основе *GPU* представляет собой сложную иерархическую структуру, имеющую несколько адресных пространств и ряд специфических механизмов. Широкому использованию *GPU* для решения вычислительных задач до недавнего времени препятствовало отсутствие адекватных средств программирования, не связанных с программированием графики. Ситуация изменилась с появлением в 2008 году технологии *CUDA* (*Compute Unified Device Architecture*) компании *NVIDIA*, включающей в себя компилятор параллельного расширения языка *C* для программирования *GPU* и набор библиотек. В настоящее время для программирования *GPU* также возможно использование технологии параллельного программирования *OpenCL* (*Open Computing Language*).

Модель программирования *CUDA* предполагает группирование потоков в *блоки потоков* (*thread block*), которые объединяют в одно- или двумерную сетку (*grid*) блоков. Каждый блок может включать в себя до 1024 потоков, число блоков практически не ограничено. Потоки в одном блоке взаимодействуют между собой при помощи разделяемой памяти *GPU*. Блоки выполняются конкурентно, назначения блоков на потоковые мультипроцессоры производятся аппаратным планировщиком.

В отличие от программирования классических процессоров, когда обычно доступно только два вида памяти (регистровая и оперативная), при использовании технологии *CUDA* программисту может быть доступно шесть видов памяти – самые быстрые *регистровая* (*registers*) и *разделяемая* (*shared*) памяти, более медленные *локальная* (*local*), *глобальная* (*global*) и *константная* (*constant*) памяти, а также *текстурная* (*texture*) память, предназначенная главным образом для работы с последовательными выборками текстур.

Технология *OpenCL* представляет собой инструментальное программное обеспечение, предназначенное для разработки параллельных программ для различных графических и центральных процессоров. В качестве основы *OpenCL* использует стандарт *ISO C99* языка *C*.

Стандарт *OpenCL 1.0* определяет четыре пространства памяти – частное (*private*) пространство, локальное (*local*), константное (*constant*) и глобальное (*global*). Частная память схожа с регистровой памятью и может использоваться только одним мультипроцессором. Локальная память может быть использована программными единицами, объединенными в рабочую группу. В константной памяти могут храниться постоянные данные, доступные по чтению для всех мультипроцессоров *GPU*. Глобальная память находится на графической карте вне *GPU* и может использоваться всеми его мультипроцессорами. Основным отличием технологии *OpenCL* является предварительная компиляция параллельной программы драйвером *OpenCL*, осуществляющим оптимизацию и трансляцию программы в аппаратно-специфичный код.

9.2. БАЛАНСИРОВКА ЗАГРУЗКИ ПАРАЛЛЕЛЬНОЙ ЭВМ

Задача балансировки загрузки параллельной ЭВМ представляет собой частный случай более широкой задачи *оптимального отображения* (*mapping*) вычислительного алгоритма на архитектуру параллельной ЭВМ. Рассмотрим прежде эту задачу.

9.2.1. Задача оптимального отображения алгоритма на архитектуру параллельной ЭВМ

Данная задача может быть поставлена в различных терминах. Используем графовые модели вычислительного алгоритма и параллельной ЭВМ.

Пусть $\langle A, C^A \rangle$ – ациклический граф данного вычислительного алгоритма. Здесь $A = (a_1, a_2, ..., a_{|A|})$ – вершины графа, отождествляемые с составными частями (операторами) алгоритма, $C^A = (c^A_{i,j}, i, j \in [1:|A|], j \neq i)$ – ребра графа, соответствующие информационным связям между операторами, так что $c^A_{i,j}$ – объем информации в байтах, который оператор a_i передает оператору a_j. Таким образом, неравенство $c^A_{i,j} \neq 0$ означает, что операторы a_i, a_j связаны информационно, а равенство $c^A_{i,j} = 0$ – не связаны. Обозначим $v^A_1, v^A_2, ..., v^A_{|A|}$ вычислительные сложности (числа соответствующих арифметических операций) операторов $a_1, a_2, ..., a_{|A|}$. Под оператором понимаем достаточно крупный фрагмент алгоритма, которому в программной реализации соответствует блок, подпрограмма, функции, процедура и т. п.

Положим, что имеется *MIMD*-параллельная ЭВМ с универсальными процессорами, каждый из которых может выполнить программный код, соответствующий любому из операторов $a_1, a_2, ..., a_{|A|}$. Пусть $\langle P, C^P \rangle$ – граф (возможно, циклический) данной ЭВМ. Здесь $P = (p_1, p_2, ..., p_{|P|})$ – вершины графа, соответствующие процессорам; $C^P = (c^P_{i,j}, i, j \in [1:|P|], j \neq i)$ – ребра графа, отождествляемые с коммуникационной сетью, где $c^P_{i,j}$ – пропускная способность в байтах в секунду линии связи процессоров p_i, p_j без учета латентности этой связи. Условие $c^P_{i,j} \neq 0$ означает, что имеется возможность прямой передачи данных от процессора p_i процессору p_j, а при условии $c^P_{i,j} = 0$ указанной возможности нет. Время в секундах, необходимое для организации передачи данных от процессора p_i процессору p_j, обозначаем $e_{i,j}$. Величина $e_{i,j}$ имеет смысл приведенной *латентности* коммуникационной сети ЭВМ, которая складывается из аппаратной латентности этой сети и задержек, обусловленных коммуникационными программными средствами (например, используемой коммуникационной библиотекой). Производительности процессоров $p_1, p_2, ..., p_{|P|}$, измеренные в числе арифметических операций в секунду, обозначаем $v^P_1, v^P_2, ..., v^P_{|P|}$ соответственно.

Введем в рассмотрение $|A| \times |P|$ *отображающую матрицу* (*mapping matrix*)

$$Z = (z_{i,j}, i \in [1:|A|], j \in [1:|P|]),$$

компоненты которой имеют следующий смысл: $z_{i,j} = 1$ – оператор a_i назначен на выполнение процессору p_j; $z_{i,j} = 0$ – оператор a_i не назначен на выполне-

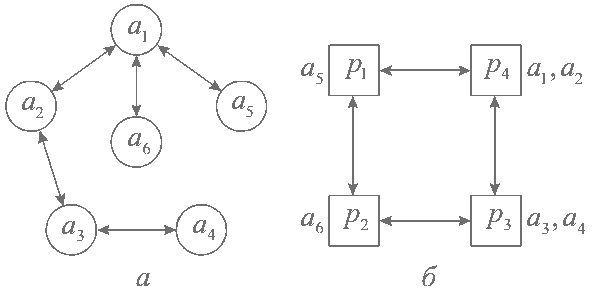


Рис. 9.8. К примеру 9.1: *а* – информационные связи операторов $a_1,...,a_6$ в графе $\langle A, C^A \rangle$; *б* – коммуникационные связи процессоров $p_1,..., p_4$ в графе $\langle P, C^P \rangle$ и отображение операторов $a_1,...,a_6$ на этот граф

ние процессору p_j. Заметим, что отображающая матрица Z не выполняет отображение ребер C^A графа $\langle A, C^A \rangle$ на ребра C^P графа $\langle P, C^P \rangle$.

Пример 9.1. Рассмотрим отображение графа $\langle A, C^A \rangle$ на граф $\langle P, C^P \rangle$ (рис. 9.8).

Отображение операторов $a_1,...,a_6$ на сеть процессоров $p_1,..., p_4$, приведенное на рис. 9.8, *б*, выполнено с помощью отображающей матрицы

$$Z = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

Обмен данными, например между операторами a_4, a_6 может быть реализован как по цепочке p_3, p_4, p_1, так и по цепочке p_3, p_2, p_1 •

Введем в рассмотрение критерий оптимальности отображения $\mu(Z)$. Тогда задачу оптимального отображения графа $\langle A, C^A \rangle$ на граф $\langle P, C^P \rangle$ можно формализовать в виде задачи целочисленного математического программирования: найти оптимальную отображающую матрицу Z^* такую, что

$$\min_{Z \in D_Z} \mu(Z) = \mu(Z^*), \tag{9.1}$$

где D_Z – множество допустимых отображающих матриц.

Если операторы $a_1, a_2,..., a_{|A|}$ выполняются многократно и вычислительная сложность их меняется в процессе вычислений, то рассмотренное *статическое* отображение (*static mapping*) может быть неэффективным. В этом случае задачу отображения приходится решать многократно в ходе вычислений, то есть использовать *динамическое* отображение (*dynamic mapping*).

Как всякая задача отображения графа на граф, задача оптимального отображения является *NP*-сложной. Отсюда следует, что принципиально не су-

ществует универсальных методов решения этой задачи. Основным методом приближенного решения задачи оптимального отображения является метод *балансировки загрузки* (*load balancing*). Идея метода состоит в распределении операторов алгоритма по процессорам таким образом, чтобы суммарная вычислительная и коммуникационная загрузки процессоров были примерно одинаковы. При этом не учитывают коммуникационные загрузки процессоров, обусловленные транзитными обменами, конфликты при обменах вследствие перегрузки коммуникационной сети, а также время $e_{i,j}$ на организацию обменов.

9.2.2. Постановка задачи балансировки загрузки

Задача балансировки загрузки также может быть поставлена в различных терминах. Рассматриваем постановку задачи в терминах математического программирования, а также в терминах теории графов. Напоминаем, что речь идет о балансировке загрузки *MIMD*-вычислительной системы.

Постановка в терминах математического программирования

Назовем *вычислительной* загрузкой (*work load*) процессора p_j величину

$$\mu_j^w(Z) = \frac{1}{v_j^p} \sum_{k=1}^{|A|} z_{k,j} v_k^A, \ j \in [1:|P|], \tag{9.2}$$

а его *коммуникационной* загрузкой (*communication load*) – величину

$$\mu_j^c(Z) = \sum_{k=1}^{|P|} \sum_{l=1}^{|A|} \sum_{m=1}^{|A|} z_{l,k} z_{m,j} \frac{c_{m,l}^A}{c_{j,k}^P}, \ j \in [1:|P|]. \tag{9.3}$$

Легко видеть, что сумма в выражении (9.2) представляет собой суммарную вычислительную сложность операторов, назначенных на выполнение процессору p_j. Отсюда следует, что сама величина $\mu_j^w(Z)$ есть не что иное, как суммарное время обработки этим процессором всех указанных операторов. Аналогично величина (9.3) равна суммарному времени, необходимому для выполнения всех обменов тех операторов, которые назначены на выполнение процессору p_j, с операторами, выполняемыми на других процессорах системы и информационно связанными с первыми операторами.

Во введенных обозначениях критерий оптимальности отображения можно записать в виде

$$\mu(Z) = \max_{j \in [1:|P|]} \mu_j = \max_{j \in [1:|P|]} (\mu_j^w(Z) + \mu_j^c(Z)), \tag{9.4}$$

где величина $\mu(Z)$ имеет смысл максимального суммарного времени на вычисления и коммуникации. Задача целочисленного нелинейного программирования (9.1)–(9.4) представляет собой формальную постановку задачи балансировки загрузки.

Формула (9.4) определяет скалярную аддитивную свертку целевых функций $\mu_j^w(Z)$, $\mu_j^c(Z)$ и формализует компромисс между двумя следующими тен-

денциями. С одной стороны, увеличение числа процессоров, между которыми распределены вычисления, уменьшает общее время решения задачи. С другой стороны, распределение вычислений между многими процессорами увеличивает коммуникационные расходы, что увеличивает общее время решения задачи.

В соответствии с общими принципами решения задач многоцелевой оптимизации, вместо свертки (9.4) можно предложить большое число иных способов сведения данной двухцелевой задачи к одноцелевой.

Задача (9.1)–(9.4) также является *NP*-сложной, однако существуют эффективные приближенные методы решения этой задачи (особенно без учета коммуникационной загрузки процессоров).

В некоторых случаях задача (9.1)–(9.4) может быть решена точно. Пусть, например, граф $\langle A, C^A \rangle$ представляет собой двухмерную регулярную решетку, вычислительные и коммуникационные сложности операторов $a_1, a_2, ..., a_{|A|}$ постоянны и одинаковы, $|A| > |P|$ и величины $|A|$, $|P|$ кратны. Тогда оптимальным является разрезание этой решетки на $|P|$ одинаковых полос (по числу процессоров в системе) и назначение каждой из этих полос своему процессору.

Постановка в графовых терминах

В терминах теории графов задача балансировки может быть сформулирована следующим образом. Необходимо разрезать граф $\langle A, C^A \rangle$ на $|P|$ частей так, чтобы суммарный вес разрезанных ребер был минимальным, а суммарные веса вершин, входящих в разрезанные части, равны. Очевидно, что если в графе $\langle A, C^A \rangle$ имеет место передача большого числа коротких сообщений, то целесообразно минимизировать не вес разрезанных ребер, а их число, поскольку основное время при передаче короткого сообщения тратится на инициализацию самого процесса передачи данных.

Если операторы $a_1, a_2, ..., a_{|A|}$ выполняются многократно и вычислительная сложность их меняется в процессе вычислений, то *статическая балансировка* загрузки может быть неэффективной. В этом случае приходится использовать *динамическую балансировку*.

Задачи статической и динамической балансировки загрузки могут решаться программными и аппаратными средствами.

Отметим следующее обстоятельство. Проблема обеспечения сбалансированной загрузки процессоров является одним из основных препятствий на пути эффективного использования параллельных ЭВМ. Несмотря на наличие большого числа высокоэффективных алгоритмов статической и динамической балансировки загрузки, типичный коэффициент интегральной загрузки большинства систем массового параллелизма составляет по разным оценкам всего от 3 % до 10 %.

9.2.3. Методы статической балансировки загрузки

Методы статической балансировки загрузки отличаются, в первую очередь, способом формализации задачи балансировки. С этой точки зрения выделяем методы, использующие формализацию в виде задачи математического программирования, а также методы, основанные на графовой постановке

задачи. В качестве соответствующих примеров рассматриваем метод упаковки в контейнеры и иерархический графовый метод.

Метод упаковки в контейнеры

В методе упаковки в контейнеры учитывается только вычислительная загрузка процессоров (и не учитывается их коммуникационная загрузка). Основой такого упрощения задачи балансировки является то, что современные процессорные модули параллельных ЭВМ содержат в своих сетевых адаптерах специальные коммуникационные процессоры, которые ведут обмен данными без участия центральных процессоров этих модулей.

Для простоты записи рассматриваем гомогенную *MIMD*-параллельную ЭВМ, производительность всех процессоров которой одинакова и равна v^P опер/с.

Поставим в соответствие процессорам $p_1, p_2, ..., p_{|P|}$ рассматриваемой параллельной ЭВМ набор из $|P|$ двумерных контейнеров единичной ширины и не фиксированной высоты, а оператору a_i – плоский объект упаковки, имеющий единичную ширину и высоту, равную вычислительной сложности v_i^A этого оператора; $i \in [1:|A|]$. Данные соглашения иллюстрирует рис. 9.9, на котором принято, что распределены все операторы $a_1, a_2, ..., a_{|A|}$ и ни один из этих операторов не распределен дважды. Процессору p_j распределены операторы a_{j_k}, $j_k \in I_j$, где наборы индексов I_j удовлетворяют условиям

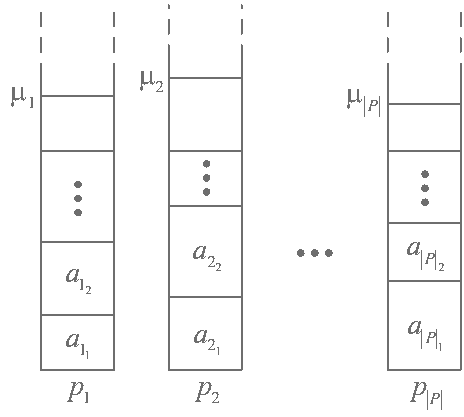


Рис. 9.9. К методу упаковки в контейнеры

$$\bigcup_{j=1}^{|P|} I_j = I = (1, 2, ..., |A|); \; I_j \bigcap_{j \neq k} I_k = Null.$$

Величина μ_j на рис. 9.9 представляет собой суммарную высоту упаковки контейнера p_j:

$$\mu_j = \sum_{j_k \in I_j} v_{j_k}^A, \quad j \in [1:|P|].$$

Таким образом, задача балансировки загрузки сведена к задаче минимизации максимальной высоты упаковки

$$\min_{Z \in D_Z} \max_{j \in [1:|P|]} \mu_j(Z) = \min_{Z \in D_Z} \mu(Z) = \mu(Z^*). \tag{9.5}$$

Для решения задачи (9.5) часто используют дискретные варианты генетического алгоритма. С той же целью могут быть использованы дискретные модификации любого из рассмотренных популяционных алгоритмов глобальной оптимизации.

Иерархический графовый метод

Наиболее известным методом приближенного решения задачи балансировки в графовой постановке является иерархический графовый метод. В соответствии с этим методом разрезание графа $\langle A, C^A \rangle$ на подграфы осуществляют в три этапа: рекурсивное огрубление графа; рекурсивная бисекция огрубленного графа; восстановление исходного графа.

Рекурсивное огрубление графа. В процессе рекурсивного огрубления графа $\langle A, C^A \rangle = \langle A, C^A \rangle^0$ строим последовательность графов $\langle A, C^A \rangle^k$ меньшего размера; $k = 1, 2, \ldots$. При построении графа $\langle A, C^A \rangle^{k+1}$ в графе $\langle A, C^A \rangle^k$ выделяем некоторое число подмножеств вершин, и вершины каждого из этих подмножеств стягиваем в одну мультивершину графа $\langle A, C^A \rangle^{k+1}$ (рис. 9.10). Вес полученной мультивершины принимаем равным сумме весов входящих в него вершин. Аналогично в мультиребра графа $\langle A, C^A \rangle^{k+1}$ стягиваем соответствующие ребра графа $\langle A, C^A \rangle^k$, и вес мультиребра определяем путем суммирования весов стягиваемых ребер. Из графа $\langle A, C^A \rangle^{k+1}$ по той же схеме получается граф $\langle A, C^A \rangle^{k+2}$ и так далее до достижения приемлемого размера результирующего графа.

Для формирования подмножеств узлов, стягиваемых в один мультиузел, используют, в основном, подходы на основе паросочетаний и создании мультиузлов из сильносвязных групп узлов.

В первом подходе мультивершину получаем путем стягивания смежных вершин. Идею этого подхода обычно формулируют в терминах *паросочетаний* – наборов ребер графа, в котором любые два ребра не инцидентны общей вершине. Достаточно эффективным алгоритмом формирования паросочетаний является алгоритм *случайных паросочетаний*. Суть алгоритма состоит в том, что если мультивершина $a_{i_1}^k$ не включена ни в одно из паросочетаний, то случайным образом выбираем одну из ее смежных мультивершин $a_{i_2}^k$, и мультиребро $(c_{i_1,i_2}^A)^k$ включаем в паросочетание. В качестве мультивершины $a_{i_2}^k$ может быть также выбрана вершина с максимальным весом мультиребра $(c_{i_1,i_2}^A)^k$. В этом случае алгоритм паросочетаний называют алгоритмом *паросочетаний из тяжелых ребер*.

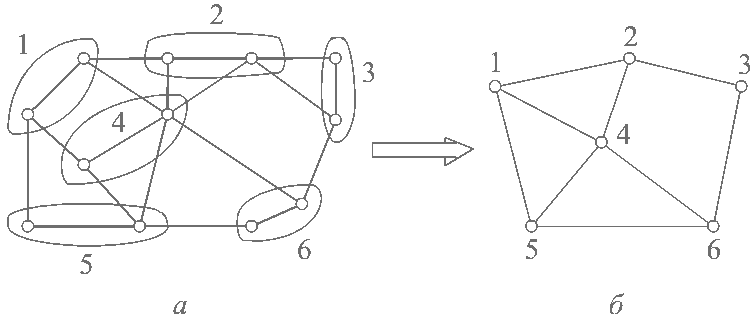


Рис. 9.10. К рекурсивному огрублению графа: 1–5 – мультиузлы: *а* – граф $\langle A, C^A \rangle^k$; *б* – граф $\langle A, C^A \rangle^{k+1}$

Во втором подходе в мультивершины стягиваем группы сильносвязанных вершин (с высоким уровнем внутригрупповой связности), которые слабо взаимодействуют с другими группами вершин (низкая межгрупповая связность). Для выделения таких групп используют различные эвристические алгоритмы, например алгоритм *паросочетаний из тяжелых клик*. Напомним, что кликой называется граф, все вершины которого являются смежными. В качестве меры близости графа к клике обычно используют *реберную плотность* графа $\frac{\beta(\beta+1)}{\alpha(\alpha+1)}$, где α, β – числа вершин и дуг графа соответственно. В алгоритме паросочетаний из тяжелых клик паросочетание образует свободная мультивершина $a_{i_1}^k$ и та из ее смежных мультивершин $a_{i_2}^k$, которая обеспечивает максимальную реберную плотность мультивершины, полученной объединением мультивершин $a_{i_1}^k$, $a_{i_2}^k$.

Рекурсивная бисекция огрубленного графа. На втором этапе иерархического графового алгоритма выполняем рекурсивное деление пополам графа, полученного на первом этапе алгоритма. Деление выполняется таким образом, чтобы каждая из частей содержала примерно половину узлового веса первоначального графа.

Бисекция графа может быть выполнена с использованием различных алгоритмов:

– алгоритм спектрального деления пополам;
– поисковый алгоритм бисекции графа;
– алгоритм бисекции графа путем наращивания.

Суть алгоритма *спектрального деления пополам* поясним на примере простейшей задачи балансировки, когда вычислительные сложности всех операторов $a_i, i \in [1:|A|]$ одинаковы и величина $|A|$ четна. В этом случае задачу бисекции графа $\langle A, C^A \rangle^k$ решают, исходя из требования, чтобы числа вершин в каждом из подграфов были одинаковы и суммарное число разрезанных ребер было минимальным. Как отмечалось выше, последнее условие позволяет с высокой точностью балансировать коммуникационную загрузку процессоров в случае, когда объемы данных, передаваемых между ними, невелики.

Определим $|A| \times |A|$ матрицу смежности $Q = (q_{i,j},\ i, j \in |A|)$ графа $\langle A, C^A \rangle$ такую, что $q_{i_1,i_2} = 1$, если вершины a_{i_1}, a_{i_2} связаны между собой ребром, и $q_{i_1,i_2} = 0$ – в противном случае. Определим также $|A| \times |A|$ диагональную матрицу $B = diag(b_i),\ i \in [1:|A|]$ степеней соответствующих вершин графа $\langle A, C^A \rangle$, где величина b_i равна степени вершины a_i (числу ребер, инцидентных этой вершине). Кроме того, введем в рассмотрение $|A| \times |A|$ *матрицу Лапласа* для графа $\langle A, C^A \rangle$

$$Y = B - Q.$$

Определим нормализованные собственные векторы $U_1, U_2, ..., U_{|A|}$ матрицы Y, а также соответствующие собственные значения $\lambda_1 \le \lambda_2 \le ... \le \lambda_{|A|}$ этой матрицы.

Утверждение 9.1. Искомая оптимальная бисекция графа $\langle A, C^A \rangle$ может быть найдена следующим образом:

– находим взвешенное среднее $\tilde{u}_2$ компонент вектора $\bar{U}_2 = \sqrt{|A|}\, U_2$;

– относим вершины графа $\langle A, C^A \rangle^k$, соответствующие значениям $u_{2,i} < \tilde{u}_2$, к первому подграфу, а остальные вершины – ко второму подграфу;

– если несколько величин $u_{2,i}$ имеют значение $\tilde{u}_2$, то распределяем соответствующие вершины между подграфами равномерно •

Идея *поискового* алгоритма бисекции графа состоит в следующем. Задаем некоторое исходное разбиение графа на два подграфа. Итерационно делаем попытки перенести одну или несколько вершин из одного подграфа в другой так, чтобы уменьшить значения указанного выше критерия оптимальности разбиения.

В данном алгоритме результирующее разбиение графа сильно зависит от исходного приближения. Поэтому обычно приходится решать задачу для некоторого числа случайных исходных разбиений (по некоторым рекомендациям достаточно примерно 10 разбиений).

Другой простой путь деления графа пополам состоит в использовании алгоритма бисекции графа *путем наращивания*. Алгоритм начинает бисекцию с одной вершины и наращивает область вокруг этой вершины до тех пор, пока не будет получено два удовлетворительных (с точки зрения указанного выше критерия оптимальности) подграфа.

Данный алгоритм также чувствителен к выбору вершины, с которой начинается рост графа. Поэтому, как и в предыдущем алгоритме, приходится решать задачу многократно для некоторого числа случайных исходных вершин.

Восстановление исходного графа. На данном этапе каждый из подграфов, на которые был разделен огрубленный граф на предыдущем этапе, восстанавливаем до соответствующей части исходного графа. Поскольку полученные подграфы имеют значительно большее число степеней свободы, чем огрубленные подграфы, можно попытаться улучшить полученное разделение с помощью какого-либо локального алгоритма оптимизации.

9.2.4. Методы динамической балансировки загрузки

Известно большое число методов динамической балансировки загрузки, различающихся тем,

– когда процессоры обмениваются информацией о своей текущей загрузке,

– где принимается решение о перераспределении загрузки,

– когда производится это перераспределение,

– в соответствии с каким алгоритмом производится перераспределение.

Процессоры могут обмениваться информацией о своей текущей загрузке на каждой итерации решения исходной задачи; периодически; когда загрузка процессора станет меньше некоторой допустимой; по появлении свободного процессора и т. д.

Решение о перераспределении загрузки может приниматься централизованно (на основе глобальной информации о загрузке) и децентрализованно (на основе только локальной информации).

Перераспределение загрузки может происходить синхронно и асинхронно.

Может производиться с использованием детерминированных и стохастических алгоритмов; алгоритмов, в которых перераспределение загрузки производится по инициативе получателя и по инициативе отправителя и т. д.

Рассмотрим в качестве примера два динамических метода балансировки загрузки – централизованный детерминированный метод экспоненциальной балансировки и децентрализованный стохастический диффузный метод. Речь во всех случаях идет о балансировке загрузки *MIMD*-вычислительной системы.

Метод динамической экспоненциальной балансировки

Обозначим рассматриваемое множество операторов $A^0 = A$. Положим, что число операторов во много раз превышает число процессоров рассматриваемой *MIMD*-параллельной ЭВМ, то есть $\left|A^0\right| >> \left|P\right|$. Пусть A^k – совокупность операторов алгоритма, которые остались необработанными на k-й итерации рекурсивного процесса экспоненциальной декомпозиции множества A^0; $k = 0, 1, 2, ...$

Экспоненциальная декомпозиция предполагает выделение из множества A^k подмножества A^{k+1} по следующей схеме.

1) Если число оставшихся необработанными операторов не превышает число процессоров в системе, то есть $\left|A^k\right| \le \left|P\right|$, то включаем во множество A^{k+1} все $\left|A^k\right|$ необработанных операторов.

2) Если $\left|A^k\right| > \left|P\right|$, то из множества A^k выделяем подмножество A^{k+1} мощностью $\left|A^{k+1}\right| = \left\lceil \rho \left|A\right|^k \right\rceil$, где $\rho \in (0; 1]$ – коэффициент декомпозиции.

3) Множество A^{k+1} разбиваем на $\left|P\right|$ субмножеств A_i^{k+1}, содержащих по $\left|A_i^{k+1}\right| = \left\lceil \dfrac{\left|A^{k+1}\right|}{\left|P\right|} \right\rceil$ операторов; $i \in [1 : \left|P\right|]$.

Во введенных обозначениях и допущениях схема k-й итерации динамической экспоненциальной балансировки загрузки имеет следующий вид (опущены операции передачи процессорами p_j, $j \in [1 : \left|P\right|]$ результатов своей работы *host*-процессору, а также операции приема *host*-процессором этих результатов).

1) *Host*-процессор выполняет указанные ниже действия.

1.1) Выделяет из множества A^k подмножество A^{k+1}.

1.2) Разбивает множество A^{k+1} на $\left|P\right|$ субмножеств A_i^{k+1}.

1.3) Если множество A^{k+1} не исчерпано, то передает на обработку свободному процессору p_j программные коды операторов первого из необработанных субмножеств A_i^{k+1}.

1.4) Если множество A^{k+1} исчерпано, а множество A^k не исчерпано, то выполняет присваивание $k = k + 1$ и переходит к шагу 1.

1.5) Если множество A^k исчерпано, то *host*-процессор посылает процессорам p_j, $j \in [1 : \left|P\right|]$ сообщение о завершении вычислений и сам завершает вычисления.

2) Процессор p_j, $j \in [1:|P|]$ выполняет следующие действия.

2.1) Принимает от *host*-процессора программные коды операторов субмножества A_i^{k+1}.

2.2) Исполняет полученные программные коды.

Легко видеть, что рассмотренная схема приводит к тому, что с ростом номера итерации k число операторов, распределяемых каждому из процессоров, экспоненциально уменьшается (отсюда и название метода). Оптимальное значение величины ρ соответствует компромиссу между двумя противоречивыми тенденциями. С одной стороны, увеличение этого коэффициента уменьшает общее число обменов и тем самым уменьшает коммуникационные расходы, обусловленные латентностью коммуникационной сети. С другой стороны, в тех же условиях возможно ухудшение равномерности загрузки процессоров.

Метод диффузной балансировки

Сделаем следующие допущения. Операторы a_i, $i \in [1:|A|]$ алгоритма выполняются итерационно и асинхронно. Вычислительная сложность $v_i^A = v_i^A(t)$ оператора a_i в процессе итераций меняется не слишком быстро, так что на основе величины $v_i^A(t)$ можно с достаточной точностью оценить величину $v_i^A(t+1)$. Здесь $t = 1, 2, \ldots$ – номер итерации. Если процессору p_j назначены на исполнение операторы $a_{j_k}, j_k \in I_j$, то загрузка этого процессора равна $\mu_j = \mu_j^w + \mu_j^c$, где вычислительную загрузку μ_j^w и коммуникационную загрузку μ_j^c определяют выражения вида (9.2), (9.3) соответственно.

Схема метода диффузной балансировки загрузки, *инициируемой отправителем*, имеет следующий вид.

1) После завершения текущей итерации процессор p_j, $j \in [1:|P|]$ вычисляет свою текущую загрузку μ_j. Если эта величина превышает заданную загрузку $\tilde{\mu}$ (свободный параметр метода), то процессор p_j случайным образом выбирает процессор p_k и посылает ему запрос; $k \in [1:|P|]$, $k \neq j$.

2) Процессор p_k, получив запрос от процессора p_j, проверяет свою загрузку μ_k. Если эта величина меньше заданной загрузки $\tilde{\mu}$ (свободный параметр метода), то процессор p_k посылает процессору p_j согласие на прием части его загрузки.

3) Получив указанное сообщение, процессор p_j передает процессору p_k, а тот принимает программный код одного из неисполняемых в настоящее время его операторов.

4) Если же загрузка процессора p_k превышает величину $\tilde{\mu}$, то этот процессор посылает процессору p_j отказ в приеме части его загрузки.

5) Получив такое сообщение, процессор p_j выбирает по рассмотренной схеме другой процессор p_m. И так далее до достижения фиксированного числа попыток.

В случае, когда все или большая часть процессоров параллельной ЭВМ перегружены, они будут в соответствии с рассмотренным алгоритмом постоянно посылать в коммуникационную сеть запросы. При этом загрузку удастся уменьшить немногим из них, но сеть может оказаться сильно перегруженной.

Схема метода диффузной балансировки загрузки, *инициируемой получателем*, имеет близкий вид.

1) После завершения текущей итерации процессор p_j вычисляет свою текущую загрузку μ_j, и если $\mu_j < \tilde{\mu}$, то случайным образом выбирает процессор p_k и посылает ему запрос.

2) Процессор p_k, получив указанный запрос, проверяет свою загрузку μ_k, и если $\mu_k > \hat{\mu}$, посылает процессору p_j согласие на передачу части своей загрузки и передает ему программный код одного из операторов алгоритма.

3) Если $\mu_k \le \hat{\mu}$, то процессор p_k посылает процессору p_j отказ в передаче ему части своей загрузки.

4) Получив такое сообщение, процессор p_j выбирает по рассмотренной схеме другой процессор p_m и т. д.

Преимущество данного метода балансировки состоит в том, что он не оказывает дополнительной нагрузки на коммуникационную сеть в то время, когда параллельная ЭВМ перегружена. Очевидно, что в случае, когда почти все процессоры ЭВМ простаивают, данный метод создает существенную нагрузку на коммуникационную сеть. Однако, поскольку система недогружена, эта нагрузка не слишком обременительна.

Очевидным вариантом рассмотренных методов является посылка сообщений с предложением передать или принять часть загрузки в первую очередь ближайшим соседним процессорам.

9.2.5. Задача согласования алгоритма с архитектурой параллельной ЭВМ

Как отмечалось выше, одной из важнейших особенностей популяционных алгоритмов оптимизации является наличие большого числа свободных параметров, поскольку, во-первых, от их значений может сильно зависеть эффективность алгоритма, а во-вторых, оптимальные значения этих параметров различны для различных целевых и ограничивающих функций и могут меняться в процессе итераций. Формальные правила выбора значений свободных параметров обычно отсутствуют. Задача определения оптимальных значений этих параметров представляет собой метазадачу оптимизации (гл. 7).

Параллельные популяционные алгоритмы оптимизации имеют, как правило, еще большее число свободных параметров. Наряду с задачей метаоптимизации при реализации параллельного популяционного алгоритма возникает задача согласования этого алгоритма с архитектурой используемой параллельной ЭВМ. Задача состоит в поиске таких значений свободных параметров алгоритма, которые обеспечивают его максимальную эффективность для данной параллельной ЭВМ.

Если множество допустимых значений свободных параметров алгоритма является счетным и его мощность не слишком велика, то отыскать *согласованный* алгоритм, очевидно, можно следующим путем: перебрать все алгоритмы, порожденные указанным множеством свободных параметров; выполнить оптимальное отображение каждого из полученных алгоритмов на архитектуру ЭВМ; определить на этой основе соответствующие значения критерия качества согласования; выбрать алгоритм, обеспечивающий лучшее значение критерия.

Программная реализация идеи согласования приводит к программе, самосогласующейся с архитектурой параллельной ЭВМ, то есть к *самосогласующейся* программе. Иногда такую программу называют *гибкой* программой.

Аналогично метаоптимизации, можно выделить задачи параметрического, структурного и структурно-параметрического согласования. Ограничимся рассмотрением задачи *параметрического согласования.*

Пусть алгоритм решения рассматриваемой задачи поисковой оптимизации есть $A(B) = \{a_i(B), i \in [1:|A|]\}$, где $B = (b_1, b_2, ..., b_{|B|}) \in D_B$ вектор стратегий алгоритма, D_B – множество допустимых стратегий. Число $|A|$ операторов алгоритма $A(B)$ в общем случае зависит от стратегии B, так что $|A| = |A(B)|$.

Говоря более строго, стратегией оператора a_i является не обязательно весь вектор B, но лишь его некоторый подвектор B_i, то есть $a_i = a_i(B_i)$, где

$$\bigcup_{i=1}^{|A|} B_i = B.$$

Для простоты записи опускаем этот нюанс в обозначениях не только алгоритма и его операторов, но и в обозначениях других объектов (ребер графа алгоритма, вычислительной сложности его операторов и т. д.).

Во введенных обозначениях и соглашениях матрица Z, отображающая алгоритм $A(B)$ на *MIMD*-параллельную ЭВМ, приобретает вид

$$Z(B) = (z_{i,j}(B), i \in [1:|A(B)|], j \in [1:|P|]) \in D_Z.$$

Критерий оптимальности согласования, обычно – время решения исходной задачи оптимизации, обозначаем $\mu(B, Z)$. Положим, что требуется минимизировать значения этой функции.

Таким образом, задача параметрического согласования алгоритма $A(B)$ с архитектурой параллельной ЭВМ, определяемой графом $\langle P, C^P \rangle$, ставится следующим образом. Найти стратегию $B^* \in D_B$, а тем самым и алгоритм $A^* = A(B^*)$, которые обеспечивают минимальное значение целевой функции $\mu(B, Z)$:

$$\min_{B \in D_B} \min_{Z \in D_Z} \mu(B, Z) = \mu(B^*, Z^*). \qquad (9.6)$$

Внешняя задача в выражении (9.6) представляет собой задачу глобальной условной оптимизации, которая принципиально может быть решена с помощью любого из рассмотренных популяционных алгоритмов. На каждой итерации этого алгоритма должна быть решена задача отыскания оптимальной отображающей матрицы Z^*, то есть задача оптимального отображения.

9.3. МЕТОДЫ РАСПАРАЛЛЕЛИВАНИЯ ПОПУЛЯЦИОННЫХ АЛГОРИТМОВ ОПТИМИЗАЦИИ

Идеи, лежащие в основе популяционных алгоритмов оптимизации, дают основания надеяться на существование эффективных параллельных вариантов этих алгоритмов. Действительно, известно значительное число таких хорошо масштабируемых алгоритмов, которые, однако, в своей массе отличаются от соответствующих последовательных прототипов.

В литературе представлено несколько способов классификации параллельных популяционных алгоритмов. Наиболее известна классификация, в соответствии с которой выделяют следующие классы таких алгоритмов: алгоритмы, использующие глобальную модель параллелизма; миграционные алгоритмы, основанные на островной модели параллелизма; алгоритмы, в основе которых лежит диффузная модель параллелизма; алгоритмы, использующие другие модели параллелизма.

9.3.1. Глобальная модель параллелизма

Параллельные алгоритмы, построенные на основе глобальной модели параллелизма (*global model*), представляют собой параллельные аналоги соответствующих последовательных алгоритмов. Алгоритмы используют параллелизм по данным и ориентированы на организацию параллельных вычислений по типу «хозяин-раб» (*master-slave*) или, что то же самое, по типу «клиент-сервер» (*client-server*). В некоторых публикациях алгоритмы данного класса называют алгоритмами *параллельных агентов*.

Мастер-процесс в этом случае выполняется на *host*-процессоре параллельной ЭВМ и реализует собственно популяционный алгоритм. Каждый из подчиненных (рабочих) процессов назначают на выполнение одному из процессоров ЭВМ. Рабочий-процесс производит вычисления значений фитнес-функции, а также фазовых координат соответствующего агента (его положения в пространстве поиска, а также, быть может, скорости и ускорения). Функции рабочих-процессов могут ограничиваться только вычислением значений фитнес-функции. Для простоты изложения далее исходим именно из такой ситуации. Рабочие-процессы после каждой итерации отсылают мастер-процессу соответствующие значения фитнес-функции, который на основе этих данных вычисляет новые фазовые координаты агентов и посылает их рабочим-процессам.

Параллельные вычисления в алгоритмах рассматриваемого класса могут быть как синхронными, так и асинхронными.

Глобальная синхронная модель параллелизма (*global synchronous model*) предполагает, что текущая итерация завершается только после того, как мастер-процессором получены значения фитнес-функции от всех рабочих-процессоров. Другими словами, данная модель параллелизма предполагает использование так называемой глобальной барьерной синхронизации. Вычисление мастер-процессором глобальных характеристик популяции (например, отыскание глобально лучшего агента) производится в этом случае только после получения этим процессором указанной информации обо всех агентах популяции. Легко видеть, что глобальной синхронной модели параллелизма соответствует статическая балансировка загрузки рабочих-процессоров.

Условиями высокой производительности алгоритма, использующего данную модель параллелизма, являются гомогенность вычислительной системы и одинаковое время вычисления значений фитнес-функции в любой допустимой точке пространства параметров. Обеспечить выполнение всех этих условий обычно не удается, поскольку часто приходится использовать гетерогенные вычислительные системы, и в реальных оптимизационных задачах

вычислительная сложность фитнес-функции может зависеть от значений компонент вектора варьируемых параметров.

Глобальная синхронная модель параллелизма может быть эффективно реализована на *MIMD*-системах с общей и распределенной памятью, а также на *GPU*.

Глобальная асинхронная модель параллелизма (*global asynchronous model*) исходит из того, что мастер-процессор получает данные от рабочих-процессоров не после глобальной синхронизации, а в любой момент времени по мере готовности этих данных. На основе полученной информации мастер-процессор обновляет скорости и координаты соответствующих агентов популяции и немедленно возвращает их свободным рабочим-процессорам для продолжения итераций.

В алгоритме, реализующем данную модель параллелизма, легко обеспечить динамическую балансировку загрузки процессоров. Например, номера необработанных агентов можно поместить в очередь *FIFO*. Мастер-процесс обновляет фазовые координаты первого агента из этой очереди и посылает их первому свободному процессору.

Глобальная асинхронная модель параллелизма ориентирована на *MIMD*-вычислительные системы с общей или с распределенной памятью. Данную модель невозможно эффективно реализовать в рамках парадигмы *GPU*.

Достоинство параллельных популяционных алгоритмов, основанных на глобальной модели параллелизма, состоит в возможности получения и использования глобальной информации о популяции, например в алгоритме роя частиц – информации о глобально лучшей частице. Результат оптимизации, полученный при использовании глобальной синхронной модели параллелизма, получается точно таким же, как при аналогичных последовательных вычислениях.

Недостатком алгоритмов данного класса являются высокие накладные расходы на коммуникации (при большом диаметре коммуникационной сети).

Известны работы, посвященные сравнительному исследованию эффективности синхронной и асинхронной моделей параллелизма. Так, в одной из работ Коха (*B. Koh*) с соавторами исследовано влияние дисперсии вычислительной сложности фитнес-функции на эффективность синхронного и асинхронного алгоритмов. Вычислительную сложность фитнес-функции полагали случайной величиной с заданным законом распределения и моделировали с помощью соответствующих временных задержек.

Исследование показало, что при увеличении дисперсии вычислительной сложности фитнес-функции эффективность синхронной модели линейно падает. Асинхронная модель на малом числе процессоров (менее 15) показывает в этих условиях низкую эффективность, но при дальнейшем увеличении числа процессоров эффективность модели асимптотически возрастает до 93 процентов. Вариации вычислительной сложности фитнес-функции практически не оказывают влияния на эффективность асинхронной модели. Заметим, что указанная зависимость эффективности от числа процессоров не является типичной для параллельных вычислений и объясняется малой долей коммуникаций в процессе вычислений, а также распределением их по времени (в отличие от единовременных коммуникаций в синхронной модели).

9.3.2. Островная модель параллелизма

Параллельные популяционные алгоритмы, построенные на основе островной модели (*island model*) параллелизма, в некоторых публикациях называют *миграционными* алгоритмами (*migration algorithms*) или алгоритмами *коммутирующей мультипопуляции*. Суть алгоритмов этого класса в следующем. Создают мультипопуляцию $\boldsymbol{S} = \bigcup_{i=1}^{|P|} S_i$, состоящую из числа субпопуляций (островов) S_i, равного числу используемых процессоров $|P|$ параллельной ЭВМ, так что число агентов мультипопуляции равно $|\boldsymbol{S}| = \sum_{i=1}^{|P|} |S_i|$. Каждый остров обрабатывает свой процессор системы. Обмен данными между островами выполняют после каждых $t^{mig} < \hat{t}$ независимых итераций в соответствии с используемой топологией соседства островов. Период из t^{mig} итераций популяционного алгоритма, в течение которых субпопуляции развиваются независимо, называют *сезоном* (*season*).

Островную модель параллелизма иллюстрирует рис. 9.11, на котором представлены пять субпопуляций $S_1 - S_5$, граф связей между которыми представляет собой кольцо.

Островная модель использует статическую балансировку загрузки параллельной ЭВМ и обеспечивает, как правило, высокую производительность при выполнении следующих условий: вычислительная система является гомогенной; размеры субпопуляций одинаковы и достаточно велики; условием завершения итераций каждой из субпопуляций $S_i, i \in [1:|\boldsymbol{S}|]$ является достижение одного и того же числа итераций $\hat{t}$.

Поскольку популяционные алгоритмы являются стохастическими, при разных запусках каждого из таких алгоритмов популяция может сходиться к разным локальным решениям. Островная модель позволяет, по сути, запустить алгоритм сразу несколько раз и совместить «достижения» разных островов для получения наилучшего решения. С другой стороны, островной популяционный алгоритм является ярким примером многопопуляционного алгоритма. По общим правилам построения таких алгоритмов может быть целесообразным использование в разных популяциях различных миграционных операторов и значений их свободных параметров так, чтобы в одних популяциях получить высокие интенсификационные свойства алгоритма, а в других – его такие же диверсификационные свойства.

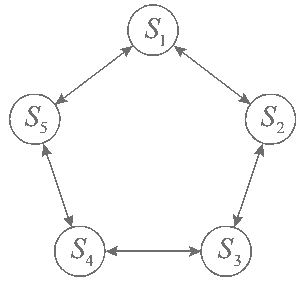


Рис. 9.11. К островной модели параллелизма: топология связей островов – кольцо; число субпопуляций в мультипопуляции $|\boldsymbol{S}| = 5$

В литературе имеются следующие рекомендации относительно предпочтительности использования модели «хозяин-раб» и островной модели. Если пропускная способность коммуникационной сети ЭВМ является невысокой и мала вычислительная сложность фитнес-функции, то лучше выбирать островную модель. Также островную модель лучше использовать в случае популяции очень большого размера. В остальных случаях более предпочтительной является модель «хозяин-раб».

Известно большое число вариантов островной модели, которые определяются топологией связей островов, длительностью сезона t^{mig}, стратегией миграции, режимом обмена агентами.

Топология связей островов. При не слишком большом числе субпопуляций в островной модели чаще других используют топологию связей в виде полносвязного графа, когда агенты любого острова могут мигрировать в любой другой остров. Часто используют также топологию связей типа кольцо. В этом случае агенты данного острова могут мигрировать лишь между двумя соседними островами. Могут использоваться и другие топологии соседства (см. 3.1).

Длительность сезона. При больших значениях величины t^{mig} алгоритм, реализующий островную модель параллелизма, имеет малые коммуникационные расходы, но недостаточно эффективно использует информацию о других островах. Напротив, при малых t^{mig} такая информация используется эффективно, но имеют место большие коммуникационные расходы. Варьируя t^{mig}, можно обеспечить высокую эффективность островного алгоритма на широком классе параллельных ЭВМ.

Известен вариант островной модели, в которой каждый из островов S_i имеет, вообще говоря, свою и не постоянную, но динамическую длительность сезона $t_i^{mig} = t_i^{mig}(t)$. Правила изменения величин t_i^{mig} могут быть неадаптивными и адаптивными. В первом случае величина t_i^{mig} представляет собой некоторую фиксированную функцию номера итерации t. Во втором случае длительность сезона определяется текущим состоянием острова, например размером соответствующей части области поиска.

Стратегия миграции. В простейшем случае для обмена используют одного или более случайно выбранных агентов. Более развитый подход предполагает использование пула некоторого числа наиболее приспособленных агентов, отобранных из всех островов методом пропорциональной селекции (см. 2.5). Агентов, подлежащих миграции, выбирают из этого пула случайным образом. В данном подходе нужен специальный механизм для предотвращения коллизий, когда одна или несколько субпопуляций получают дубликаты своих агентов.

Наиболее известны две стратегии миграции – стратегия перемещением и стратегия репликации.

Миграция *перемещением* (*moving*) агентов заключается в замене лучшего агента данного острова наихудшим агентом другого острова. Миграция перемещением имеет целью удалить наиболее успешного агента из субпопуляции и тем самым улучшить диверсификационные свойства алгоритма, исключив тенденцию к его преждевременной сходимости. Естественно, при этом неизбежно уменьшается скорость сходимости алгоритма, то есть ухудшаются его интенсификационные свойства.

В миграции *репликацией* (*replication*) сначала в каждом из островов S_i осуществляют поиск лучшего s_{i_b} и худшего s_{i_w} агентов. Затем из всех особей s_{i_b} выбирают наилучшего агента мультипопуляции и осуществляют репликацию (копирование) этого агента во все остальные острова S_j на место агентов s_{j_w}; $i, j \in [1:|S|]$. Миграция репликацией преследует цель улучшить интенсификационные свойства алгоритма, то есть обеспечить его более высокую скорость сходимости.

Режим обмена агентами. В зависимости от режима обмена агентами различают синхронный и асинхронный варианты островной модели.

Синхронная островная модель параллелизма предполагает, что все острова ожидают завершения обработки полученных ими данных и только затем выполняют их новую рассылку.

Лучше использовать ресурсы сети позволяет *асинхронная* островная модель параллелизма. В этом случае каждый из островов отсылает свои данные по мере их готовности, и эти данные находятся в «почтовом ящике» острова-получателя до тех пор, пока соответствующая субпопуляция не будет готова их обработать. Программная реализация последнего варианта требует обработки ситуации переполнения «почтовых ящиков».

9.3.3. Диффузная модель параллелизма

Методы этого класса можно считать частным случаем методов, основанных на островной модели параллелизма. Каждая из субпопуляций в этом случае включает в себя только одного агента ($|S_i| = 1, i \in [1:|\boldsymbol{S}|]$), так что число островов равно числу агентов и одновременно числу используемых процессоров вычислительной системы, то есть $|\boldsymbol{S}| = |S| = |P|$.

Основная идея диффузных методов состоит в том, что лучших агентов определяют параллельно и только среди соседних агентов (в смысле используемой топологии их соседства). Для этого в каждом сезоне миграции процессор, обрабатывающий агента s_i, получает от всех тех процессоров, на которых обрабатываются агенты, соседние с данным, их фазовые координаты и соответствующие значения фитнес-функции. Такие коммуникации выполняются параллельно всеми $|P|$ процессорами.

Коммуникационные расходы в диффузной модели напрямую определяются используемой топологией соседства агентов и являются высокими в случае, если соответствующий граф имеет высокую связность. Методы данного класса обеспечивают высокую производительность только при невысокой связности указанного графа и/или высокой вычислительной сложности фитнес-функции $\varphi(X)$.

В целом методы, основанные на диффузной модели параллелизма, ориентированы на *MIMD*-параллельные вычислительные системы с общей памятью или сильносвязные системы с распределенной памятью (например, гиперкуб или двумерный тор), имеющие быстрые коммуникационные сети.

Вариантом диффузной модели параллелизма можно считать модель, используемую *клеточным* генетическим алгоритмом (*cellular genetic algorithm*). Топология связей агентов между собой в этом случае фиксирована и представляет собой двумерный тор. Каждый агент обменивается данными только с четырьмя своими соседями (сверху, снизу, слева, справа).

9.3.4. Другие модели параллелизма

Кроме рассмотренных, выделяют модель некоммутирующей мультипопуляции, пространственно вложенную модель, модель расщепления, модель коэволюции, комбинированные модели.

Модель *некоммутирующей мультипопуляции* можно считать частным случаем островной модели, когда непосредственный обмен данными между субпопуляциями отсутствует. Суть алгоритмов, использующих данную модель, передает следующая последовательность их основных шагов.

1) Создаем мультипопуляцию, содержащую некоторое число субпопуляций с разными, вообще говоря, начальными состояниями агентов и значениями свободных параметров.

2) Обеспечиваем каждой из созданных субпопуляций независимую эволюцию на своем процессоре используемой параллельной ЭВМ.

3) Лучшее решение, найденное каждым из процессоров, передаем на *host*-процессор системы, который выбирает из них решение задачи.

Достоинством алгоритмов данного класса является минимум коммуникационных расходов, а недостатком – слабое использование потенциала мультипопуляции. Можно считать, что алгоритмы некоммутирующей мультипопуляции представляют собой параллельный вариант алгоритма мультистарта.

Пространственно вложенную модель параллелизма (*spatially embedded*) целесообразно использовать для распараллеливания, например популяционного алгоритма типа «хищник-жертва», когда агенты популяции связаны с узлами некоторой виртуальной сетки (п. 8.3.2). Данная модель параллелизма предполагает декомпозицию этой сетки на одинаковые фрагменты (в случае гомогенной параллельной ЭВМ) и их обработку различными процессорами. Вследствие того, что в процессе функционирования алгоритма агенты перемещаются по узлам указанной сетки, модель предполагает миграцию агентов между процессорами.

Модель параллелизма *расщеплением* используют при высокой размерности вектора варьируемых параметров. Идея алгоритмов этого класса состоит в том, что каждому из процессоров параллельной ЭВМ назначают для обработки равное число компонент этого вектора (если его размерность кратна числу процессоров и вычислительная система гомогенна). Различные компоненты вектора могут, вообще говоря, обрабатывать различные популяционные алгоритмы, как это предполагает метод декомпозиции пространства поиска (параграф 6.1).

Коэволюционную модель параллелизма можно считать частным случаем или развитием островной модели, ориентированным на распараллеливание коалгоритмов. Модель коэволюции предполагает, что каждая из коэволюционирующих субпопуляций выполняется на своем процессоре. Основной проблемой данной модели является проблема балансировки загрузки используемой параллельной ЭВМ, поскольку размеры субпопуляций в процессе коэволюции меняются и, возможно, весьма существенно. Для повышения равномерности загрузки целесообразно назначать каждому из процессоров не одну, а несколько субпопуляций. Повысить эффективность такой модели параллелизма можно путем динамической балансировки загрузки, когда по тому или иному правилу в процессе вычислений субпопуляции перераспределяются между процессорами.

Примером широко используемой *комбинированной* модели параллелизма может служить комбинация островной модели и модели «хозяин-раб», когда каждая из субпопуляций имеет свое множество «рабов», оценивающих при-

способленность агентов этой субпопуляции. При этом «раб» может обрабатывать как одного агента, так и их некоторую совокупность.

Примером комбинированной модели может служить также объединение диффузной модели и модели расщепления. В этом случае каждому агенту назначают равное число процессоров (группу). В пределах каждой группы выделяют один процессор, отвечающий за моделирование эволюции данного агента, а остальным процессорам группы поручают обработку соответствующих частей вектора варьируемых параметров.

9.4. ПРИМЕРЫ ПАРАЛЛЕЛЬНОГО РЕШЕНИЯ ЗАДАЧ ОПТИМИЗАЦИИ

В первом пункте рассматриваем параллельный алгоритм роя частиц *GIPSO*, предназначенный для решения задачи глобальной безусловной оптимизации на *GPU* с использованием технологии программирования *CUDA*. Второй пункт посвящен параллельному построению на *GPU* дискретной аппроксимации множества Парето в задаче многоцелевой оптимизации с помощью алгоритма *MOPSO*.

9.4.1. Параллельный алгоритм роя частиц *GIPSO*

Алгоритм роя частиц *GIPSO* (от *GPU Island PSO*) ориентирован на использование *GPU* и технологии программирования *CUDA*. Основу алгоритма составляют модифицированный канонический алгоритм роя частиц и две модели параллелизма – островная модель и модель расщепления.

Полагаем, что область поиска представляет собой гиперпараллелепипед Π, размеры которого определяет $(|X|\times 1)$-вектор $X_{\max}$ с компонентами $x_i^+ - x_i^-, i\in[1:|X|]$. Модификация алгоритма *PSO* заключается в использовании ограничения сверху скоростей частиц величиной $V_{\max} = \gamma X_{\max}$, где $\gamma \in [0,01;1]$ – свободный параметр.

Отображения алгоритма GIPSO на архитектуру GPU

Рассмотрим отображение алгоритма *GIPSO* на архитектуру *GPU* в случае использования островной модели параллелизма. Для модели расщепления отображение выполняется по аналогичной схеме.

Положим, что число частиц в каждом из суброев $S_i, i\in[1:|\boldsymbol{S}|]$ одинаково и равно $|S|$. Поскольку частицы внутри суброев достаточно интенсивно коммутируют (после каждой итерации выбирается лучшая частица), частицы каждого из суброев назначаем на обработку одному вычислительному блоку. Каждой из частиц ставим в соответствие *CUDA*-поток. Для уменьшения влияния латентности глобальной памяти целесообразно использовать число суброев $|\boldsymbol{S}|$, которое значительно превосходит число мультипроцессоров *GPU*.

В *CUDA* нет средств явного управления объемом памяти, используемой потоком. Если локальные данные потока не помещаются в регистровую память, то *CUDA*-компилятор (*NVCC*) использует для этих данных разделяемую память мультипроцессора, которая, как отмечалось выше, является очень

медленной по сравнению с регистровой памятью. Поэтому при отображении алгоритма *GIPSO* на архитектуру *GPU* важен предварительный анализ требуемых для реализации алгоритма объемов памяти.

Пусть потоку для его выполнения требуется r_{st} регистров. Тогда для вычислительного блока, состоящего из $|S|$ потоков, число требуемых регистров

$$r_b = \left\lfloor r_{st} \left\lfloor |S|, 32 \right\rfloor, \frac{g_{reg}}{32} \right\rfloor \le g_{reg}, \tag{9.7}$$

где g_{reg} – емкость памяти каждого из потоковых процессоров. Легко видеть, что объем общей памяти в байтах, используемой алгоритмом *GIPSO* для одного суброя, определяет выражение

$$q_{shm} = 4\,(3\,|X|\,|S| + |S| + |X| + 1) \le g_{shm},$$

где g_{shm} – емкость разделяемой памяти.

В соответствии со спецификацией *Compute Capability 1.2*, например, $g_{reg} = g_{shm}$ =16 Кб. Отсюда для указанной спецификации получаем следующее ограничение на число частиц в суброе:

$$|S| \le \frac{\frac{g_{shm}}{4} - (|X| + 1)}{3|X| + 1}. \tag{9.8}$$

Программная реализация алгоритма GIPSO

Основу программы *GIPSO* составляют два типа вычислительных блоков, называемых вычислительными ядрами. Ядро 1 реализует движение частиц соответствующего суброя в течение одного сезона. Ядро 2 осуществляет миграцию частиц между суброями. Ядра 1 образуют $|\boldsymbol{S}|$ вычислительных блоков, а ядра 2 – один блок, состоящий из $|\boldsymbol{S}|$ потоков.

Программа включает в себя две реализации ядра 1. В реализации *PsoV1* поток соответствует одной частице, так что общее число потоков равно $|\boldsymbol{S}|\,|S|$, а число потоков в блоке – $|S|$. В реализации *PsoV2* поток отвечает за движение одной частицы по одной размерности пространства поиска, так что общее число потоков равно $|\boldsymbol{S}|\,|S|\,|X|$, а число потоков в одном блоке – $|S|\,|X|$.

Ядро 2 в программе *GIPSO* реализует стратегии миграции как репликацией, так и перемещением. Ядро осуществляет распараллеливание операции миграции путем переноса данных непосредственно в глобальную память *GPU*, не копируя их в память *host*-процессора.

Схема программы *GIPSO* имеет следующий вид.

1) Инициализируем на *host*-процессоре суброи – задаем начальные положения и скорости всех частиц популяции.

2) Копируем начальные положения и скорости частиц в глобальную память *GPU*.

3) Генерируем на *host*-процессоре два массива, содержащих по $|\boldsymbol{S}|\,|S|\,|X|\,\hat{t}$ случайных чисел, равномерно распределенных в интервалах $[0; b_C]$, $[0; b_S]$ соответственно (см. формулу (3.2)).

4) Копируем указанные массивы в глобальную память *GPU*.

5) Выполняем на *GPU* цикл оптимизации на протяжении первого сезона.

5.1) Выполняем на *GPU* ядра 1, реализующие движение частиц всех суброев в течение одного сезона.

5.2) Передаем *host*-процессору информацию о лучших и худших частицах каждого из суброев путем копирования соответствующих данных из глобальной памяти *GPU* в память этого процессора.

5.3) Строим на *host*-процессоре план миграции частиц.

5.4) Передаем соответствующие данные *GPU*.

5.5) Выполняем на *GPU* ядра 2, которые в соответствии с разработанным планом осуществляют перемещение частиц между суброями.

6) По схеме шага 5 выполняем на *GPU* цикл оптимизации на протяжении второго сезона и так далее до последнего сезона, который завершается передачей *host*-процессору информации о лучших частицах каждого из суброев.

7) Находим на *host*-процессоре положение $\tilde{X}^*$ наилучшей частицы среди лучших частиц всех суброев, а также соответствующее значение целевой функции $\tilde{f}^*$. Принимаем эти величины в качестве приближенного решения задачи.

Заметим, что необходимость предварительной генерации на *host*-процессоре случайных чисел (шаг 3) обусловлена тем обстоятельством, что генерация таких чисел на *GPU* затруднена.

Рассмотренная структура программы *GIPSO* показывает наличие значительного числа последовательных фрагментов, которые не могут быть распараллелены – шаги 1 – 4, 5.1–5.4, 7. По известному закону Амдала наличие таких фрагментов может значительно снизить максимально возможное ускорение алгоритма.

Известно, что для оптимизации отображения вычислительных алгоритмов на архитектуру *CUDA* следует соблюдать следующие требования:

– доступ потоков в блоке к глобальной памяти и памяти констант должен быть выровнен по адресам и осуществляться по определенным шаблонам;

– доступ различных потоков должен осуществляться к различным банкам разделяемой памяти;

– вместо локальной памяти следует использовать регистры;

– число ветвлений и барьерных синхронизаций в программе должно быть минимальным.

В программе *GIPSO* обеспечены указанные требования по доступу к состоянию роев в разделяемой памяти, а также по доступу к массивам случайных чисел в глобальной памяти. Однократные чтение и запись начального состояния роя в глобальной памяти при инициализации (шаг *2*) не выравниваются по адресам. Поскольку шаг 2 алгоритма выполняется однократно, данное обстоятельство незначительно снижает общую производительность алгоритма.

Исследование эффективности алгоритма GIPSO

Проиллюстрируем эффективность алгоритма *GIPSO* на примере многоэкстремальной функции Растригина (Приложение Б). В качестве области поиска рассматриваем гиперкуб $\Pi = (X \mid -20 \le x_j \le 20,\ j \in [1:|X|])$. Длительность сезона миграции t^{mig} принимаем равной 50 итерациям.

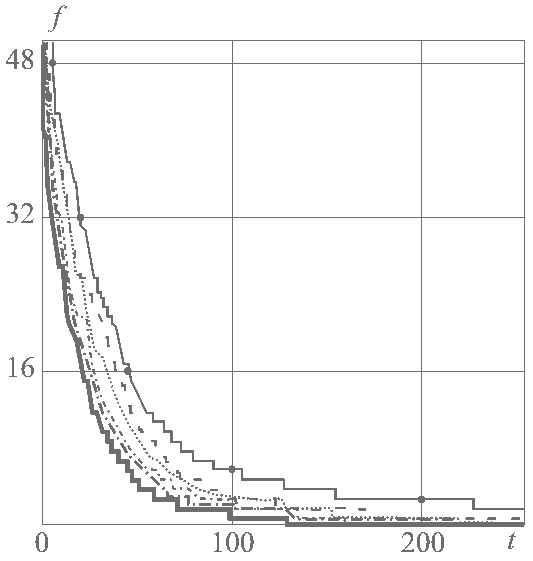


Рис. 9.12. Сходимость алгоритма *GIPSO*, функция Растригина: $|X| = 4$; $|S| = 4$; верхняя кривая соответствует $|\boldsymbol{S}| = 2$, нижняя – $|\boldsymbol{S}| = 64$

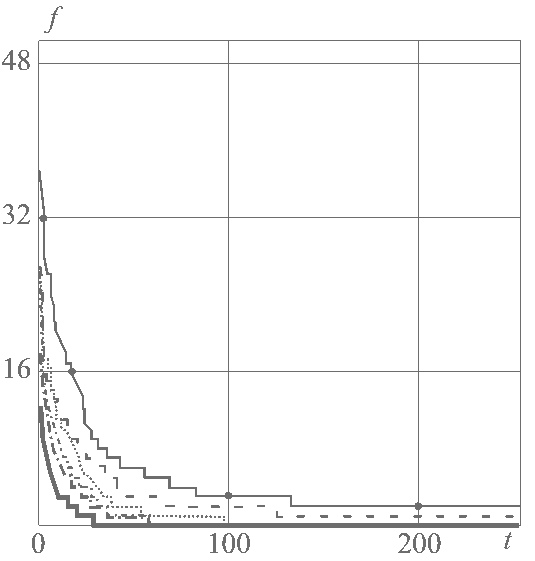


Рис. 9.13. Сходимость алгоритма *GIPSO*, функция Растригина: $|X| = 4$; $|\boldsymbol{S}| = 4$; верхняя кривая соответствует $|S_\Sigma = 8$, нижняя – $S_\Sigma| = 128$

Влияние свободных параметров. Рассмотрим прежде влияние на характеристики алгоритма следующих его свободных параметров: число суброев $|\boldsymbol{S}|$; общий размер популяции $|S_\Sigma| = |\boldsymbol{S}||S|$.

Сходимость алгоритма для случая использования стратегии миграции репликацией иллюстрирует рис. 9.12. Рисунок показывает высокую скорость сходимости алгоритма и примерно равномерное ее повышение по мере увеличения числа суброев $|\boldsymbol{S}|$.

Хорошо известно, что для канонического алгоритма *PSO* размер популяции оказывает слабое влияние на скорость сходимости, если этот размер превышает примерно 50. Общие рекомендации относительно канонического алгоритма сводятся к использованию не более 50 частиц в рое для сложных задач оптимизации и 20–50 частиц – для простых задач. По отношению к алгоритму *GIPSO* такие рекомендации, вообще говоря, неприменимы. Дело в том, что, во-первых, многороевая структура алгоритма и миграция частиц между суброями меняют смысл термина *размер популяции*. Во-вторых, ограничения (9.7), (9.8) на параметры алгоритма *GIPSO* не позволяют использовать большие суброи для решения задач оптимизации в пространстве высокой размерности.

Рис. 9.13 иллюстрирует влияние на сходимость алгоритма *GIPSO* размера популяции $|S_\Sigma|$ для четырехмерного пространства поиска $|X| = 4$. Число суброев $|\boldsymbol{S}|$ во всех случаях также равно 4, число частиц $|S|$ в суброях меняется от 2 до 32, так что $|S_\Sigma|$ варьируется от 8 до 128.

Рис. 9.13 показывает, что, как и для канонического алгоритма *PSO*, при малой размерности пространства поиска ($|X| = 4$) размер роя слабо влияет на скорость сходимости. Приемлемые с этой точки зрения результаты дают рои

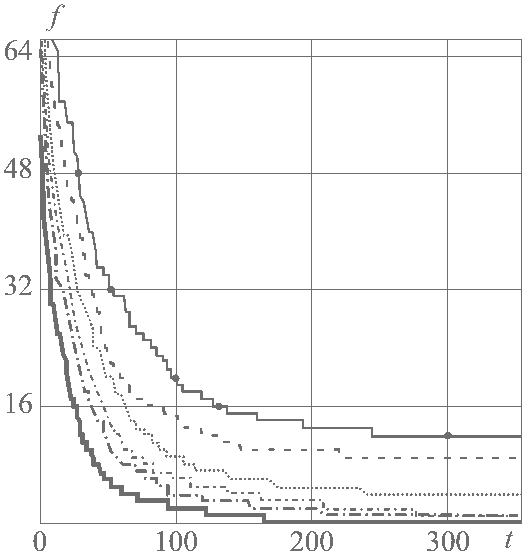


Рис. 9.14. Сходимость алгоритма *GIPSO*, функция Растригина: $|X| = 8$; $|\boldsymbol{S}| = 4$; верхняя кривая соответствует $|S_\Sigma| = 8$, нижняя – $|S_\Sigma| = 128$

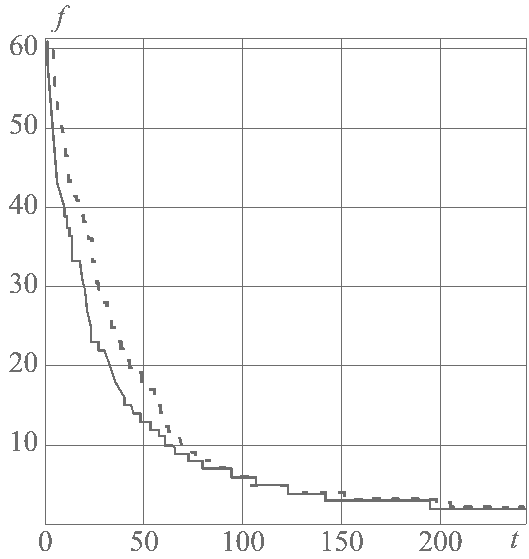


Рис. 9.15. Сходимость алгоритмов *GIPSO* и *PSO*, функция Растригина: $|X| = 4$; $|\boldsymbol{S}| = 4$; $|S| = |S_\Sigma| = 32$; сплошная линия – *GIPSO*; пунктирная – *PSO*

уже из 4–8 частиц. При этом во всех случаях имеет место локализация глобального минимума.

Аналогичные результаты для восьмимерного пространства поиска приведены на рис. 9.14. Из рисунка следует, что для пространства большей размерности ($|X| = 8$) размер роя начинает оказывать существенное влияние на скорость сходимости. Результаты аналогичного исследования, выполненного для функций «сфера», Розенброка и Экли (Приложение Б), показывают, что в данном случае размер роя существенно влияет также на вероятность локализации глобального минимума. Оказывается, что маленькие рои из 2–4 частиц не способны вывести процесс поиска из локального минимума для функций Экли и Растригина. Минимальный размер роя, способного отыскать глобальный минимум всех четырех указанных тестовых функций, составляет 8 частиц.

Результаты сравнительного исследования скорости сходимости алгоритма *GIPSO* и канонического алгоритма *PSO*, реализованного на *host*-процессоре системы, имеющем архитектуру x86, иллюстрирует рис. 9.15. Исследование выполнено при следующих значениях свободных параметров:

PSO – размер популяции $|S| = 32$;

GIPSO – общий размер популяции $|S_\Sigma| = 32$, число суброев $|\boldsymbol{S}| = 4$, длительность сезона $t^{mig} = 50$.

Рис. 9.15 показывает, что даже при одинаковом размере популяции алгоритм *GIPSO* имеет более высокую скорость сходимости по сравнению с алгоритмом *PSO*. Результаты аналогичного исследования, выполненного для функций «сфера», Розенброка и Экли, показывают, что преимущества алгоритма *GIPSO* увеличиваются с ростом сложности ландшафта минимизируемой функции.

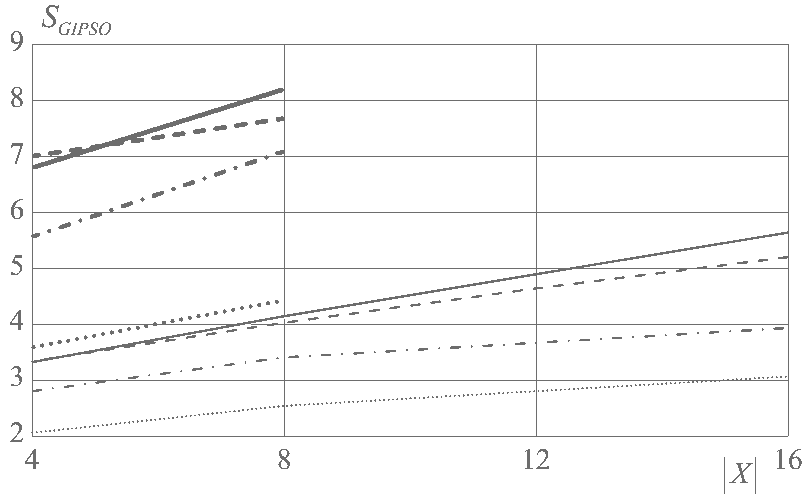


Рис. 9.16. Ускорение алгоритма *GIPSO*, ядро *PsoV2*; $\boldsymbol{S} = 4$; $|S_\Sigma| = 32$: —— функция Экли, – – – функция Растригина, ····· функция Розенброка, -·–·- сферическая функция; $|S_\Sigma| = 64$: ▬▬ функция Экли, ▬ ▬ ▬ функция Растригина, ····· функция Розенброка, -·-·- сферическая функция

Оценка производительности. Обозначим τ_{GIPSO}, τ_{PSO} времена решения задачи алгоритмами *GIPSO* и *PSO* соответственно. В качестве меры производительности алгоритма *GIPSO* используем ускорение

$$S_{GIPSO} = \frac{\tau_{PSO}}{\tau_{GIPSO}}. \tag{9.9}$$

Результаты исследования эффективности алгоритма *GIPSO* иллюстрирует рис. 9.16. Исследование выполнено для размеров популяции $|S_\Sigma| = 32,\ 64$ и при размерностях пространства поиска $|X| = 4, 8, 16$. В алгоритме *GIPSO* использовано ядро *PsoV2* при числе суброев $|\boldsymbol{S}| = 4$. Рисунок показывает, что в рассмотренном диапазоне изменения величины $|X|$ ускорение практически линейно растет с ростом этой величины. Увеличение размера популяции с $|S_\Sigma| = 32$ до $|S_\Sigma| = 64$ повышает ускорение почти в два раза. Эффект объясняется ростом загрузки мультипроцессоров с ростом числа частиц и/или размерности пространства поиска. Поскольку $|\boldsymbol{S}| = 4$, переход от $|S_\Sigma| = 32$ к $|S_\Sigma| = 64$ означает увеличение размеров суброев алгоритма *GIPSO* с 8 до 16.

Масштабируемость алгоритма. Введем в рассмотрение приведенное время решения задачи алгоритмом *GIPSO*

$$\overline{\tau}_{GIPSO}(|\boldsymbol{S}|) = \frac{\tau_{GIPSO}}{|\boldsymbol{S}|}$$

при фиксированном размере популяции $|S_\Sigma|$. Функциональную зависимость $\overline{\tau}_{GIPSO}(|\boldsymbol{S}|)$ иллюстрирует рис. 9.17. Исследование выполнено для функции Розенброка при $|S_\Sigma| = 16$ и при размерностях пространства поиска $|X| = 4, 8, 16$. Использовано ядро *PsoV2*. Рисунок показывает, что при числе суброев $|\boldsymbol{S}|$, превышающем 32, имеют место практически постоянные значения величины

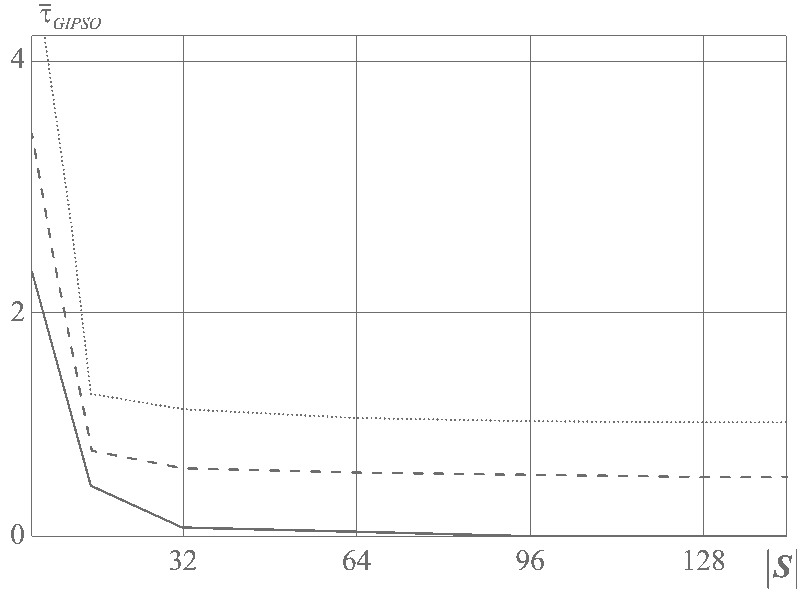


Рис. 9.17. К масштабируемости алгоритма *GIPSO*: ядро *PsoV2*; функция Розенброка, $|S_\Sigma| = 16$: —— $|X| = 4$; – – – $|X| = 8$; ····· $|X| = 16$

$\overline{\tau}_{GIPSO}(|S|)$ или, другими словами, почти линейный рост времени решения задачи с ростом величины $|S|$. Это обстоятельство позволяет при увеличении сложности целевой функции увеличивать число суброев для повышения эффективности алгоритма. Конечно, при этом не следует допускать преждевременной стагнации процесса поиска.

9.4.2. Параллельный алгоритм Парето-аппроксимации роем частиц

Рассматриваем алгоритм многоцелевой оптимизации роем частиц *MOPSO* (см. 8.6) и простейшую двухмерную двухцелевую задачу (пример 8.1). Используем схему распараллеливания «хозяин-раб», когда каждый из подчиненных процессоров вычисляет лишь соответствующие значения целевых функций. Вычисления выполнены на параллельной вычислительной системе, состоящей из *host*-ЭВМ (*Intel Pentium Dual E2160*, 1,8 ГГц, 2 Гб) и *GPU* (*NVIDIA GeForce 8500 GT*), содержащей 16 потоковых процессоров.

В практически значимых задачах Парето-аппроксимации основной объем вычислений связан с вычислениями значений целевых функций. Поэтому при параллельной аппроксимации множества Парето важным является исследование эффективности параллельных вычислений в зависимости от вычислительной сложности этих функций. Для организации такого исследования значения целевых функций вычислялись κ раз.

Исследование выполнено при $\kappa = 1, 10, 100, 1000$; числе частиц в популяции $|S| = 32, 64, 128, 256$; числе итераций $\hat{t} = 50$. Использован метод мультистарта, так что представленные ниже результаты усреднены по 30 запускам алгоритма. В качестве меры производительности алгоритма *MOPSO* использовано ускорение (9.9).

Таблица 9.1 **Эффективность распараллеливания алгоритма *MOPSO***

$\|S\|$	к	τ_{CPU}, с	τ_{GPU}, с	S_{MOPSO}	$\|S\|$	к	τ_{CPU}, с	τ_{GPU}, с	S_{MOPSO}
32	1	0,06	0,21	0,3	128	1	0,54	0,66	0,8
	10	0,30	0,27	1,1		10	2,50	0,80	3,1
	100	2,39	0,66	3,6		100	28,50	2,61	10,9
	1000	25,64	4,34	5,9		1000	219,03	15,41	14,2
64	1	0,15	0,36	0,4	256	1	3,36	1,23	2,7
	10	0,64	0,42	1,5		10	12,44	1,50	8,3
	100	6,56	1,14	5,8		100	81,44	4,45	18,3
	1000	69,83	7,64	9,1		1000	690,44	29,38	23,5

Некоторые результаты исследования представлены в табл. 9.1. Таблица показывает, что, как и следовало ожидать, при малой вычислительной сложности целевых функций распараллеливание неэффективно, поскольку в этом случае оказывается слишком большой доля вычислений, производимых на *host*-ЭВМ, а также доля коммуникационных расходов. При увеличении вычислительной сложности целевых функций и числа частиц в популяции ускорение вычислений повышается и достигает 23. Таблицу иллюстрирует рис. 9.18.

Представленные результаты показывают высокий потенциал использования *GPU* для приближенного построения множества Парето в задаче многоцелевой оптимизации.

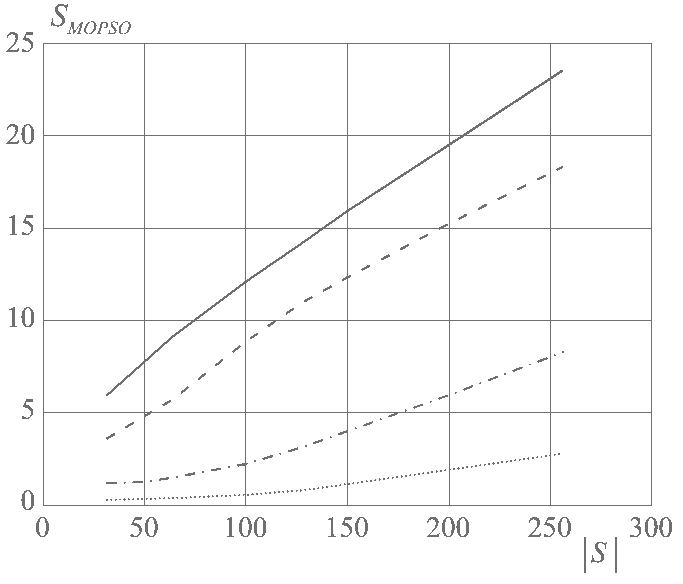


Рис. 9.18. К эффективности распараллеливания алгоритма *MOPSO*: —— к = 1000; – – – к = 100; -·–·- к = 10; ····· к = 1

Вопросы для самопроверки

1. Представьте основные классы параллельных вычислительных систем в соответствии с классификацией Флинна, а также с классификациями по типу строения оперативной памяти, по типу коммуникационной сети, по степени однородности.

2. Назовите основные архитектурные особенности следующих классов параллельных вычислительных систем: мультипроцессоры, мультикомпьютеры, графические процессорные устройства.

3. Представьте постановки задачи балансировки загрузки в терминах математического программирования и терминах теории графов.

4. Опишите метод статической балансировки загрузки, использующий алгоритм упаковки в контейнеры, а также иерархический графовый метод.

5. Приведите описания следующих методов динамической балансировки загрузки: экспоненциальная балансировка, диффузная балансировка.

6. Перечислите основные классы моделей параллелизма.

7. Определите общую схему параллельных популяционных алгоритмов оптимизации, использующих глобальную модель параллелизма.

8. Изложите схему параллельных популяционных алгоритмов оптимизации, использующих островную и диффузную модели параллелизма.

9. Назовите основные особенности следующих моделей параллелизма: модель некоммутирующей мультипопуляции, пространственно вложенная модель, модель расщепления, модель коэволюции, комбинированная модель.

Литература

1. Аттетков А. В., Галкин С. В., Зарубин В. С. Методы оптимизации: Учебник для ВУЗов.– М.: Из-во МГТУ им. Н. Э. Баумана, 2001, 439 с.

2. Воеводин В. В., Воеводин Вл.В. Параллельные вычисления. СПб.: БХВ-Петербург, 2002, 608 с.

3. Курейчик В. В., Курейчик В. М., Родзин С. И. Теория эволюционных вычислений. М.: ФИЗМАТЛИТ, 2012, 260 с.

4. Ларичев О. И. Теория и методы принятия решений: Учебник для ВУЗов. М.: Университетская книга, Логос, 2006, 392 с.

5. Рутковская Д. Нейронные сети, генетические алгоритмы и нечеткие системы: Пер. с польск. / Рутковская Д., Пилиньский М., Рутковский Л. М.: Горячая линия–Телеком, 2004, 452 с.

6. Семенкин Е. С. Эволюционные методы моделирования и оптимизации сложных систем / Семенкин Е. С. и др. // Электронные публикации НГУ. (http://library.krasu.ru/ft/ft/_umkd/22/u_lectures.pdf)

7. Черноруцкий И. Г. Методы принятия решений: Учебное пособие. СПб.: БХВ-Петербург, 2005, 416 с.

8. Штойер Р. Многокритериальная оптимизация. М.: Радио и связь, 1992, 504 с.

9. Blum Ch., Aguilera M. J.B., Roli A., Sampels M. (Eds.) Hybrid Metaheuristics. An Emerging Approach to Optimization. Springer-Verlag, Berlin, Germany, 2008, 289 pp.

10. Deb K. Multi-objective optimization using evolutionary algorithms. Chichester, UK: Wiley, 2001, 518 pp.

Дополнительная литература

1. Гладков Л. А., Курейчик В. В., Курейчик В. М., Сороколетов П. В. Биоинспирированные методы в оптимизации. М.: ФИЗМАТЛИТ, 2009, 384 с.

2. Семенкин Е. С., Жукова М. Н., Жуков В. Г., Панфилов И. А., Тынченко В. В. Эволюционные методы моделирования и оптимизации сложных систем. Конспект лекций. Красноярск, 2007, 515 с. (http://files.lib.sfu-kras.ru/ebibl/umkd/22/u_lectures.pdf)

3. Ashlock D. Evolutionary Computation for Modeling and Optimization. Springer-Verlag, Berlin, Germany, 2006, 571 pp.

4. Blum C., Andrea R., Michael S. (Eds.) Hybrid Metaheuristics. An Emerging Approach to Optimization. Springer-Verlag, Berlin, Germany, 2008, Vol. 114, 290 p.

5. Dos Santos W. P. (ed.). Evolutionary Computation. InTech, 2009, 582 pp.

6. Engelbrecht A. P. Computational Intelligence. An Introduction. Second Edition. John Wiley & Sons, 2007, 597 pp.

7. Kita E. (ed.). Evolutionary Algorithms. InTech, 2011, 596 pp.

8. Luke S. Essentials of Metaheuristics, Lulu, 2009. (http://cs.gmu.edu/~sean/book/metaheuristics)

9. Pohlheim H. Evolutionary Algorithms: Overview, Methods and Operators. (http://www.geatbx.com/)

10. Yang X. S. Nature-inspired Metaheuristic Algorithms. Luniver Press, 2010, UK, pp. 81–95.

11. Suganthan P. N., Hansen N., Liang J. J., Deb K., Chen Y.P., Auger A., Tiwari S. Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization. Technical Report / Nanyang Technological University, Singapore, 2005 (http://www.ntu.edu.sg/home/EPNSugan)

12. Tang K., Yao X., Suganthan P. N., MacNish C., Chen Y. P., Chen C. M., Yang Z. Benchmark Functions for the CEC'2008 Special Session and Competition on Large Scale Global Optimization. Technical Report / Nature Inspired Computation and Applications Laboratory, USTC, China, 2007. (http://nical.ustc.edu.cn/cec08ss.php)

13. Zhang Q., Zhou A., Zhao Sh., Suganthan P. N., Liu W., Tiwari S. Multiobjective optimization Test Instances for the CEC 2009 Special Session and Competition. Technical Report CES-487 / The School of Computer Science and Electronic Engineering University of Essex, Colchester, C04, 3SQ, UK, 2009, 30 pp.

14. Zitzler E., Deb K., Thiele L. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results // Evolutionary Computation, 2000, Vol. 8 (2), pp. 173–195.

Приложение А

История разработки популяционных алгоритмов поисковой оптимизации

Год	Наименование алгоритма	
	Английское	Русское
2010	*Charged System Search, CSS*	Поиск системой зарядов
	Bat-Inspired Algorithm, BIA	Алгоритм, инспирированный летучими мышами
2009	*Gravitational Search, GS*	Гравитационный поиск
	Cuckoo Search, CS	Кукушкин поиск
2008	*Fish School Search, FSS*	Поиск косяком рыб
	Monkey Search, MS	Обезьяний поиск
2007	*FireFlay Algorithm, FFA*	Алгоритм светлячков
	Monkey algorithm	Алгоритм обезьян
2006	*Invasive Weed Optimization, IWO*	Алгоритм сорняковой оптимизации
2005	*Bees Algorithm, BA*	Пчелиный алгоритм
	Glowworm Swarm Optimization, GSO	Алгоритм оптимизации роем светлячков
2003	*ElectroMagnetism-like algorithm, EM algorithm*	Электромагнитный алгоритм
	Shuffled Frog-Leaping, SFL	Тасующий алгоритм прыгающих лягушек
2002	*Bacterial Optimization, BO*	Бактериальная оптимизация
	Saplings Sowing and Growing up, SSG	Алгоритм растущих деревьев
2001	*Harmony search*	Гармонический поиск
1999	*Self Organizing Migrating Algorithm, SOM*	Самоорганизующийся миграционный алгоритм
1998	*Mind Evolutionary Computation, MEC*	Алгоритм эволюции разума
1997	*Scatter Search, SS*	Рассеянный поиск
	Path Relinking, PR	Алгоритм прокладки путей
1995	*Differential Evolution, DE*	Дифференциальная эволюция
	Particle Swarm Optimization, PSO	Оптимизация роем частиц
	Continuous Ant Colony Optimization, CACO	Непрерывная оптимизация колонией муравьев
1994	*Cultural Algorithm, CA*	Культурный алгоритм
1992	*Ant Colony Optimization, ACO*	Оптимизация колонией муравьев
1989	*Stochastic Diffusion Search, SDS*	Стохастический диффузионный поиск

Год	Наименование алгоритма	
	Английское	Русское
1986	*Artificial Immune Systems, AIS*	Искусственные иммунные системы
1975	*Genetic Algorithm, GA*	Генетический алгоритм
1966	*Evolution Programming, EP*	Эволюционное программирование
1964	*Evolution Strategies, ES*	Эволюционные стратегии

Приложение Б

Тестовые функции для одноцелевой задачи глобальной оптимизации

В представленной ниже табл. Б.1 запись вида $[\alpha;\beta]^{|X|}$ означает гиперкуб допустимых значений вектора варьируемых параметров

$$\Pi = \left(X \mid \alpha \le x_i \le \beta,\ i \in [1:|X|]\right);$$

x_i^0, f^0 – свободные параметры; $z_i = x_i - x_i^0$; $i \in [1:|X|]$.

Таблица Б.1 **Тестовые функции для одноцелевой задачи глобальной оптимизации**

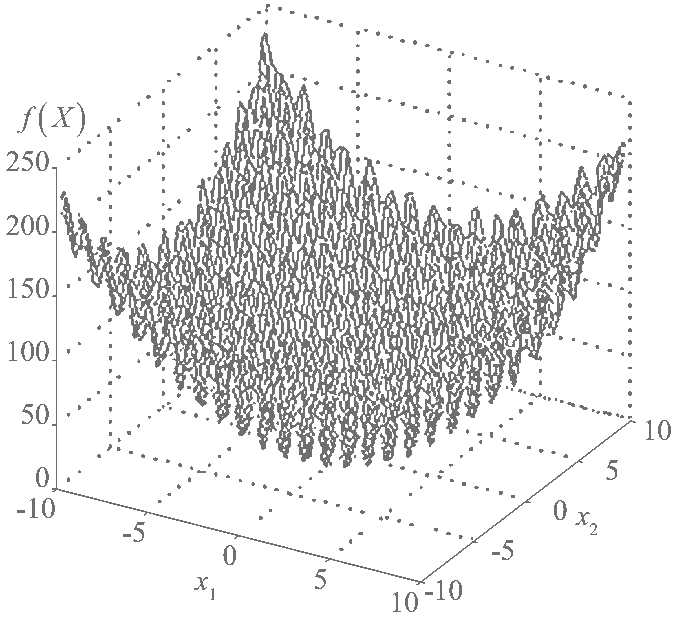
Область поиска	Функция
	Одномерные функции
$[-\alpha;\alpha]^1$ $\alpha = 2$	Функция «волна» $f(x) = e^{-x^2} + 0{,}01\cos(200x) \to \max$ $x^* = 0$ $f^* = 2$
$[-\alpha;\alpha]^1$ $\alpha = 2$	Производная от функции «волна» $f(x) = -2xe^{-x^2} - 2\sin(200x) \to \max$ $x^* = 0$ $f^* = -0{,}7$
	Двумерные функции
$[-\alpha;\alpha]^2$ $\alpha = 2{,}048$	Функция Де Янга (*De Jong*) $f(X) = 3905{,}93 - 100(x_1^2 - x_2^2)^2 - (1 - x_1)^2 \to \max$ $X^* = (1;1)$ $f^* = 3905{,}93$
$[-\alpha;\alpha]^2$ $\alpha = 100$	Функция Дэвиса (*Davis*) $f(X) = (x_1^2 + x_2^2)^{0{,}25}\left(\sin^2(50(x_1^2 + x_2^2)^{0{,}1})\right) + 1 \to \min$ $X^* = (0;0)$ $f^* = 0$
$[-\alpha;\alpha]^2$ $\alpha = 2$	Функция Голдштейна (*Goldshtein*) и Прайса (*Price*) $f(X) = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times$ $\times\left[30 + \left(2x_1 + 3x_2\right)^2\left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2\right)\right] \to \min$ $X^* = (0;-1)$ $f^* = 3$

Область поиска	Функция
$[-\alpha;\alpha]^2$ $\alpha = 0{,}5$	Функция Демирхана (*Demirhan*) $f(X) = 0{,}5 + \dfrac{\sin^2\sqrt{x_1^2 + x_2^2} - 0{,}5}{\left(1{,}0 + 0{,}001\,(x_1^2 + x_2^2)\right)^2} \to \min$ $X^* = (0;\,0)$ $f^* = 0$
$[-\alpha;\alpha]^2$ $\alpha = 10$	Функция Мартина (*Martin*) и Гадди (*Gaddy*) $f(X) = (x_1 - x_2)^2 + ((x_1 + x_2 - 10)/3)^2 \to \min$ $X^* = (5;\,5)$ $f^* = 0$
$[\alpha;\beta]^2$ $\alpha_1 = -3$ $\beta_1 = 12{,}1$ $\alpha_2 = 4{,}1$ $\beta_2 = 5{,}8$	Функция Михалевича (*Michalewicz*) $f(X) = -21{,}5 - x_1 \sin(4\pi\, x_1) - x_2 \sin(20\pi\, x_2) \to \min$ $X^* = (11{,}62523\,;\,5{,}72082)$ $f^* = -38{,}85$
$[-\alpha;\alpha]^2$ $\alpha = 2$	Функция Пресса (*Press*) $f(X) = \left(x_1^3 - 3x_1x_2^2 - 1\right)^2 + \left(3x_1^2x_2 - x_3^3\right)^2 \to \min$ $X^* = \left\{(-1;\,0), \left(-\dfrac{1}{2}, \sqrt{\dfrac{1}{2}}\right), \left(-\dfrac{1}{2}, \sqrt{\dfrac{3}{2}}\right)\right\}$ $f^* = 0$
$[\alpha;\beta]^2$ $\alpha = -1$ $\beta = 4$	Функция Стенгера (*Stenger*) $f(X) = \left(x_1^2 - 4x_2\right)^2 + \left(x_2^2 - 2x_1 + 4x_2\right)^2 \to \min$ $X^* = \{(0;\,0), (1{,}695415;\,0{,}7186082)\}$ $f^* = 0$
$[-\alpha;\alpha]^2$ $\alpha = 5{,}12$	Функция ступенька (*Step*) $f(X) = \sum_{i=1}^{\lvert X\rvert} \lfloor x_i \rfloor$ $X^* = ([-5{,}12;\,-5);[-5{,}12;\,-5);\ldots;[-5{,}12;\,-5)\}$ $f^* = 0$
$[-\alpha;\alpha]^2$ $\alpha = 6$	Функция Химмельблау (*Himmelblau*) $f(X) = \left(x_1^2 + x_2 - 11\right)^2 + \left(x_1 + x_2^2 - 7\right)^2 \to \min$ $X^* = \{(3;\,2), (-2{,}805118;\,3{,}131312), (3{,}584428;\,-1{,}848126),$ $(-3{,}779310;\,-3{,}283186)\}$ $f^* = 0$

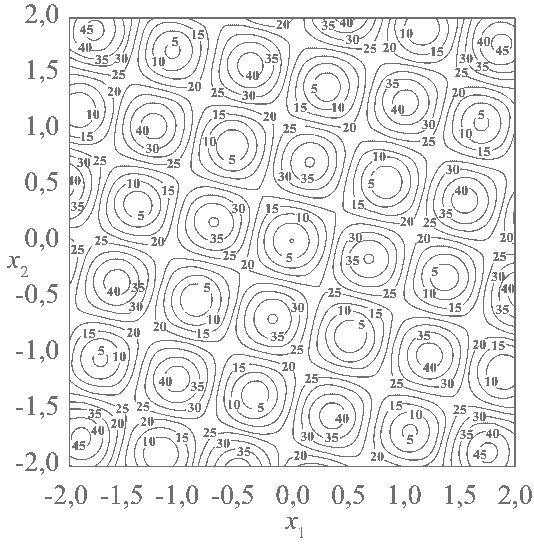
Область поиска	Функция
	Многомерные функции
$[-\alpha;\alpha]^{\lvert X\rvert}$ $\alpha = 600$	Функция Гриванка (*Griewank*) $f(X) = \sum_{i=1}^{\lvert X\rvert} \frac{z_i^2}{4000} - \prod_{i=1}^{\lvert X\rvert} \cos\left(\frac{z_i}{\sqrt{i}}\right) + 1 + f^0 \to \min$ $X^* = (x_1^0; \ldots; x_{\lvert X\rvert}^0)$ $f^* = f^0 = -180$
$[\alpha;\beta]^4$ $\alpha = -1$ $\beta = 10$	Функция Кирфута (*Kearfott*) $f(X) = \left(x_1^2 - x_2^2\right)^2 + \left(x_2^2 - x_3^2\right)^2 + (x_3^2 - x_4^2)^2 + (x_4^2 - x_1^2)^2 \to \min$ $X^* = (0; 0; 0; 0)$ $f^* = 0$
$[-\alpha;\alpha]^{\lvert X\rvert}$ $\alpha = 10$	Функция Леви (*Levy*) $f(X) = \sin^2\left(\pi\left(1 + \frac{x_1 - 1}{4}\right)\right) + \sum_{i=2}^{9}\left(\frac{x_{i-1} - 1}{4}\right)^2 \left[1 + 10\sin^2\left(\pi\left(1 + \frac{x_i - 1}{4}\right)\right)\right] +$ $+ \left(\frac{x_{10} - 1}{4}\right)^2 \to \min$ $X^* = (0; 0; \ldots; 0)$ $f^* = 0$
$[-\alpha;\alpha]^5$ $\alpha = 25$	Пятимерная функция Ньюмайера (*Neumaier*) $f(X) = \sum_{i=1}^{5}(x_i - 1)^2 - \sum_{i=2}^{5} x_i\, x_{i-1} \to \min$ $X^* = (5; 8; 9; 8; 5)$ $f^* = -30$
$[-\alpha;\alpha]^{20}$ $\alpha = 25$	Двадцатимерная функция Ньюмайера (*Neumaier*) $f(X) = \sum_{i=1}^{20}(x_i - 1)^2 - \sum_{i=2}^{20} x_i\, x_{i-1} \to \min$ $X^* = (20; 38; 54; 68; 80; 90; 98; 104; 108; 110; 110; 108; 104; 98; 90; 80; 68; 54;$ $38; 20)$ $f^* = -1520$
$[-\alpha;\alpha]^{\lvert X\rvert}$ $\alpha = 5$	Функция Растригина (*Rastrigin*) – рис. Б.1 $f(X) = \sum_{i=1}^{\lvert X\rvert}\left(z_i^2 - 10\cos(2\pi z_i) + 10\right) + f^0 \to \min$ $X^* = (x_1^0; \ldots; x_{\lvert X\rvert}^0)$ $f^* = f^0 = -330$

<table>
<tr><th>Область поиска</th><th>Функция</th></tr>
<tr><td>$[-\alpha;\alpha]^{|X|}$
$\alpha = 100$</td><td>Функция Розенброка (Rozenbrock)
$f(X) = \sum_{i=1}^{|X|-1}\left(100\,(z_i^2 - z_{i+1})^2 + (z_i - 1)^2\right) + f^0 \to \min$
$X^* = (x_1^0;\ldots;x_{|X|}^0)$
$f^* = f^0 = 390$</td></tr>
<tr><td>$[-\alpha;\alpha]^{|X|}$
$\alpha = 0{,}5$</td><td>Функция Сриниваса (Srinivas) и Патнайка (Patnaik)
$f(X) = \sum_{i=1}^{|X|}\frac{\sin(\pi k x_i)}{\pi k x_i} \to \min$
$X^* = (0;0;\ldots;0)$
$f^* = 0$</td></tr>
<tr><td>$[-\alpha;\alpha]^{|X|}$
$\alpha = 100$</td><td>Функция «сфера» (sphere)
$f(X) = \sum_{i=1}^{|X|} z_i^2 + f^0 \to \min$
$X^* = (x_1^0;\ldots;x_{|X|}^0)$
$f^* = f^0 = -450$</td></tr>
<tr><td>$[-\alpha;\alpha]^{|X|}$
$\alpha = 1$</td><td>Функция «фрактал»
$f(X) = \sum_{i=1}^{|X|}\phi\left(x_i + t\left(x_{(i \bmod |X|)+1}\right)\right) \to \min,\quad t(y) = 4\left(y^4 - 2y^3 - y^2\right)$
$\phi(x) = \sum_{k=1}^{3}\sum_{1}^{2^{k-1}}\sum_{1}^{U_1(0;2)} d\left(x, U_1(0;1), \frac{1}{2^{k-1}\left(2 - U_1(0;1)\right)}\right)$
$d(x,c,s) = \begin{cases}\left(-6144(x-c)^6 + 3088(x-c)^4 - 392(x-c)^2 + 1\right)s, & x \in (-0{,}5;0{,}5)\\ 0, & \text{иначе}\end{cases}$
X^*, f^* – неизвестны</td></tr>
<tr><td>$[\alpha;\beta]^4$
$\alpha = 0$
$\beta = 1$</td><td>Трехмерная функция Хартмана (Hartman)
$f(X) = -\sum_{j=1}^{4}\left(c_i \exp\left(-\sum_{i=1}^{3} a_{i,j}(x_i - p_{i,j})^2\right)\right) \to \min$
<table>
<tr><th>j</th><th>$a_{1,j}$</th><th>$a_{2,j}$</th><th>$a_{3,j}$</th><th>$p_{1,j}$</th><th>$p_{2,j}$</th><th>$p_{3,j}$</th><th>c_j</th></tr>
<tr><td>1</td><td>3,0</td><td>10</td><td>30</td><td>0,36890</td><td>0,11700</td><td>0,26730</td><td>1,0</td></tr>
<tr><td>2</td><td>0,1</td><td>10</td><td>35</td><td>0,46990</td><td>0,43870</td><td>0,74700</td><td>1,2</td></tr>
<tr><td>3</td><td>3,0</td><td>10</td><td>30</td><td>0,10910</td><td>0,87320</td><td>0,55470</td><td>3,0</td></tr>
<tr><td>4</td><td>0,1</td><td>10</td><td>35</td><td>0,03815</td><td>0,57430</td><td>0,88280</td><td>3,2</td></tr>
</table>
$X^* = (0{,}1;\ 0{,}55592;\ 0{,}85218)$
$f^* = -3{,}8628$</td></tr>
</table>

<table>
<tr><th>Область поиска</th><th>Функция</th></tr>
<tr><td>$[\alpha;\beta]^4$
$\alpha=0$
$\beta=1$</td><td>
Шестимерная функция Хартмана (Hartman)
$$f(X)=-\sum_{j=1}^{4}\left(c_i\exp\left(-\sum_{i=1}^{6}a_{i,j}(x_i-p_{i,j})^2\right)\right)\to\min$$
<table>
<tr><th>j</th><th>$a_{1,j}$</th><th>$a_{2,j}$</th><th>$a_{3,j}$</th><th>$a_{4,j}$</th><th>$a_{5,j}$</th><th>$a_{6,j}$</th></tr>
<tr><td>1</td><td>10,00</td><td>3,0</td><td>17,00</td><td>3,5</td><td>1,7</td><td>8,0</td></tr>
<tr><td>2</td><td>0,05</td><td>10,0</td><td>17,00</td><td>0,1</td><td>8,0</td><td>14,0</td></tr>
<tr><td>3</td><td>3,00</td><td>3,5</td><td>1,70</td><td>10,0</td><td>17,0</td><td>8,0</td></tr>
<tr><td>4</td><td>17,00</td><td>8,0</td><td>0,05</td><td>10,0</td><td>0,1</td><td>14,0</td></tr>
</table>
<table>
<tr><th>j</th><th>$p_{1,j}$</th><th>$p_{2,j}$</th><th>$p_{3,j}$</th><th>$p_{4,j}$</th><th>$p_{5,j}$</th><th>$p_{6,j}$</th><th>c_j</th></tr>
<tr><td>1</td><td>0,1312</td><td>0,1696</td><td>0,5569</td><td>0,0124</td><td>0,8283</td><td>0,5886</td><td>1,0</td></tr>
<tr><td>2</td><td>0,2329</td><td>0,4135</td><td>0,8307</td><td>0,3736</td><td>0,1004</td><td>0,9991</td><td>1,2</td></tr>
<tr><td>3</td><td>0,2348</td><td>0,1451</td><td>0,3522</td><td>0,2883</td><td>0,3047</td><td>0,6650</td><td>3,0</td></tr>
<tr><td>4</td><td>0,4047</td><td>0,8828</td><td>0,8732</td><td>0,5743</td><td>0,1091</td><td>0,0381</td><td>3,2</td></tr>
</table>
$X^*=(0,20169;0,15001;0,47687;0,2753;0,31165;0,65730)$
$f^*=-3,3224$
</td></tr>
<tr><td>$[-\alpha;\alpha]^{|X|}$
$\alpha=100$</td><td>
Функция Швефеля (Schwefel)
$$f(X)=\max_i\left\{abs\,(z_i),\ i\in[1:|X|]\right\}\to\min$$
$X^*=(x_1^0;...;x_{|X|}^0)$
$f^*=f^0=-450$
</td></tr>
<tr><td>$[\alpha;\beta]^{|X|}$
$x_{i,j}^0\in\Pi$,
$i,j\in[1:|X|]$</td><td>
Функция Шекеля (Shekel) – рис. Б.2
$$f(X)=-\sum_{i=1}^{|X|}\frac{1}{f_i^0+\sum_{j=1}^{|X|}\left(x_j-x_{i,j}^0\right)^2}\to\min,\quad\sum_{i=1}^{|X|}\frac{1}{f_i^0}=f^0$$
$X^*=(x_1^0;...;x_{|X|}^0)$
$f^*=f^0=-1$
</td></tr>
<tr><td>$[-\alpha;\alpha]^{|X|}$
$\alpha=32$</td><td>
Функция Экли (Ackley) – рис. Б.3
$$f(X)=-20\exp\left(-0,2\sqrt{\frac{1}{|X|}\sum_{i=1}^{|X|}z_i^2}\right)-\exp\left(\frac{1}{|X|}\sum_{i=1}^{|X|}\cos(2\pi z_i)\right)+20+f^0\to\min$$
$X^*=(x_1^0;...;x_{|X|}^0)$
$f^*=f^0=-140$
</td></tr>
</table>



а



б

Рис. Б.1. Ландшафт (*а*) и линии уровня (*б*) функции Растригина: $|X| = 2$; $x_1^0 = x_2^0 = f^0 = 0$

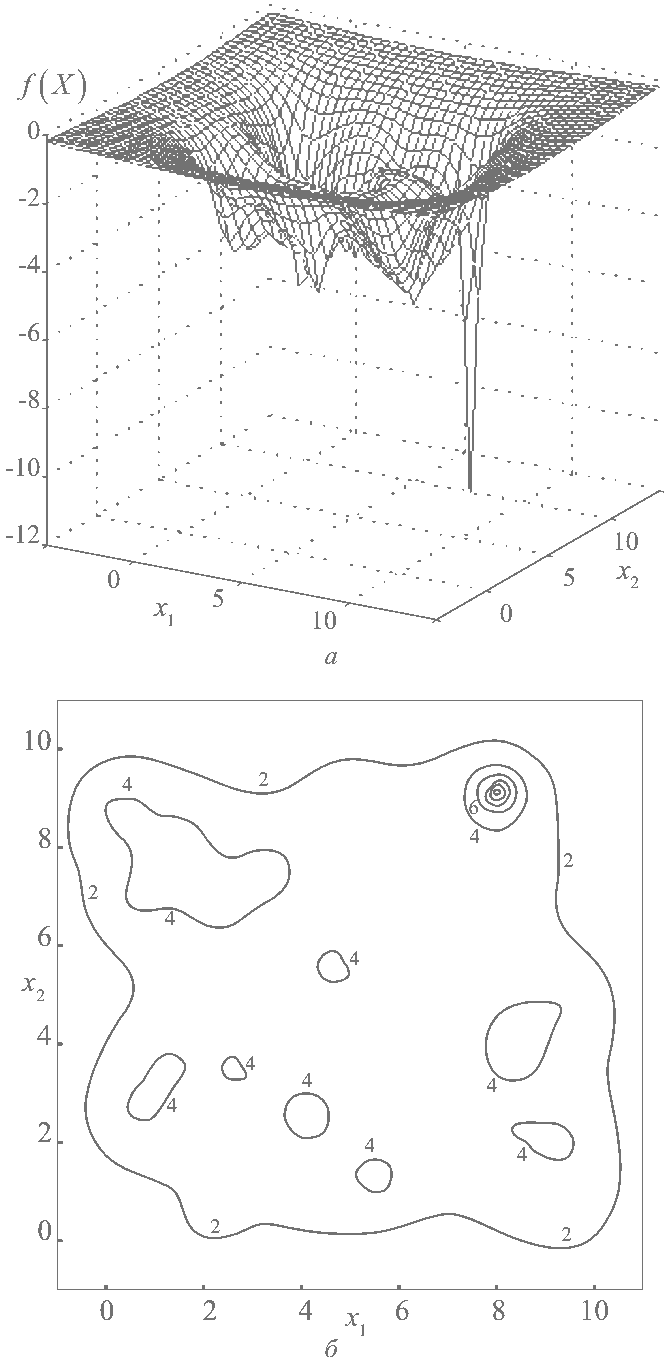
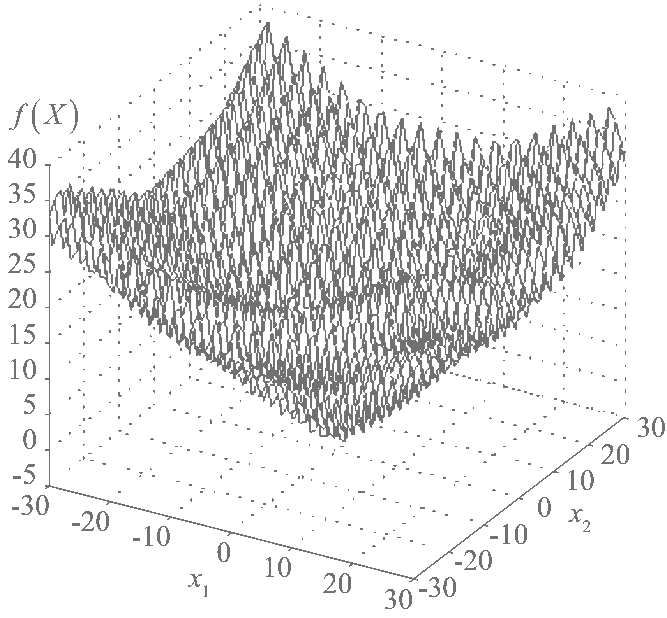
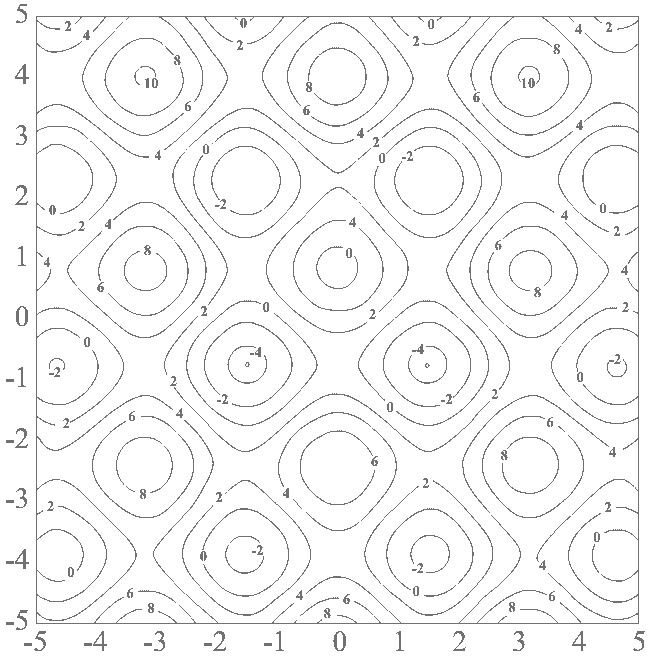


Рис. Б.2. Ландшафт (*а*) и линии уровня (*б*) функции Шекеля: $|X| = 2$; $f^* = f^0 = -1$



а



б

Рис. Б.3. Ландшафт (*а*) и линии уровня (*б*) функции Экли: $|X| = 2$; $x_1^0 = x_2^0 = f^0 = 0$

Большое число очень сложных тестовых задач глобальной оптимизации представлено в известной базе данных Кембриджа по молекулярным кластерам (*The Cambridge Cluster Database*). База доступна по адресу (*http://www-wales.ch.cam.ac.uk/CCD*).

Тестовые задачи многоцелевой оптимизации

В табл. В.1 представлены тестовые задачи двухцелевой оптимизации без ограничений на значения целевых функций. Задачи предложили Цицлер (*E. Zitzler*), Деб (*K. Deb*) и Тиле (*L. Thiele*) в 2000 г. Общий вид целевых функций определяют выражения

$$f_1(x_1) \to \min,$$

$$f_2(x_2, x_3, ..., x_{|X|}) = g(x_2, x_3, ..., x_{|X|}) \cdot h\left(f_1(x_1), g(x_2, x_3, ..., x_{|X|})\right) \to \min,$$

где $g(\cdot), h(\cdot)$ – функции указанных аргументов.

Мы не приводим тестовую задачу *ZDT5*, поскольку компоненты вектора варьируемых параметров этой задачи являются бинарными (а предметом работы являются только задачи непрерывной оптимизации).

Таблица В.1 **Тестовые задачи многоцелевой оптимизации: ограничения на значения целевых функций отсутствуют**

Задача	Целевые функции
ZDT1 $x_i \in [0; 1]$ $\lvert X \rvert = 30$	$f_1(x_1) = x_1$ $f_2(x_2, x_3, ..., x_{\lvert X \rvert}) = g(x_2, x_3, ..., x_{\lvert X \rvert}) \cdot h\left(f_1, g\right)$ $g\left(x_2, ..., x_{\lvert X \rvert}\right) = 1 + 9\sum_{i-2}^{\lvert X \rvert} \frac{x_i}{\lvert X \rvert - 1}$ $h(f_1, g) = 1 - \sqrt{\frac{f_1}{g}}$
ZDT2 $x_i \in [0; 1]$ $\lvert X \rvert = 30$	$f_1(x_1) = x_1$ $f_2(x_2, x_3, ..., x_{\lvert X \rvert}) = g(x_2, x_3, ..., x_{\lvert X \rvert}) \cdot h\left(f_1, g\right)$ $g\left(x_2, ..., x_{\lvert X \rvert}\right) = 1 + 9\sum_{i-2}^{\lvert X \rvert} \frac{x_i}{\lvert X \rvert - 1}$ $h(f_1, g) = 1 - \left(\frac{f_1}{g}\right)^2$
ZDT3 $x_i \in [0; 1]$ $\lvert X \rvert = 30$	$f_1(x_1) = x_1$ $f_2(x_2, x_3, ..., x_{\lvert X \rvert}) = g(x_2, x_3, ..., x_{\lvert X \rvert}) \cdot h\left(f_1, g\right)$ $g\left(x_2, ..., x_{\lvert X \rvert}\right) = 1 + 9\sum_{i-2}^{\lvert X \rvert} \frac{x_i}{\lvert X \rvert - 1}$ $h\left(f_1, g\right) = 1 - \sqrt{\frac{f_1}{g}} - \frac{f_1}{g}\sin\left(10\pi f_1\right)$

Задача	Целевые функции
ZDT4 $x_1 \in [0;1]$ $x_i \in [-5;5], i \in [2:\lvert X\rvert]$ $\lvert X\rvert = 10$	$f_1(x_1) = x_1$ $f_2(x_2, x_3, ..., x_{\lvert X\rvert}) = g(x_2, x_3, ..., x_{\lvert X\rvert}) \cdot h(f_1, g)$ $g\left(x_2, ..., x_{\lvert X\rvert}\right) = 1 + 10\left(\lvert X\rvert - 1\right) + \sum_{i=2}^{\lvert X\rvert}\left(x_1^2 - 10\cos(4\pi x_i)\right)$ $h(f_1, g) = 1 - \sqrt{\frac{f_1}{g}}$
ZDT6 $x_i \in [0;1]$ $\lvert X\rvert = 10$	$f_1(x_1) = 1 - \exp(-4x_1) \cdot \sin^6(6\pi x_1)$ $f_2(x_2, x_3, ..., x_{\lvert X\rvert}) = g(x_2, x_3, ..., x_{\lvert X\rvert}) \cdot h(f_1, g)$ $g\left(x_2, ..., x_{\lvert X\rvert}\right) = 1 + 9\left(\frac{1}{\lvert X\rvert - 1}\sum_{i=2}^{\lvert X\rvert} x_i\right)^{0,25}$ $h(f_1, g) = 1 - \left(\frac{f_1}{g}\right)^2$

Задача *ZDT1* имеет выпуклый фронт Парето (см. рис. 8.23), который определяет равенство $g(x_2, x_3, ..., x_{|X|}) = 1$.

Задача *ZDT2* обладает невыпуклым фронтом Парето (см. рис. 8.24), который также определяет равенство $g(x_2, x_3, ..., x_{|X|}) = 1$.

Фронт Парето задачи *ZDT3* является дискретным, состоящим из некоторого числа выпуклых фрагментов (см. рис. 8.25). Множество Парето задачи при этом является непрерывным. Фронт Парето, как и в предыдущих задачах, определяет равенство $g(x_2, x_3, ..., x_{|X|}) = 1$.

Особенностью задачи *ZDT4* является наличие большого числа локальных субоптимальных фронтов (см. рис. 8.26). Точный фронт Парето попрежнему определяет равенство $g(x_2, x_3, ..., x_{|X|}) = 1$, а лучший локальный фронт – равенство $g(x_2, x_3, ..., x_{|X|}) = 1,25$. Авторы теста утверждают, что не все локальные фронты различимы в пространстве параметров.

Задача *ZDT6* имеет невыпуклый фронт Парето (рис. В.1).

Известно большое число других тестовых задач многоцелевой оптимизации, как без ограничений на значения целевых функций, так и с такими ограничениями. Представительный список таких задач можно найти, например, в работе Чжана (*Q. Zhang*) с соавторами, опубликованной в 2009 г.

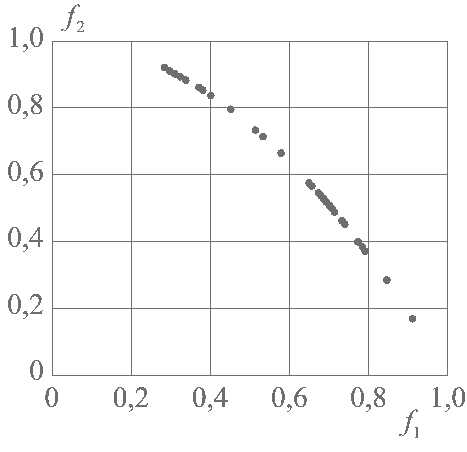


Рис. В.1. Задача *ZDT6*: дискретная аппроксимация фронта Парето

Предметный указатель

Алгоритм
– адаптивный взвешенных сумм 8.2.2
– бактериальной оптимизации 4.2
  – адаптивный (*ABFO*) 4.2.3
  – использующий эффект роения бактерий 4.2.4
  – канонический (*BFO*) 4.2.2
  – кооперативный (*CBFO*) 4.2.3
    – гибридный (*CBFO–H*) 4.2.3
    – последовательный (*CBFO–S*) 4.2.3
  – взвешенных целевых функций 8.2.2
  – восхождения на гору 2.7
  – генетический
    – адаптивный 2.7
    – бинарный 2.2
    – гибридный 2.7
    – канонический 2.7
    – клеточный 2.7, 9.4.3
    – коэволюционный 2.12
    – многопопуляционный 2.7
    – непрерывный 2.2
    – островной 2.7
    – простой 2.7
    – сегрегированный 2.6
    – *CHC* *2.7*
    – *Genitor* 2.7
    – *NPGA* 8.3.3
– генетического программирования 7.3.2
– гиперкубовый 8.3.4
– гиперсферы 1.2.3
– динамического симплекса 3.2.5
– динамического соседства 8.3.2
– дифференциальной эволюции 2.11
– зондирования 8.1.2
– идеальной точки 8.2.2
– имитации отжига (*SA*) 1.4.2
– инспирированный летучими мышами (*BI*) 4.7.2
– искусственной иммунной системы 4.1
– клонального отбора (*CLONALG*) 4.1.3
– колонии искусственных пчел (*ABC*) 3.3.3
– колонии муравьев (*ACO)* 3.2
  – непрерывно взаимодействующей (*CIAC*) 3.2.3
  – непрерывный (*CACO*) 3.2.2
    – гибридный (*HCIAC*) 3.2.5
      – динамический (*DHCIAC*) 3.2.5
    – ортогональный (*COAC*) 3.2.4
– коммутирующей мультипопуляции 9.4.2
– комплексов 1.2.3
  – модифицированный 1.3.2
– композитных точек 8.3.3
– конфигураций 1.2.3
– кукушкиного поиска (*CS*) 4.5
  – улучшенный (*ICS*) 4.5
  – модифицированный (*MCS*) 4.5
– культурный 5.6
– лексикографического упорядочения 8.1.2
– лексикографической турнирной селекции 8.3.1
– локальной унимодальной выборки (*LUS*) 1.2.1
– локальный хаотический (*CLS*) 1.4.2
– мемеевый 5.7
  – адаптивный 5.7.2
  – самоадаптирующийся 5.7.3
– микроиммунной системы (*Micro-AIS*) 4.1.3
– миграционный 9.4.2
– Монте–Карло 1.4.2
– мультимемный 5.7.1
– мультистарта 1.4.2
– недоминируемой сортировки (*NDS*) 8.3.3
– обезьяний 4.6.2
– обезьяньего поиска (*MS*) 4.6.1
– одномерного рандомизированного поиска (*SDPS*) 1.2.1
– одномерных вложений 1.4.1
– Парето-
  – аппроксимации 8
  – силы 8.3.3
– переключающихся целевых функций 8.3.2
– поиска
  – гармонии (*HS*)
    – глобально лучший (*GHS*) 5.1.2
    – дифференциальный (*DHS*) 5.1.2
    – канонический 5.1.1

– улучшенный (*IHS*) 5.1.2
– гравитационный (*GS*) 5.2
– косяком рыб (*FSS*) 4.7.3
– плотный (*DFSS*) 4.7.3
– рассеянный (*SS*) 5.9.1
– с чередующимися окрестностями 1.4.2
– системой зарядов (*CSS*) 5.3.2
– со случайно чередующимися окрестностями (*RVNS*) 1.4.2
– стохастического диффузионного (*SDS*) 5.5
– контекстно-зависимый 5.5
– фокусированный 5.5
– пороговый 1.4.2
– последовательных уступок 8.1.2
– прокладки путей (*PR*) 5.9.2
– вперед 5.9.2
– комбинированный 5.9.2
– многородительский 5.9.2
– назад 5.9.2
– периодический 5.9.2
– рандомизированный 5.9.2
– смешанный 5.9.2
– усеченный 5.9.2
– прыгающих лягушек тасующий (*SFL*) 4.7.1
– пчелиного роя
– базовый (*BA*) 3.3.2
– улучшенный (*IBA*) 3.3.2
– растущих деревьев (*SSG*) 4.7.4
– роя частиц (*PSO*) 3.1
– волевой (*VPSO*) 4.7.3
– канонический 3.1.1
– клубный (*CBPSO*) 3.1.4
– многоцелевой (*MOPSO*) 8.6.2, 9.5.2
– многороевый (*MS MOPSO*) 8.6.2
– притягательно-отталкивающий (*ARPSO*) 3.1.2
– ранжированный (*RFIPS*) 3.1.2
– с весами (*WFIPS*) 3.1.2
– с дополнением графа соседства (*DSPSO*) 3.1.4
– со стереотипированием 3.1.4
– с полной информацией (*FIPS*) 3.1.2
– с управлением скоростью частиц 3.1.2
– *ARPSO* 3.1.2
– *FDR* 3.1.2
– с возвратом при неудачном шаге 1.2.1
– с запретами (*TS*) 1.4.3
– с чередующимися окрестностями (*VNS*) 1.4.2
– самоорганизующийся миграционный (*SOM*) 5.8
– сверхбыстрого отжига 1.4.2
– светлячковый 4.3.1
– роевый (*GSO*) 4.3.2
– случайного поиска (*RS*) 1.2.1
– адаптивный (*ASSRS*) 1.2.1
– адаптивный максимизирующий (*AMRS*) 1.2.2
– жадный (*GRASP*) 1.4.2
– неадаптивный (*FSSRS*) 1.2.1
– с интенсификацией и диверсификацией (*ARSID*) 1.2.1
– с постоянным радиусом поиска и случайными направлениями 1.2.3
– сорняковой оптимизации (*IWO*) 4.4
– стохастический линейный (*RLS*) 1.2.1
– ранжирования
– средневзвешенного (*WAR*) 8.3.3
– максимального взвешенного (*WMR*) 8.3.3
– стохастической аппроксимации (*SPSA*) 1.2.1
– суммы взвешенных оценок (*SWR*) 8.3.3
– глобальных (*SWGR*) 8.3.3
– с эффектом Болдуина 2.7
– «хищник–жертва» 8.3.2
– центральной силы 5.2
– чередующихся целевых функций 8.3.2
– штрафных функций 1.3.1
– эволюции разума (*MEC*) 5.4
– простой (*SMEC*) 5.4.1
– расширенный (*EMEC*) 5.4.2
– улучшенный (*IMEC*) 5.4.3
– хаотический (*CMEC*) 5.4.4
– эвристический на основе паттернов (*HPS*) 1.2.1
– электромагнитный (*EM*)
– канонический 5.3.1
– *FFGA* 8.3.3
– *k*-средних 7.4.2
– *MOGA* 8.3.3
– *NSGA* 8.3.3
– *NSGA II* 8.3.3
– *PAES* 8.3.3
– *SPEA* 8.3.3
– *SPEA 2* 8.3.3
– *VEGA* 8.3.2

Аппроксимация направления убывания функции (*Approximate Descent Direction*, *ADD*) 1.2.1
Аутбридинг 2.5.2
Гибридизация
– вложением 6.1.1
– высокоуровневая 6.1.1
– последовательная 6.2.1
– на основе декомпозиции пространства поиска 6.2.1
– низкоуровневая 6.1.1, 6.2.2
– высокого уровня 6.1.3
– коалгоритмическая 6.1.1
– низкого уровня 6.1.3
– параллельная 6.1.3
– последовательная 6.1.3
– типа препроцессор / постпроцессор 6.1.1, 6.3
– конвейерная 6.1.1, 6.3.2
– последовательная 6.1.1, 6.3.1
– чередующаяся 6.1.3
Гиперэвристика 5.7.1
– жадная 5.7.2
– случайная 5.7.2
– с функцией выбора 5.7.2
Декомпозиция пространства поиска
– комбинированная 6.1.2
– неявная 6.1.2
– явная 6.1.2
Инбридинг 2.5.2
Интрон 7.6.1
Коалгоритм 6.4
Кодирование
– адаптивное 2.2
– бинарное 2.2
– векторное 2.2
– вещественное 2.2
– двоичное 2.2
– логарифмическое 2.2
– символьное 2.2
– числовое 2.2
Коэволюция
– двухпопуляционная 6.4.1
– многопопуляционная 6.4.1
– типа соперничества 6.4.1
– типа сотрудничества 6.4.1
Критерий
– бинарный 8.4
– оптимальности 1.1.1
– унимодальный 1.1.1
– скользящего допуска 1.3.1
– сходимости
– сильный 5.5
– слабый 5.5
– унарный 8.4
Кроссовер 2.1
– арифметический 2.4.2
– равномерный 2.4.2
– бинарный 2.4.1
– вещественный 2.4.2
– геометрический 2.4.2
– равномерный 2.4.2
– двухродительский 2.4.1
– двухточечный 2.4.1
– дискретный 2.4.2
– имитирующий бинарный (*SBX*) 2.4.2
– комбинированный 2.4.1
– линейный 2.4.2
– расширенный 2.4.2
– многородительский 2.4.1
– многоточечный 2.4.1
– нечеткий 2.4.2
– однородный 2.4.1
– одноточечный 2.4.1
– перетасовочный 2.4.1
– плоский 2.4.2
– простейший 2.4.2
– смешанный 2.4.2
– равномерный 2.4.2
– триадный 2.4.1
– упорядоченный 2.4.1
– эвристический 2.4.2
Мем 5.7
Мемеплекс 4.7.1
Метаоптимизация
– параметрическая 7.1.1
– динамическая 7.1.2, 7.3.2
– адаптивная 7.3.2
– неадаптивная 7.3.2
– статическая 7.1.1, 7.3.1
– структурная 7.1.3, 7.7
– динамическая 7.1.3
– адаптивная 7.1.3
– неадаптивная 7.1.3
– статическая 7.1.3
– структурно-параметрическая 7.1.4, 7.7
– статическая 7.1.4, 7.7
– динамическая 7.1.4, 7.7
– адаптивная 7.1.4
– неадаптивная 7.1.4

Метод
– лечения локальным поиском 1.3.2
– настройки параметров 7.2.1
  – однократный 7.2.1, 7.3
  – перманентный 7.2.1, 7.4
– нишевания
  – локального 8.5.2
– ограничения скрещивания 8.5.2
– отбора
  – вытеснением 2.5.1
  – на основе элитизма 2.5.1
  – отсечением 2.5.1
  – пропорциональный 2.5.1
  – ранговый 2.5.1
  – селективный 2.5.2
  – с остатком 2.5.1
  – стохастический универсальный 2.5.1
  – турнирный 2.5.1
– панмиксии 2.5.2
– последовательного нишевания 2.6
– разделяющей (сепарирующей) функции 2.6
– редукции недопустимых особей к допустимым 2.6
– рулетки 2.5.1
– сетевого оператора 7.3.2
– скользящего допуска 1.3.1
– управления параметрами 7.2.1
  – адаптивный 7.2.1, 7.5
    – коэволюционный
      – гетерогенный 7.5.2
      – гомогенный 7.5.1
  – детерминированный 7.2.1
  – самоадаптивный 7.2.1, 7.6
– штрафов
  – динамических 2.6
  – смертельных 1.3.2, 2.6
  – статических 2.6
Миграция
– перемещением 9.4.2
– репликацией 9.4.2
Микроалгоритм
– генетический 2.7
Модель
– коэволюции
  – простая 6.4.1
  – композитная 6.4.1
– некоммутирующей мульти-популяции 9.4.4
– обучаемой эволюции 2.7
– островная 6.4.1
– параллелизма
  – глобальная 9.4.1
    – асинхронная 9.4.1
    – синхронная 9.4.1
  – диффузная 9.4.3
  – комбинированная 9.4.4
  – коэволюционная 9.4.4
  – островная 9.4.2
    – асинхронная 9.4.2
    – синхронная 9.4.2
  – пространственно вложенная 9.4.4
  – расщеплением 9.4.4
  – типа «хозяин–раб» 9.4.1
Монохромосома 2.2
– модифицированная 2.2
Мультихромосома 2.2
– гомоморфная 2.2
– негомоморфная 2.2
Мутация
– знаковая 5.2
– переупорядочивающая 5.2
Ниша 8.5.1
Нишевание 2.6, 5.4.3, 8.5.1
Оператор
– адаптивный 2.1
– арифметического сдвига
  – вещественный 2.3.2
– ветвления 4.7.4
– вставки 2.4.1
– генетический 2.1
– геометрического сдвига
  – вещественный 2.3.2
– кормления 4.7.3
– макромутации 2.3.1
– миграции 2.11
– мутации
  – бинарный 2.3.1
  – гауссовый 2.3.2, 5.6.3
  – граничный 2.3.2, 5.6.3
  – динамический 2.3.2
  – зеркальный 2.3.1
  – неравномерный
    – генный 2.3.1
    – Михалевича 2.3.2, 5.6.3
  – обменный 2.3.1
  – равномерный
    – генный 2.3.1
    – хромосомный 2.3.1
  – случайный 2.3.2
  – степенной 2.3.2
  – *BGA* 2.3.2

– плавания 4.7.3
– прививки 4.7.4
– разбиения 4.7.3
– репликации 2.10
– сегрегации 2.4.1
– скрещивания 4.7.4
– транслокации бинарный 2.6
– удаления 2.4.1
– ускоряющий 2.11
– транспозиции 2.3.1
Операция
– локальных состязаний 5.4.1
– диссимиляции 5.4.1
  – модифицированная 5.4.2
– конкретизации 5.6.2
– кооперации 5.4.2
– обобщения 5.6.2
– разделения 5.6.2
– распространения 5.6.2
  – обратного 5.6.2
– слияния 5.6.2
– случайного выбора 5.1.1
Оптимизация
– альтернативная 6.3.2
– безусловная 1.1.2
– без ограничений 1.1.2
– глобальная 1.1.2
– детерминированная 1.1.1
– динамическая 3.2.5
– локальная 1.1.2
– многоэкстремальная 1.1.2
– многоцелевая 8.1
– одномерная 1.1.2
– однопараметрическая 1.1.2
– одноэкстремальная 1.1.2
– с ограничениями 1.1.2
  – типа неравенств 1.1.2
  – типа равенств 1.1.2
  – общего вида 1.1.2
– условная 1.1.2
Отображение
– динамическое 9.3
– логистическое 1.4.2
– оптимальное 9.3
– палаточное 1.4.2
– статическое 9.3
Принцип Эджворта – Парето 8.1.1
Программирование
– выпуклое 1.1.2
– геометрическое 1.1.2
– дискретное 1.1.2
– дробно-линейное 1.1.2
– квадратичное 1.1.2
– линейное 1.1.2
– нелинейное 1.1.2
– сепарабельное 1.1.2
– целочисленное 1.1.2
– эволюционное 2.10
Пространство
– убеждений 5.6
– решений 5.9.2
– целевое 8.1.1
Процедура
– локального поиска 1.4.2
– меметической эволюции 4.7.1
– модуляции 3.2.4
– разбиения единичного гиперкуба 1.4.1
– разбиения единичного отрезка 1.4.1
– тасования 4.7.1
– улучшения решения 5.9.1
Решение
– инициализирующее 5.9.2
– многообещающее 1.4.3
– опорное 5.9.1
– целевое 5.9.2
– эталонное 8.4
Сеть
– иммунная 4.1.2
– ограничений 5.6.2
Сжатие
– клональное 4.1.2
– сетевое 4.1.2
Система
– асинхронная 9.1.1
– гетерогенная 9.1.1
– гомогенная 9.1.1
– искусственных иммунных (*AIS*) 4.1
– неоднородная 9.1.1
– однородная 9.1.1
– сильносвязанная 9.1.1
– синхронная 9.1.1
– слабосвязанная 9.1.1
– с памятью
  – гибридной 9.1.1
  – общей 9.1.1
  – разделяемой 9.1.1
  – распределенной 9.1.1
– *MIMD* 9.1.1
– *MISD* 9.1.1
– *SIMD* 9.1.1

Стратегия
– алгоритма 7
– выбора
  – декомпозиционного 5.7.2
  – простого случайного 5.7.2
  – прямого 5.7.2
  – ранжированного 5.7.2
  – рулеточного 5.7.2
  – с запретами 5.7.2
– динамическая 7.1.2
  – апостериорная (неадаптивная) 7.1.2
  – априорная (адаптивная) 7.1.2
– жадная рандомизированная 5.9.2
– миграции 9.4.2
– спуска
  – случайного 5.7.2
  – со случайной перестановкой 5.7.2
– статическая 7.1.1
– структурная 7.1.3
– структурно–параметрическая 7.1.4
– управления
  – интегративная 6.1.3
  – кооперативная 6.1.3
– эволюционная 2.9
Субпопуляция 7.5
Суброй 7.5.1
Суперпопуляция 7.5
Суперрой 7.5.1
Схема взаимодействия субпопуляций
– параллельная 6.4.1
– последовательная 6.4.1
Теорема
– шим 2.8
  – модифицированная 2.8
– Куна – Такера 2.12
Топология соседства
– глобально оптимальная 3.1.3
– двумерный тор 3.1.3
– динамическая 3.1.4
– кластерная 3.1.3
– клика 3.1.3
– кольцо 3.1.3
– локально оптимальная 3.1.3
– статическая 3.1.4
– фон Неймана 3.1.3
Точка
– допустимая 1.3.1
– почти допустимая 1.3.1
– недопустимая 1.3.1
– феромонная 3.2.1
– центральная 8.2.2
Участок
– многообещающий 4.2.3
– отобранный 3.3.2
– перспективный 3.3.2
– элитный 3.3.2
Фаза
– отталкивания 3.1
– положительного конфликта 3.1.2
– притяжения 3.1
– промежуточная 3.1.2
Фенотип 2.1
Фронт Парето 8.1.1
Функция
– аспирации 1.4.3
– влияния 5.6.1
– временная 3.2.5
– выбора 3.2.5
– выпуклая 1.1.1
– динамическая 8.3.2
– Лагранжа 2.12
– многоэкстремальная 1.1.1
– мультимодальная 1.1.1
– овражная 1.1.1
– отбора 2.5.2
– подобия 8.5.1
– позиномиальная 1.1.1
– предпочтений 8.4.2
  – согласованная 8.4.2
  – строго согласованная 8.4.2
– принятия 5.6.1
– разделяющая 8.5.1
– сепарабельная 1.1.1
– сравнения 8.4.2
– строго выпуклая 1.1.1
– структурная 3.2.5
– целевая 1.1.1
  – динамическая 3.2.5
  – частная 8.1.1
– штрафа 1.3.1
Частица
– неправильная 6.4.2
– правильная 6.4.2
Шима 2.8
– определяющая длина 2.8
– порядок 2.8
– приспособленность 2.8
Эвристика
– полужадная 1.4.2
– случайная жадная 1.4.2
Экзон 7.6.1
Элитизм 8.3.2

ABC, Artificial Bee Colony 3.3.3
ABFO, Adaptive Bacterial Foraging Optimization 4.2.3
ACO, Ant Colony Optimization 3.2
ACO_R, extended Ant Colony Optimization to continuous domains 3.2
ADD, Approximate Descent Direction 1.2.1
AIS, Artificial Immune Systems 4.1
Altering objective functions algorithm 8.3.2
AMRS, Adaptive Maxing Random Search 1.2.2
ARPSO, Attractive and Repulsive PSO 3.1.2
ARSID, Adaptive Random Search with Intensification and Diversification 1.2.1
AS, Ant System 3.2
ASSRS, Adaptive Step Size Random Search 1.2.1
AWS, Adaptive Weighted Sum algorithm 8.2.2
BA, B-algorithm, Bees algorithm 3.3.2
BAS, Binary Ant System 3.2
BFO, Bacterial Foraging Optimization 4.2.2
BFO-GA 4.2.5
BFO-PSO 4.2.5
BI, Bat-Inspired algorithm 4.7.2
C-algorithm, Cultural algorithm 5.6
CACO, Continuous Ant Colony Optimization 3.2.2
CACS, Continuous Ant Colony System 3.2
CBFO, Cooperative BFO algorithm 4.2.3
CBFO-H, Cooperative BFO algorithm Hybrid 4.2.3
CBFO-S, Cooperative BFO algorithm Sequence 4.2.3
CBPSO, Clubs-Based PSO 3.1.4
CFO, Central Force Optimization 5.2
C-GRASP 1.4.2
CIAC, Continuous Interaction Ant Colony 3.2.3
CHC, Cross-population selection, Heterogenous recombination and Cataclysmic mutation 2.7
CLONALG, Clonal Selection Algorithm 4.1.3
CLS, Chaotic Local Search 1.4.2
CMEC, Chaotic MEC 5.4.4
COAC, Continuous Orthogonal Ant Colony 3.2.4
CS, Cuckoo Search 4.5
CSS, Charged System Search 5.3.2
CUDA, Compute Unified Device Architecture 9.1.2
DACO, Direct Application of Ant Colony Optimization 3.2
DFSS, Density FSS 4.7.3
DHCIAC, Dynamic Hybrid Continuous Interacting Ant Colony 3.2.5
DHCIACfind 3.2.5
DHCIACtrack 3.2.5
DHS, Differential HS algorithm 5.1.2
DSPSO, Dynamic Sociometry PSO 3.1.4
Dynamic neighborhood Algorithm 8.3.2
EM, ElectroMagnetism-like algorithm 5.3.1
EMEC, Extended Mind Evolutionary Computation 5.4.2
F-algorithm, Firefly algorithm 4.3
FDR, Fitness-Distance Ratio based PSO 3.1.2
FFGA, Fonseca and Fleming's multi-objective Genetic Algorithm 8.3.3
FIPS, Fully Informed Particle Swarm 3.1.2
FO, Fly Optimization 4.7
FSS, Fish School Search 4.7.3
FSSRS , Fixed Step Size Random Search 1.2.1
Genitor 2.7
GCA, Genetic Coevolutionary Algorithm 2.12
GENOCOP, GEnetic algorithm for Numerical Optimization for COnstrained Problem 5.6.3
GHS, Global HS 5.1.2
GIPSO, GPU Island PSO 8.4.1
GPU, Graphics Processing Unit 9.1.2
GRASP, Greedy Randomized Adaptive Search Procedure 1.4.2
GS, Gravitational Search algorithm 5.2
GSO, Glowworm Swarm Optimization 4.3.2
HCIAC, Hybrid Continuous Interacting Ant Colony 3.2.5
HMCR, Harmony Memory Considering Rate 5.1.1

HPS, Heuristic Pattern Search 1.2.1
HS, Harmony Search algorithm 5.1
IBA, IB algorithm, Improved Bee algorithm 3.3.2
ICS, Improved Cuckoo Search 4.5
IHS, Improved HS 5.1.2
ILS, Iterated Local Search 1.4.2
IMEC, Improved MEC 5.4.3
IWD, Intelligent Water Drop algorithm 5
IWO, Invasive Weed Optimization 4.4
Lévy flights 1.4.2
LUS, Local Unimodal Sampling 1.2.1
M-algorithms, Memetic algorithms 5.7
MCS, Modified Cuckoo Search 4.5
MEC, Mind Evolutionary Computation 5.4
Micro-AIS, Micro Artificial Immune System 4.1.3
MOEA/D, Multi-Objective Evolutionary Algorithm based on Decomposition 8.1.2
MOGA, Multi-Objective Genetic Algorithm 8.3.3
Monkey algorithm 4.6.2
MOPSO, Multi-Objective Particle Swarm Optimization 8.6.2, 9.5.2
MPI, Message Passing Interface 9.1.2
MPP, Massive Parallel Processing 9.1.1
MS, Monkey Search 4.6.1
MS MOPSO, Multi-Swarm MOPSO 8.6.2
NDS, Non-Dominated Sorting 8.3.3
NPGA, Niched-Pareto Genetic Algorithm 8.3.3
NSGA, Non-dominated Sorting Genetic Algorithm 8.3.3
NSGA II, Non-dominated Sorting Genetic Algorithm II 8.3.3
NUMA, Non-Uniform Memory Access 9.1.1
OpenCL, Open Computing Language 9.1.2
OpenMP, Open Multi-Processing 9.1.2
PAES, Pareto Achieved Evolution Strategy 8.3.3
PR, Path Relinking algorithm 5.9.2
Predator-prey algorithm 8.3.2
PSO, Particle Swarm Optimization 3.1
PSO with stereotyping 3.1.4
RFD, River Formation Dynamics 5
RFIPS, Ranked FIPS 3.1.2
RGA, Real-coded Genetic Algorithm 2.2
RLS, Random Line Search 1.2.1
RS, Random Search 1.2.1
RTS, Restricted Tournament Selection 8.5.2
RVNS, Random VNS 1.4.2
SA, Simulated Annealing 1.4.2
SDPS, Single-Dimension Perturbation Search 1.2.1
SDS, Stochastic Diffusion Search 5.5
SFL, Shuffled Frog-Leaping 4.7.1
Sigma algorithm 8.3.4
SMP, Symmetric Multi-Processing 9.1.1
SMEC, Simple MEC, Simple Mind Evolutionary Computation 5.4.1
SOM, Self Organizing Migrating algorithm 5.8
SPEA, Strength Pareto Evolutionary Algorithm 8.3.3
SPEA 2, Strength Pareto Evolutionary Algorithm 2 8.3.3
SPSA, Simultaneous Perturbation Stochastic Approximation 1.2.1
SS, Scatter Search algorithm 5.9.1
SSG, Saplings Sowing and Growing up 4.7.4
SWGR, Sum of Weighted Global Ratios 8.3.3
SWO, Sum of Weighted Objectives 8.2.2
SWR, Sum of Weighted Ratios 8.3.3
TS, TS algorithm, Tabu Search algorithm 1.4.3
UMA, Unified Memory Access 9.1.1
VEGA, Vector Evaluated Genetic Algorithm 8.3.2
VNS algorithm, Variable Neighborhood Search algorithm 1.4.2
VPSO, Volitive PSO 4.7.3
WAR, Weighted Average Ranking 8.3.3
WFIPS, Weighted FIPS 3.1.2
WMR, Weighted Maximum Ranking 8.3.3

Оглавление

Предисловие ... 3
Основные обозначения ... 5
Введение ... 8
Глава 1. Постановка задачи поисковой оптимизации и непопуляционные стохастические алгоритмы ее решения ... 14
1.1. Постановка и классификация алгоритмов решения детерминированной задачи поисковой оптимизации ... 14
1.1.1. Постановка задачи ... 14
1.1.2. Классификация задач оптимизации ... 16
1.1.3. Классификация алгоритмов оптимизации ... 19
1.2. Локальная безусловная оптимизация ... 22
1.2.1. Одношаговые алгоритмы ... 22
1.2.2. Многошаговые алгоритмы ... 26
1.2.3. Многоточечные алгоритмы ... 29
1.3. Локальная условная оптимизация ... 33
1.3.1. Алгоритмы на основе сведения задачи условной оптимизации к задаче безусловной оптимизации ... 34
1.3.2. Алгоритмы, не использующие редукцию задачи условной оптимизации к задаче безусловной оптимизации ... 42
1.4. Глобальная оптимизация ... 46
1.4.1. Алгоритмы одномерной редукции ... 46
1.4.2. Алгоритмы случайного поиска ... 53
1.4.3. Поиск с запретами ... 62
Вопросы для самопроверки ... 63
Глава 2. Эволюционные алгоритмы ... 65
2.1. Биологические предпосылки и общая схема эволюционных алгоритмов ... 66
2.2. Кодирование особей ... 69
2.3. Операторы мутации ... 75
2.3.1. Бинарные мутаторы ... 75
2.3.2. Вещественные мутаторы ... 77
2.4. Операторы скрещивания (кроссоверы) ... 80
2.4.1. Бинарные кроссоверы ... 80
2.4.2. Вещественные кроссоверы ... 84
2.5. Операторы отбора ... 88
2.5.1. Операторы управления популяцией ... 88
2.5.2. Операторы селекции ... 92
2.6. Другие операторы и процедуры ... 96
2.7. Типовые генетические алгоритмы ... 102
2.8. Теория шим ... 106
2.9. Эволюционная стратегия ... 109
2.10. Эволюционное программирование ... 112
2.11. Дифференциальная эволюция ... 114
2.12. Генетический коэволюционный алгоритм ... 117
2.13. Пример применения генетического алгоритма ... 121
Вопросы для самопроверки ... 125

Глава 3. Алгоритмы роя частиц, колонии муравьев и пчелиного роя 127
3.1. Оптимизация роем частиц 127
3.1.1. Канонический алгоритм роя частиц 127
3.1.2. Модификации канонического алгоритма роя частиц 133
3.1.3. Топологии соседства частиц 137
3.1.4. Алгоритмы с динамической топологией соседства частиц 141
3.1.5. Гибридный алгоритм на основе роя частиц и имитации отжига 143
3.1.6. Пример решения задачи с использованием алгоритма роя частиц 146
3.2. Муравьиная оптимизация 147
3.2.1. Бионические предпосылки 148
3.2.2. Алгоритм непрерывной оптимизации колонией муравьев 150
3.2.3. Алгоритм непрерывно взаимодействующей колонии муравьев 153
3.2.4. Непрерывный ортогональный алгоритм муравьиной колонии 159
3.2.5. Гибридный алгоритм непрерывно взаимодействующей муравьиной колонии 163
3.2.6. Пример применения модифицированного алгоритма *CIAC* 170
3.3. Оптимизация пчелиным роем 175
3.3.1. Бионические предпосылки 175
3.3.2. Пчелиный алгоритм 176
3.3.3. Алгоритм колонии искусственных пчел 183
3.3.4. Гибридный алгоритм 190
Вопросы для самопроверки 193

Глава 4. Другие популяционные алгоритмы, вдохновленные живой природой 195
4.1. Искусственные иммунные системы 195
4.1.1. Биологические основы 195
4.1.2. Оптимизация с помощью модели иммунной сети 198
4.1.3. Алгоритм на основе искусственной микроиммунной системы 200
4.1.4. Пример алгоритма искусственной иммунной сети 201
4.2. Бактериальная оптимизация 204
4.2.1. Бионические предпосылки 204
4.2.2. Канонический алгоритм бактериальной оптимизации 205
4.2.3. Кооперативная бактериальная оптимизация 210
4.2.4. Алгоритм, использующий эффект роения бактерий 211
4.2.5. Гибридные бактериальные алгоритмы 212
4.3. Алгоритмы, вдохновленные роем светлячков 213
4.3.1 Алгоритм светлячков 213
4.3.2 Алгоритм оптимизации роем светлячков 216
4.4. Сорняковый алгоритм 218
4.4.1. Биологические основы 218
4.4.2. Схема алгоритма 219
4.5. Кукушкин поиск 221
4.5.1. Биологические предпосылки 221
4.5.2. Схема алгоритма 222
4.5.3. Эффективность алгоритма 223
4.6. Алгоритмы, вдохновленные поведением обезьян 227
4.6.1. Обезьяний поиск 227
4.6.2. Обезьяний алгоритм 228
4.7. Прочие алгоритмы 230

4.7.1 Тасующий алгоритм прыгающих лягушек ... 230
4.7.2. Алгоритм, инспирированный летучими мышами ... 231
4.7.3. Поиск косяком рыб ... 234
4.7.4. Алгоритм растущих деревьев ... 237
Вопросы для самопроверки ... 238
Глава 5. Популяционные алгоритмы, инспирированные неживой природой, человеческим обществом, и другие популяционные алгоритмы ... 239
5.1. Гармонический поиск ... 239
5.1.1. Канонический алгоритм ... 240
5.1.2. Некоторые модификации алгоритма гармонического поиска ... 242
5.1.3. Пример применения модифицированного алгоритма гармонического поиска ... 243
5.2. Алгоритм гравитационного поиска ... 246
5.3. Электромагнитный поиск ... 250
5.3.1. Электромагнитный алгоритм ... 250
5.3.2. Алгоритм поиска системой зарядов ... 254
5.4. Алгоритм эволюции разума ... 255
5.4.1. Простой алгоритм эволюции разума ... 256
5.4.2. Расширенный алгоритм ... 257
5.4.3. Улучшенный алгоритм ... 259
5.4.4. Хаотический алгоритм эволюции разума ... 260
5.5. Стохастический диффузионный поиск ... 261
5.6. Культурный алгоритм ... 263
5.6.1. Общая схема алгоритма ... 263
5.6.2. Построение и коррекция пространства убеждений ... 265
5.6.3. Культурная модификация генетических операторов ... 267
5.7. Меметические алгоритмы ... 268
5.7.1 Общая схема алгоритма ... 268
5.7.2. Гиперэвристические мультимемные адаптивные алгоритмы ... 269
5.7.3. Самоадаптирующиеся мультимемные алгоритмы ... 271
5.8. Самоорганизующийся миграционный алгоритм ... 272
5.9. Алгоритмы рассеянного поиска и прокладки путей ... 274
5.9.1. Алгоритм рассеянного поиска ... 274
5.9.2. Алгоритм прокладки путей ... 278
Вопросы для самопроверки ... 279

Глава 6. Гибридизация популяционных алгоритмов ... 280
6.1. Общие принципы гибридизации ... 280
6.1.1. Одноуровневая классификация Ванга ... 280
6.1.2. Двухуровневая классификация Эль-Абда и Камэла ... 282
6.1.3. Четырехуровневая классификация Рейдла ... 283
6.2. Вложенные алгоритмы ... 285
6.2.1. Высокоуровневая гибридизация вложением ... 285
6.2.2. Низкоуровневая гибридизация вложением ... 287
6.3. Гибридизация по схеме препроцессор / постпроцессор ... 288
6.3.1. Последовательная гибридизация ... 288
6.3.2. Конвейерная гибридизация ... 290
6.4. Коалгоритмы ... 296
6.4.1. Классификация коэволюционных алгоритмов ... 296
6.4.2. Пример коалгоритма ... 299
Вопросы для самопроверки ... 301

Глава 7. Метаоптимизация популяционных алгоритмов ... 302
7.1. Постановка метазадачи оптимизации ... 302
7.1.1. Статическая параметрическая метаоптимизация ... 302
7.1.2. Динамическая параметрическая метаоптимизация ... 304
7.1.3. Структурная метаоптимизация ... 307
7.1.4. Структурно-параметрическая метаоптимизация ... 308
7.2. Классификация методов метаоптимизации ... 309
7.2.1. Классификация Эйбена ... 309
7.2.2. Классификация Смита – Фогарти ... 312
7.3. Однократная настройка параметров ... 313
7.3.1. Статическая параметрическая метаоптимизация ... 313
7.3.2. Динамическая параметрическая метаоптимизация ... 315
7.4. Перманентная настройка параметров ... 316
7.4.1. Схема подхода ... 316
7.4.2. Задачи кластеризации и собственно метаоптимизации ... 317
7.4.3. Особенности программной реализации подхода ... 318
7.4.4. Исследование эффективности подхода ... 320
7.5. Адаптивное управление параметрами ... 324
7.5.1. Гомогенные методы ... 325
7.5.2. Гетерогенные методы ... 326
7.6. Самоадаптивное управление параметрами ... 327
7.6.1. Расширение вектора варьируемых параметров базовой задачи ... 327
7.6.2. Коэволюция типа соперничества ... 328
7.7. Структурная метаоптимизация ... 328
Вопросы для самопроверки ... 330

Глава 8. Популяционные алгоритмы многоцелевой оптимизации ... 332
8.1. Задача многоцелевой оптимизации (МЦО-задача) и алгоритмы ее решения ... 332
8.1.1. Постановка задачи ... 333
8.1.2. Классификация алгоритмов решения МЦО-задачи ... 336
8.2. Непопуляционные алгоритмы Парето-аппроксимации ... 340
8.2.1. Сеточные алгоритмы ... 340
8.2.2. Алгоритмы на основе свертки целевых функций ... 341
8.3. Популяционные алгоритмы Парето-аппроксимации ... 345
8.3.1. Лексикографическая турнирная селекция ... 345
8.3.2. Алгоритмы чередующихся целевых функций ... 346
8.3.3. Алгоритмы на основе ранжирования агентов ... 349
8.3.4. Алгоритмы, не использующие ранжирование агентов ... 353
8.4. Критерии оценки качества Парето-аппроксимации ... 357
8.4.1. Унарные критерии ... 358
8.4.2. Бинарные критерии ... 360
8.5. Методы обеспечения качества Парето-аппроксимации ... 365
8.5.1. Нишевание ... 366
8.5.2. Другие методы ... 367
8.6. Примеры Парето-аппроксимации ... 370
8.6.1. Парето-аппроксимация на основе генетического алгоритма ... 370
8.6.2. Аппроксимация с помощью алгоритма роя частиц ... 373
8.6.3. Аппроксимация на основе алгоритма колонии муравьев ... 379
8.6.4. Аппроксимация на основе пчелиного алгоритма ... 380
Вопросы для самопроверки ... 383

Глава 9. Параллельные популяционные алгоритмы поисковой оптимизации ... 384
9.1. Классификация и основные типы параллельных ЭВМ ... 384
9.1.1. Классификация параллельных вычислительных систем ... 385
9.1.2. Основные типы параллельных ЭВМ ... 389
9.2. Балансировки загрузки параллельной ЭВМ ... 393
9.2.1. Задача оптимального отображения алгоритма на архитектуру параллельной ЭВМ ... 393
9.2.2. Постановка задачи балансировки загрузки ... 395
9.2.3. Методы статической балансировки загрузки ... 396
9.2.4. Методы динамической балансировки загрузки ... 400
9.2.5. Задача согласования алгоритма с архитектурой параллельной ЭВМ ... 403
9.3. Методы распараллеливания популяционных алгоритмов оптимизации ... 404
9.3.1. Глобальная модель параллелизма ... 405
9.3.2. Островная модель параллелизма ... 407
9.3.3. Диффузная модель параллелизма ... 409
9.3.4. Другие модели параллелизма ... 409
9.4. Примеры параллельного решения задач оптимизации ... 411
9.4.1. Параллельный алгоритм роя частиц *GIPSO* ... 411
9.4.2. Параллельный алгоритм Парето-аппроксимации роем частиц ... 417
Вопросы для самопроверки ... 419

Литература ... 420
Приложение А. История разработки популяционных алгоритмов поисковой оптимизации ... 422
Приложение Б. Тестовые функции для одноцелевой задачи глобальной оптимизации ... 424
Приложение В. Тестовые задачи многоцелевой оптимизации ... 432
Предметный указатель ... 434

Учебное издание

Карпенко Анатолий Павлович

СОВРЕМЕННЫЕ АЛГОРИТМЫ ПОИСКОВОЙ ОПТИМИЗАЦИИ

Алгоритмы, вдохновленные природой

Редактор *Л.Н. Лащёва*
Технический редактор *Э.А. Кулакова*
Художник *А.К. Ездовой*
Корректор *А.К. Еникеева*
Компьютерная графика *Ю.В. Мосягина*
Компьютерная верстка *Ю.В. Мосягина*

В оформлении использованы шрифты Студии Артемия Лебедева.

Оригинал-макет подготовлен в Издательстве МГТУ им. Н.Э. Баумана.

Подписано в печать 15.01.2017. Формат 70×100 1/16.
Усл. печ. л. 36,4. Тираж 500 экз. Заказ №

Издательство МГТУ им. Н.Э. Баумана.
105005, Москва, 2-я Бауманская ул., д. 5, стр. 1.
press@bmstu.ru; www.baumanpress.ru

Отпечатано в ПАО «Т8 Издательские Технологии»
109316, Москва, Волгоградский проспект, д. 42, корп. 5.