Парадигма развития науки

Методологическое обеспечение

А. Е. Кононюк

ДИСКРЕТНО-НЕПРЕРЫВНАЯ МАТЕМАТИКА

Книга 10

Алгоритмы

Часть 3

Генетические алгоритмы

Киев
«Освіта України»
2017

УДК 51 (075.8)
ББК В161.я7
К213

Рецензенты:

В. В. Довгай — к-т физ.-мат. наук, доц. (Национальный тех— нический университет «КПІ»);

В. В. Гавриленко — д-р физ.-мат. наук, проф.,

О. П. Будя — к-т техн. наук, доц. (Киевский университет эко— номики, туризма и права);

Н. К. Печурин — д-р техн. наук, проф. (Национальный ави— ационный университет).

Кононюк А. Е.
К213 Дискретно-непрерывная математика. (Алгоритмы). —В 12-и кн. Кн. 10,Ч.3— К.: 2017. —444 с.

ISBN 978-966-373-693-8 (многотомное издание)
ISBN 978-966-373-694-12 (книга 10, ч.3)

Многотомная работа содержит систематическое изложение математических дисциплин, исспользуемых при моделировании и исследованиях математических моделей систем.

В работе излагаются основы теории множеств, отношений, поверхностей, пространств, алгебраических систем, матриц, графов, математической логики, теории вероятностей и массового обслуживания, теории формальных грамматик и автоматов, теории алгоритмов, которые в совокупности образуют единную методолгически взамосвязанную математическую систему «Дискретно-непрерывная математика».

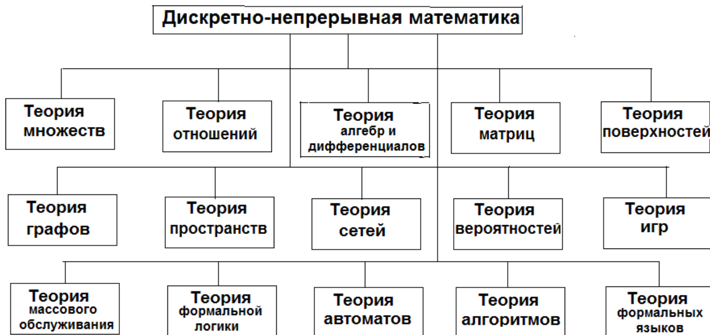
Для бакалавров, специалистов, магистров, аспирантов, докторантов и просто ученых и специалистов всех специальностей.

УДК 51 (075.8)
ББК В161.я7

ISBN 978-966-373-693-8 (многотомное издание)
ISBN 978-966-373-694-12 (книга 10, ч. 3)
© Кононюк А. Е., 2017
© Освіта України, 2017

Кононюк Анатолий Ефимович

Структурная схема развивающейся панмедийной системы наук
I. Группа междисциплинарных наук
ПН Теория понятий
ЛГ Теория лингвистики
ПЗ Теория познания
СЗ Теория созидания
ПТ Общая теория понятоведения
II. Группа наук познания (фундаментальных наук)
ИН Теория информации
СТ Теория систем
РС Теория распознавания
КС Теория консалтинга (консульти-рования)
ПР Теория принятия решения
РР Теория реализации принятых решений
III. Группа наук созидания (прикладных наук)
СС Теория социальных систем
ЖО Теория жизнеобеспечения
КМ Теория коммуникаций и транспортирования
ОБ Теория образования (обучения)
СР Теория сохранения (защиты и охраны)
РЗ Теория развития
IV. Группа общедисциплинарных наук (обеспечивающих наук)
ДМ Теория дискретно-непрерывной математики (четкая и не-четкая)
НИ Теория научных исследований
ММ Теория моделей и модели-рования
ОМ Теория оптимизации


Структура
открытой развивающейся панмедийной системы математических наук (дисциплин)
"Дискретно-непрерывная математика"
Дискретно-непрерывная математика
Теория множеств
Теория отношений
Теория алгебр и дифференциалов
Теория матриц
Теория поверхностей
Теория графов
Теория пространств
Теория сетей
Теория вероятностей
Теория игр
Теория массового обслуживания
Теория формальной логики
Теория автоматов
Теория алгоритмов
Теория формальных языков

Оглавление

Введение............. 8
1. Основные положения... 9
1.1. Базовые понятия................................. 9
1.2. Природный механизм................................. 14
1.3. Особенности ГА................................. 23
1.4. Задачи оптимизации и применение алгоритмов............................27
1.5. Мягкие вычисления 29
1.6. Эволюционные вычисления................................. 42
1.7. Описание генетического алгоритма...57
2. Классический генетический алгоритм и его релизация.....................60
2.1. Функция приспособленности и кодирование решений.................60
2.2. Классический генетический алгоритм..65
2.3. Принцип и алгоритм работы ГА...83
2.4. Применение генетических алгоритмов...88
2.5. Пример выполнения классического генетического алгоритма......91
2.6. Представление данных в генах..96
2.7. Примеры кодирования параметров задачи в генетическом алгоритме..... 99
3. Основы теории ГА.......... 105
3.1. Шаблоны...... 105
3.2. Настройка ГА................................. 111
3.3. Другие модели ГА................................. 113
3.4. Некоторые модели генетических алгоритмов...............................117
3.5. Параллельные ГА................................. 119
3.6. Наблюдения................ 122
3.7. Основная теорема о генетических алгоритмах.............................125
3.8. Строительные блоки (Building blocks) 143
3.9. Модификации классического генетического алгоритма.............146
3.9.1. Методы селекции................................. 147
3.9.2. Особые процедуры репродукции...151
3.9.3. Генетические операторы...151
3.9.4. Методы кодирования................................. 154
3.9.5. Масштабирование функции приспособленности.......................155
3.9.6. Ниши в генетическом алгоритме...157
3.9.7. Генетические алгоритмы для многокритериальной оптимизации................................. 159

3.9.8. Генетические микроалгоритмы..161
4. Модели генетических алгоритмов 163
4.1. Модели генетических алгоритмов и стратегии отбара и формирования нового поколения................ 163
4.2. Проверка эффективности ГА с использованием тестовых функций.................................. 169
4.3. Примеры оптимизации функции с помощью программы FlexTool 173
4.4. Генетические алгоритмы и математический аппарат...................211
5. Эволюционное моделирование.................................. 226
5.1. Эволюционные алгоритмы...226
5.2. Приложения эволюционных алгоритмов.......................................234
5.2.1. Примеры оптимизации функции с помощью программы Evolver.................................. 235
5.2.2. Решение комбинаторных задач с помощью программы Evolver.................................. 267
5.3. Эволюционные алгоритмы в нейронных сетях.............................273
5.3.1. Независимое применение генетических алгоритмов и нейронных сетей.................................. 275
5.3.2. Нейронные сети для поддержки генетических алгоритмов......276
5.3.3. Генетические алгоритмы для поддержки нейронных сетей......277
5.3.4. Применение генетических алгоритмов для обучения нейронных сетей.................................. 280
5.3.5. Генетические алгоритмы для выбора топологии нейронных сетей...281
5.3.6. Адаптивные взаимодействующие системы..............................282
5.3.7. Типовой цикл эволюции..282
5.3.7.1. Эволюция весов связей...284
5.3.7.2. Эволюция архитектуры сети...288
5.3.7.3. Эволюция правил обучения..292
5.4. Примеры моделирования эволюционных алгоритмов в приложении к нейронным сетям..295
5.4.1. Программы Evolver и BrainMaker........ 295
5.4.2. Программа GTO6. Применение генетических алгоритмов........326
6. Применение генетических алгоритмов 332
6.1. Применение ГА для автоматической генерации тестов............... 332
6.2. Генетические алгоритмы, распознающие изображений............... 344
6.3. Генетические алгоритмы в MATLAB 351
6.3.1. Суть генетических алгоритмов...351

6.3.2. Работа с GENETIC ALGORITHM TOOL....................................352
6.4. Аппроксимация изображений генетическим алгоритмом при помощи EvoJ................................ 361
6.4.1. Выбор способа описания решения..361
6.4.2. Кофигурирование EvoJ при помощи Detached Annotations........366
6.4.3. Создание фитнес функции...368
6.4.4. Осуществление итераций.................................... 369
6.4.5. Улучшение алгоритма................................... 371
6.5. Разработка и исследование гибридного алгоритма решения сложных задач оптимизации.. 372
6.5.1. Генетический алгоритм (ГА) 372
6.6. Примеры генетического алгоритма..379
6.6.1. Масштаб пригодности...379
6.6.2. Сопоставление ранга и Масштабирования высшего уровня......381
6.6.3. Селекция................................. 383
6.6.4. Опции репродуцирования. 384
6.6.5. Мутация и кроссовер...385
6.6.6. Установка числа мутаций.................................. 386
6.6.7. Установка кроссоверной доли...389
6.6.8. Установка без Мутаций..390
6.6.9. Мутации без кроссовера...392
6.6.10. Сравнение результатов с фракциями измененных после операции кроссовера.................................. 393
6.6.11. Пример – сравнение глобального и локального минимумов....394
6.6.12. Пример решения данной задачи с помощью Генетического алгоритма................................. 396
6.6.13. Использование гибридной функции...400
6.6.14. Установка максимального числа поколений............................403
6.6.15. Векторизация функции пригодности..405
6.6.16. Минимизация при наличии ограничений с использованием функции ga.................................. 406
6.6.17. Параметризация функций, вызываемых с помощью ga..........415
6.6.18. Параметризация функций при помощи анонимных функциq для ga 416
6.6.19. Параметризация функции при помощи вложенной функции для ga.................................. 417
Приложение. Практическая часть реализации генетических алгоритмов..419
Литература................................. 441

Введение

Эволюция в природе показала себя как мощный механизм развития и приспособления живых организмов к окружающей среде и удивляет многообразием и эффективностью решений. Поэтому исследователи в области компьютерных технологий обратились к природе в поисках новых алгоритмов.

Группа алгоритмов, использующих в своей основе идею эволюции Дарвина, называется *эволюционными алгоритмами*. В ней выделяют следующие направления.

- Генетические алгоритмы (ГА).
- Эволюционные стратегии.
- Генетическое программирование.
- Эволюционное программирование.

Генети́ческий алгори́тм (англ. *genetic algorithm*) — это эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путём случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, аналогичных естественному отбору в природе. Является разновидностью эволюционных вычислений, с помощью которых решаются оптимизационные задачи с использованием методов естественной эволюции, таких как наследование, мутации, отбор и кроссинговер. Отличительной особенностью генетического алгоритма является акцент на использование оператора «скрещивания», который производит операцию рекомбинации решений-кандидатов, роль которой аналогична роли скрещивания в живой природе.

Генетические алгоритмы применяются для решения таких задач, как:

- поиск глобального экстремума многопараметрической функции,
- аппроксимация функций,
- задачи о кратчайшем пути,

- задачи размещения,
- настройка искусственной нейронной сети,
- игровые стратегии,
- обучение машин.

Фактически, генетические алгоритмы максимизируют многопараметрические функции. Поэтому их область применения столь широка. Все приведенные задачи решаются именно путем формирования функции, зависящей от некоторого числа параметров, глобальный максимум которой будет соответствовать решению задачи.

1. Основные положения

1.1. Базовые понятия

Генети́ческий алгори́тм (англ. *genetic algorithm*) — это эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путем последовательного подбора, комбинирования и вариации искомых параметров с использованием механизмов, напоминающих биологическую эволюцию. Является разновидностью эволюционных вычислений (англ. *evolutionary computation*). Отличительной особенностью генетического алгоритма является акцент на использование оператора «скрещивания», который производит операцию рекомбинации решений-кандидатов, роль которой аналогична роли скрещивания в живой природе. «Отцом-основателем» генетических алгоритмов считается Джон Холланд (англ. *John Holland*), книга которого «Адаптация в естественных и искусственных системах» (англ. *Adaptation in Natural and Artificial Systems*) является основополагающим трудом в этой области исследований.

При описании генетических алгоритмов используются определения, заимствованные из генетики. Например, речь идет о *популяции особей,* а в качестве базовых понятий применяются *ген, хромосома, генотип, фенотип, аллель.* Также используются соответствующие этим терминам определения из технического лексикона, в частности, *цепь, двоичная последовательность, структура.*
Популяция - это конечное множество *особей.*

Особи, входящие в популяцию, в генетических алгоритмах представляются хромосомами с закодированным в них множествами *параметров задачи,* т.е. решений, которые иначе называются *точками в пространстве поиска (search points).* В некоторых работах особи называются *организмами.*

Хромосомы (другие названия - *цепочки* или *кодовые последовательности)* - это упорядоченные *последовательности генов.*

Ген (также называемый *свойством, знаком* или *детектором)* - это атомарный элемент *генотипа,* в частности, хромосомы.

Генотип или ***структура*** - это набор хромосом данной особи. Следовательно, особями популяции могут быть генотипы либо единичные хромосомы (в довольно распространенном случае, когда генотип состоит из одной хромосомы).

Фенотип - это набор значений, соответствующих данному генотипу, т.е. *декодированная структура* или множество *параметров задачи {решение, точка пространства поиска).*

Аллель - это значение конкретного гена, также определяемое как *значение свойства* или *вариант свойства.*

Локус или *позиция* указывает место размещения данного гена в хромосоме (цепочке). Множество позиций генов - это *локи.*

Очень важным понятием в генетических алгоритмах считается *функция приспособленности (fitness function),* иначе называемая *функцией оценки.* Она представляет меру приспособленности данной особи в популяции. Эта функция играет важнейшую роль, поскольку позволяет оценить степень приспособленности конкретных особей в популяции и выбрать из них наиболее приспособленные (т.е. имеющие наибольшие значения функции приспособленности) в соответствии с эволюционным принципом выживания «сильнейших» (лучше всего приспособившихся). Функция приспособленности также получила свое название непосредственно из генетики. Она оказывает сильное влияние на функционирование генетических алгоритмов и должна иметь точное и корректное определение. В задачах оптимизации функция приспособленности, как правило, оптимизируется (точнее говоря, максимизируется) и называется *целевой функцией.* В задачах минимизации целевая функция преобразуется, и проблема сводится к максимизации. В теории управления функция приспособленности может принимать вид *функции погрешности,* а в теории игр - *стоимостной функции.* На каждой итерации генетического алгоритма приспособленность каждой особи данной популяции оценивается при помощи функции приспособленности, и на этой основе создается

следующая популяция особей, составляющих множество потенциальных решений проблемы, например, задачи оптимизации.
Очередная популяция в генетическом алгоритме называется *поколением,* а к вновь создаваемой популяции особей применяется термин «новое поколение» или «поколение потомков».

Пример 1.1

Рассмотрим функцию

$$f(x) = 2x^2+1 \quad (1.1)$$

и допустим, что x принимает целые значения из интервала от 0 до 15. Задача оптимизации этой функции заключается в перемещении по пространству, состоящему из 16 точек со значениями 0, 1.....15 для обнаружения той точки, в которой функция принимает максимальное (или минимальное) значение.
В этом случае в качестве *параметра задачи* выступает переменная x. Множество {0, 1.....15} составляет *пространство поиска* и одновременно - множество потенциальных решений задачи. Каждое из 16 чисел, принадлежащих к этому множеству, называется *точкой пространства поиска, решением, значением параметра, фенотипом.* Следует отметить, что решение, оптимизирующее функцию, называется *наилучшим* или *оптимальным* решением. Значения параметра х от 0 до 15 можно закодировать следующим образом:
0000 0001 0010 0011 0100 0101 0110 0111
1000 1001 1010 1011 1100 1101 1110 1111
Это широко известный способ двоичного кодирования, связанный с записью десятичных цифр в двоичной системе. Представленные *кодовые последовательности* также называются *цепями* или *хромосомами.* В рассматриваемом примере они выступают и в роли *генотипов.* Каждая из хромосом состоит из 4 генов (иначе можно сказать, что двоичные последовательности состоят из 4 битов). Значение гена в конкретной позиции называется аллелью, принимающей в данном случае значения 0 или 1. *Популяция* состоит из *особей,* выбираемых среди этих 16 *хромосом.* Примером популяции с численностью, равной 6, может быть, например, множество хромосом {0010, 0101, 0111, 1001, 1100, 1110}, представляющих собой закодированную форму следующих фенотипов: {2, 5, 7, 9, 12, 14}. *Функция приспособленности* в этом примере задается выражением (1.1). Приспособленность отдельных хромосом в популяции определяется значением этой функции для значений х, соответствующих этим хромосомам, т.е. для *фенотипов,* соответствующих определенным *генотипам.*

Пример 1.2

Рассмотрим следующий пример постановки оптимизационной задачи. Для системы, изображенной на рис. 1.1, следует найти

$$\min_{k_1,k_2} J = \int_0^T f(y_1, y_2, u)dt$$

где $k_1, k_2 \in [k_{\min}, k_{\max}]$.

В роли параметров этой задачи выступают k_1, и k_2. Пространство поиска должно содержать конечное количество точек, которые можно закодировать в виде хромосом. Параметры k, и k_2 дискретизированы; множество их значений во всем диапазоне от минимального $k_{\min}$ до максимального $k_{\max}$ отображаются на соответствующие двоичные кодовые последовательности. При этом значению $k_{\min}$ сопоставлена кодовая последовательность, состоящая из одних нулей, а значению $k_{\max}$ - кодовая последовательность, состоящая из одних единиц. Длина этих кодовых последовательностей зависит от значений k_1 и k_2, а также от частоты дискретизации интервала $[k_{min}, k_{\max}]$.

Допустим, что $k_{\min} = -25$, а $k_{\max} = 25$ и для каждого из параметров k_1, и k_2 применяются кодовые последовательности длиной 10. Пример популяции, состоящей из 10 особей, приведен в таблице 1.1.

Первые 10 генов каждого генотипа соответствуют параметру k_1, а последние 10 генов - параметру k_2. Таким образом, длина хромосом равна 20.

Способ кодирования значений параметров k_1 и k_2 в виде хромосом будет подробно изложен дальше.

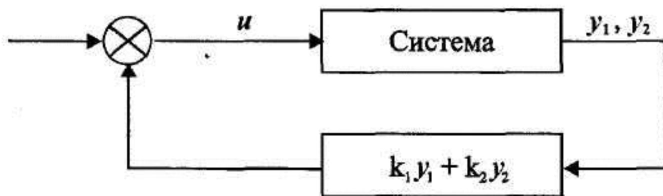


Рис. 1.1. Схема оптимизационной двухпараметрической системы.

Таблица 1.1. Популяция особей (для примера 1.2)

Генотипы	Фенотипы	
00000000000000000000	–25,00	–25,00
10100010010011001011	6,72	–15,08
01101000101111010010	–4,57	22,8.
11011010011110000111	17,67	19,13
00011011000000010001	–19,72	–24,17
00110000101011111010	–15,52	12,24
11111111111111111111	25,00	25,00

Пример 1.3

Рассмотрим нейронную сеть, представленную на рис. 1.2, для которой необходимо подобрать оптимальные веса w_{11}, w_{12}, w_{21}, w_{22}, w_{31}, w_{32} и w_{10}, w_{20} и w_{30}> минимизирующих значение погрешности

$$Q = \frac{1}{4}\sum_{i=1}^{4}(d_i - y_i)^2 .$$

Это сеть, реализующая систему XOR, поэтому значения $u_{1,i}$ и $u_{2,i}$, а также d_i для i = 1.....4 принимают значения, приведенные в таблице.

u_1	u_2	d = XOR(u_1,u_2)
+1	+1	–1
+1	–1	+1
–1	+1	+1
–1	–1	–1

В качестве параметров рассматриваемой задачи выступают перечисленные выше веса, т.е. задача имеет 9 параметров. В данном примере набор из этих 9 параметров определяет точку пространства поиска и, следовательно, представляет собой возможное решение. Допустим, что веса могут принимать значения из интервала [-10, 10]. Тогда каждая хромосома будет комбинацией из 9 двоичных последовательностей, кодирующих конкретные веса. Это, очевидно, и есть генотипы. Соответствующие им фенотипы представлены значениями отдельных весов, т.е. множесвами соответствующих действительных чисел из интервала [-10, 10].

В приведенных примерах (1.1 - 1.3) хромосомы и генотипы обозначают одно и то же - фенотипы особей популяции, закодированные в форме упорядоченных последовательностей генов со значениями (аллелями), равными 0 или 1.
В генетике генотип задает генетическую структуру особи, которая может включать более одной хромосомы. Например, клетки человека содержат 46 хромосом. В генетических алгоритмах генотип определяется аналогичным образом, однако чаще всего он состоит всего из одной хромосомы, которая и выступает в роли особи популяции.
Длина хромосом зависит от условий задачи . Следует заметить, что в естественных организмах хромосома может состоять из нескольких сотен и даже тысяч генов. У человека имеется около 100 000 генов, хотя их точное количество до сих пор неизвестно.

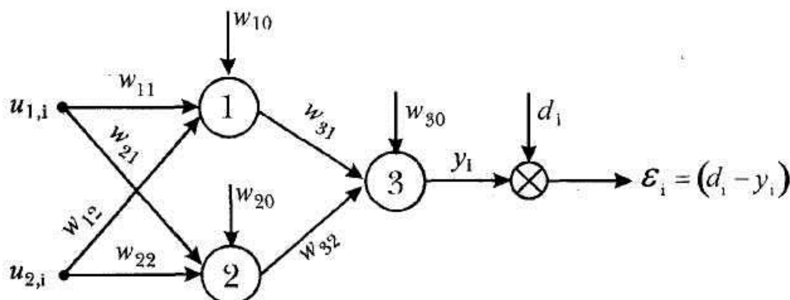


Рис. 1.2. Нейронная сеть, реализующая операцию XOR

1.2. Природный механизм (Естественный отбор в природе)

Живые существа характеризуются их внешними параметрами (фенотипом). Некоторые из параметров оказываются полезными для выживания и размножения, другие скорее вредят. Все внешние данные особи кодируются ее цепью ДНК (генотипом). Отдельные участки этой

цепи (гены) определяют различные параметры особи.

Согласно теории эволюции Чарльза Дарвина, особи популяции конкурируют между собой за ресурсы (пищу) и за привлечение брачного партнера. Те особи, которые наиболее приспособлены к окружающим условиям, проживут дольше и создадут более многочисленное потомство, чем их собратья. Скрещиваясь, они будут передавать потомкам часть своего генотипа. Некоторые дети совместят в себе части цепи ДНК, отвечающие за наиболее удачные качества родителей, и, таким образом, окажутся еще более приспособленными.

Те особи, которые не обладают качествами, способствующими их выживанию, с большой вероятностью не проживут долго и не смогут создать многочисленное потомство. Кроме того, им сложнее будет найти хорошую пару для скрещивания, поэтому с большой вероятностью генотип таких особей исчезнет из генофонда популяции.

Изредка происходит мутация: некоторый случайный нуклеотид цепи ДНК особи может измениться на другой. Если полученная цепь будет использоваться для создания потомства, то возможно появление у детей совершенно новых качеств.

Естественный отбор, скрещивание и мутация обеспечивают развитие популяции. Каждое новое поколение в среднем более приспособлено, чем предыдущее, т. е. оно лучше удовлетворяет требованиям внешней среды. Этот процесс называется эволюцией.

Рассматривая эволюцию в природе, возникает мысль о том, что можно искусственно отбирать особи, подходящие нам по некоторым параметрам, создавая таким образом искусственные внешние условия. Это называется селекцией и используется людьми для получения новых пород животных, к примеру, дающих больше молока или с более густой шерстью. Но почему бы не устроить собственную эволюцию с помощью компьютера? Действительно, пусть есть функция, которая по заданному набору численных параметров возвращает некоторое значение (многопараметрическая функция). Создадим множество строк, каждая из которых будет кодировать вектор чисел (длина вектора равна количеству параметров функции). По заданному вектору можно высчитать соответствующее ему значение функции. Те строки, для

которых это значение велико, будем считать более приспособленными, чем те, для которых оно мало. Запуская эволюцию на строках по подобию природной, на каждом поколении будем получать строки со все большими значениями функции. Таким образом, такого рода эволюция решает задачу максимизации многопараметрической функции.

Эволюционная теория утверждает, что каждый биологический вид целенаправленно развивается и изменяется для того, чтобы наилучшим образом приспособиться к окружающей среде. В процессе эволюции многие виды насекомых и рыб приобрели защитную окраску, еж стал неуязвимым благодаря иглам, человек стал обладателем сложнейшей нервной системы. Можно сказать, что эволюция - это процесс оптимизации всех живых организмов. Рассмотрим, какими же средствами природа решает эту задачу оптимизации.

Основной механизм эволюции - это естественный отбор. Его суть состоит в том, что более приспособленные особи имеют больше возможностей для выживания и размножения и, следовательно, приносят больше потомства, чем плохо приспособленные особи. При этом благодаря передаче генетической информации *(генетическому наследованию)* потомки наследуют от родителей основные их качества. Таким образом, потомки сильных индивидуумов также будут относительно хорошо приспособленными, а их доля в общей массе особей будет возрастать. После смены нескольких десятков или сотен поколений средняя приспособленность особей данного вида заметно возрастает.

Чтобы сделать понятными принципы работы генетических алгоритмов, поясним также, как устроены механизмы генетического наследования в природе. В каждой клетке любого животного содержится вся генетическая информация этой особи. Эта информация записана в виде набора очень длинных молекул ДНК (Дезоксирибо Нуклеиновая Кислота). Каждая молекула ДНК - это цепочка, состоящая из молекул *нуклеотидов* четырех типов, обозначаемых A, T, C и G. Собственно, информацию несет порядок следования нуклеотидов в ДНК. Таким образом, генетический код индивидуума - это просто очень длинная строка символов, где используются всего 4 буквы. В животной клетке каждая молекула ДНК окружена оболочкой - такое образование

называется *хромосомой.*
Каждое врожденное качество особи (цвет глаз, наследственные болезни, тип волос и т.д.) кодируется определенной частью хромосомы, которая называется *геном* этого свойства. Например, ген цвета глаз содержит информацию, кодирующую определенный цвет глаз. Различные значения гена называются его *аллелями.*
При размножении животных происходит слияние двух родительских половых клеток и их ДНК взаимодействуют, образуя ДНК потомка. Основной способ взаимодействия - *кроссовер (cross-over, скрещивание).* При кроссовере ДНК предков делятся на две части, а затем обмениваются своими половинками.
При наследовании возможны мутации из-за радиоактивности или других влияний, в результате которых могут измениться некоторые гены в половых клетках одного из родителей. Измененные гены передаются потомку и придают ему новые свойства. Если эти новые свойства полезны, они, скорее всего, сохранятся в данном виде - при этом произойдет скачкообразное повышение приспособленности вида.

Ключевую роль в эволюционной теории играет естественный отбор. Его суть состоит в том, что наиболее приспособленные особи лучше выживают и приносят больше потомков, чем менее приспособленные. Заметим, что сам по себе естественный отбор еще не обеспечивает развитие биологического вида. Поэтому очень важно понять, каким образом происходит наследование, то есть как свойства потомка зависят от свойств родителей.

Основной закон наследования интуитивно понятен каждому - он состоит в том, что потомки похожи на родителей. В частности, потомки более приспособленных родителей будут, скорее всего, одними из наиболее приспособленных в своем поколении. Чтобы понять, на чем основано это сходство, нужно немного углубиться в построение естественной клетки - в мир генов и хромосом.

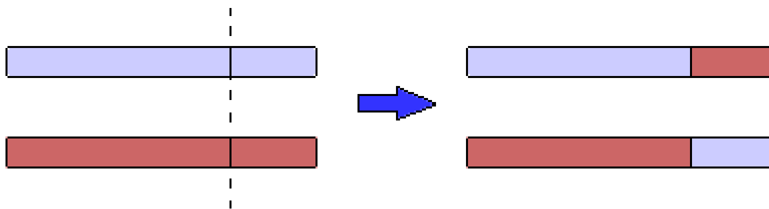
Почти в каждой клетке любой особи есть набор хромосом, несущих информацию про эту особь. Основная часть хромосомы - нить ДНК, определяющая, какие химические реакции будут происходить в данной клетке, как она будет развиваться и какие функции выполнять.

Ген - это отрезок цепи ДНК, ответственный за определенное свойство

особи, например за цвет глаз, тип волос, цвет кожи и т.д. Вся совокупность генетических признаков человека кодируется с помощью приблизительно 60 тыс. генов, длина которых составляет более 90 млн. нуклеотидов.

Различают два вида клеток: половые (такие, как сперматозоид и яйцеклетка) и соматические. В каждой соматической клетке человека содержится 46 хромосом. Эти 46 хромосом - на самом деле 23 пары, причем в каждой паре одна из хромосом получена от отца, а вторая - от матери. Парные хромосомы отвечают за одинаковые признаки - например, родительская хромосома может содержать ген черного цвета глаз, а парная ей материнская - ген голубого цвета. Существуют определенные законы, управляющие участием тех или иных генов в развитии особи. В частности, в нашем примере потомок будет черноглазым, поскольку ген голубых глаз является "слабым" и подавляется геном любого другого цвета.

В половых клетках хромосом только 23, и они непарные. При оплодотворении происходит слияние мужской и женской половых клеток и получается клетка зародыша, содержащая 46 хромосом. Какие свойства потомок получит от отца, а какие - от матери? Это зависит от того, какие именно половые клетки принимали участие в оплодотворении. Процесс выработки половых клеток (так называемый мейоз) в организме подвергается случаяйностям, благодаря которым потомки во многом отличаются от своих родителей. При мейозе, происходит следующее: парные хромосомы соматической клетки сближаются вплотную, потом их нити ДНК разрываются в нескольких случайных местах и хромосомы обмениваются своими частями (рис.).



Условная схема кроссинговера

Этот процесс обеспечивает появление новых вариантов хромосом и называется "кроссинговер". Каждая из вновь появившихся хромосом окажется затем внутри одной из половых клеток, и ее генетическая информация может реализоваться в потомках данной особи.

Второй важный фактор, влияющий на наследственность, - это мутации, которые выражаются в изменении некоторых участков ДНК. Мутации также случайны и могут быть вызваны различными внешними факторами, такими, как радиоактивное облучение. Если мутация произошла в половой клетке, то измененный ген может передаться потомку и проявиться в виде наследственной болезни либо в других новых свойствах потомка. Считается, что именно мутации являются причиной появления новых биологических видов, а кроссинговер определяет уже изменчивость внутри вида (например, генетические различия между людьми).

Что такое генетический алгоритм

Пусть дана некоторая сложная функция *(целевая функция),* зависящая от нескольких переменных, и требуется найти такие значения переменных, при которых значение функции максимально. Задачи такого рода называются *задачами оптимизации* и встречаются на практике очень часто.

Один из наиболее наглядных примеров - задача распределения инвестиций. В этой задаче переменными являются объемы инвестиций в каждый проект (10 переменных), а функцией, которую нужно максимизировать - суммарный доход инвестора. Также даны значения минимального и максимального объема вложения в каждый из проектов, которые задают область изменения каждой из переменных.

Попытаемся решить эту задачу, применяя известные нам природные способы оптимизации. Будем рассматривать каждый вариант инвестирования (набор значений переменных) как индивидуума, а доходность этого варианта - как приспособленность этого индивидуума. Тогда в процессе эволюции (если мы сумеем его организовать) приспособленность индивидуумов будет возрастать, а значит, будут появляться все более и более доходные варианты инвестирования. Остановив эволюцию в некоторый момент и выбрав самого лучшего индивидуума, мы получим достаточно хорошее решение задачи.

Генетический алгоритм - это простая модель эволюции в природе, реализованная в виде компьютерной программы. В нем используются как аналог механизма генетического наследования, так и аналог естественного отбора. При этом сохраняется биологическая терминология в упрощенном виде. Вот как моделируется генетическое наследование

Хромосома	Вектор (последовательность) из нулей и единиц. Каждая позиция (бит) называется геном.
Индивидуум = генетический код	Набор хромосом = вариант решения задачи.
Кроссовер	Операция, при которой две хромосомы обмениваются своими частями.
Мутация	Случайное изменение одной или нескольких позиций в хромосоме.

Чтобы смоделировать эволюционный процесс, сгенерируем вначале случайную популяцию - несколько индивидуумов со случайным набором хромосом (числовых векторов). Генетический алгоритм имитирует эволюцию этой популяции как циклический процесс скрещивания индивидуумов и смены поколений.

Жизненный цикл популяции - это несколько случайных скрещиваний (посредством кроссовера) и мутаций, в результате которых к популяции добавляется какое-то количество новых индивидуумов. Отбор в

генетическом алгоритме - это процесс формирования новой популяции из старой, после чего старая популяция погибает. После отбора к новой популяции опять применяются операции кроссовера и мутации, затем опять происходит отбор, и так далее.

Отбор в генетическом алгоритме тесно связан с принципами естественного отбора в природе следующим образом:

Приспособленность индивидуума	Значение целевой функции на этом индивидууме.
Выживание наиболее приспособленных	Популяция следующего поколения формируется в соответствии с целевой функцией. Чем приспособленнее индивидуум, тем больше вероятность его участия в кроссовере, т.е. размножении.

Таким образом, модель отбора определяет, каким образом следует строить популяцию следующего поколения. Как правило, вероятность участия индивидуума в скрещивании берется пропорциональной его приспособленнояти. Часто используется так называемая *стратегия элитизма,* при которой несколько лучших индивидуумов переходят в следующее поколение без изменений, не участвуя в кроссовере и отборе. В любом случае каждое следующее поколение будет в среднем лучше предыдущего. Когда приспособленность индивидуумов перестает заметно увеличиваться, процесс останавливают и в качестве решения задачи оптимизации берут наилучшего из найденных индивидуумов.

Возвращаясь к задаче оптимального распределения инвестиций, поясним особенности реализации генетического алгоритма в этом случае.

1. Индивидуум = вариант решения задачи = набор из 10 хромосом X_j
2. Хромосома X_j= объем вложения в проект j = 16-разрядная запись этого числа
3. Так как объемы вложений ограничены, не все значения хромосом являются допустимыми. Это учитывается при генерации популяций
4. Так как суммарный объем инвестиций фиксирован, то реально варьируются только 9 хромосом, а значение 10-ой определяется по ним однозначно.

Ниже приведены результаты работы генетического алгоритма для трех различных значений суммарного объема инвестиций К (рис.).

Квадратами на графиках прибылей отмечено, какой объем вложения в данный проект рекомендован генетическим алгоритмом.

Видно, что при малом значении К инвестируются только те проекты, которые прибыльны при минимальных вложениях. Если увеличить суммарный объем инвестиций, становится прибыльным вкладывать деньги и в более дорогостоящие проекты (рис.).

При дальнейшем увеличении K достигается порог максимального вложения в прибыльные проекты, и инвестирование в малоприбыльные проекты опять приобретает смысл.

1.3. Особенности ГА

Генетический алгоритм - новейший, но не единственно возможный способ решения задач оптимизации. С давних пор известны два основных пути решения таких задач - **переборный и локально-градиентный**. У этих методов свои достоинства и недостатки, и в каждом конкретном случае следует подумать, какой из них выбрать.

Рассмотрим достоинства и недостатки стандартных и генетических методов на примере классической *задачи коммивояжера* (TSP - travelling salesman problem). Суть задачи состоит в том, чтобы найти кратчайший замкнутый путь обхода нескольких городов, заданных своими координатами. Оказывается, что уже для 30 городов поиск оптимального пути представляет собой сложную задачу, побудившую развитие различных новых методов (в том числе нейросетей и

генетических алгоритмов).

Каждый вариант решения (для 30 городов) - это числовая строка, где на j-ом месте стоит номер j-ого по порядку обхода города. Таким образом, в этой задаче 30 параметров, причем не все комбинации значений допустимы. Естественно, первой идеей является полный перебор всех вариантов обхода.

Переборный метод наиболее прост по своей сути и тривиален в программировании. Для поиска оптимального решения (точки максимума целевой функции) требуется последовательно вычислить значения целевой функции во всех возможных точках, запоминая максимальное из них (рис.1.3).

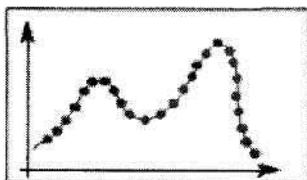


Рис.1.3.

Недостатком этого метода является большая вычислительная стоимость. В частности, в задаче коммивояжера потребуется просчитать длины более 10^{30} вариантов путей, что совершенно нереально. Однако, если перебор всех вариантов за разумное время возможен, то можно быть абсолютно уверенным в том, что найденное решение действительно оптимально.

Второй популярный способ основан на методе градиентного спуска. При этом вначале выбираются некоторые случайные значения параметров, а затем эти значения постепенно изменяют, добиваясь наибольшей скорости роста целевой функции (рис.1.4).

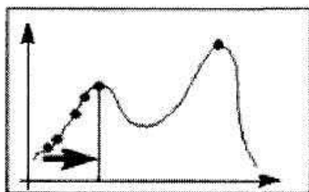


Рис.1.4.

Достигнув локального максимума, такой алгоритм останавливается, поэтому для поиска глобального оптимума потребуются дополнительные усилия.
Градиентные методы работают очень быстро, но не гарантируют оптимальности найденного решения.
Они идеальны для применения в так называемых *унимодальных* задачах, где целевая функция имеет единственный локальный максимум (он же -глобальный). Легко видеть, что задача коммивояжера унимодальной не является (рис.1.5).

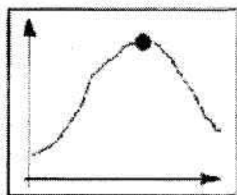


Рис.1.5.

Типичная практическая задача, как правило, *мультимодальна* и многомерна, то есть содержит много параметров. Для таких задач не существует ни одного универсального метода, который позволял бы достаточно быстро найти абсолютно точное решение (рис.1.6).

Рис.1.6.

Однако, комбинируя переборный и градиентный методы, можно надеяться поручить хотя бы приближенное решение, точность которого будет возрастать при увеличении времени расчета (рис.1.7).

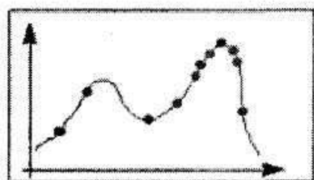


Рис.1.7.

Генетический алгоритм представляет собой именно такой комбинированный метод. Механизмы скрещивания и мутации в каком-то смысле реализуют переборную часть метода, а отбор лучших решений - градиентный спуск. На рисунке показано, что такая комбинация позволяет обеспечить устойчиво хорошую эффективность генетического поиска для любых типов задач (рис.1.8).

Рис.1.8.

Итак, если на некотором множестве задана сложная функция от нескольких переменных, то генетический алгоритм - это программа, которая за разумное время находит точку, где значение функции достаточно близко к максимально возможному. Выбирая приемлемое время расчета, мы получим одно из лучших решений, которые вообще возможно получить за это время.

1.4. Задачи оптимизации и применение алгоритмов

Генетические алгоритмы - это очень популярные способы решения задач оптимизации. В их основе лежит использование эволюционных принципов для поиска оптимального решения. Уже сама идея выглядит довольно интригующей и любопытной, чтобы претворить её в жизнь, а многочисленные положительные результаты только разжигают интерес со стороны исследователей.

Здесь не будет рассматриваться история становления и признания эволюционных вычислений вообще и генетических алгоритмов в частности. Вместо этого мы сразу перейду к рассмотрению самих алгоритмов.

Генетические алгоритмы в основном применяются для решения задач оптимизации, т.е. задач, в которых есть некоторая функция нескольких переменных $F(x_1, x_2, \dots ,x_n)$ и необходимо найти либо её максимум, либо её минимум. Функция F называется *целевой функцией*, а переменные - *параметрами функции*. Генетические алгоритмы "пришиваются" к данной задаче следующим образом.

Параметры задачи являются генетическим материалом - генами. Совокупность генов составляет хромосому. Каждая особь обладает своей хромосомой, а, следовательно, своим набором параметров. Подставив параметры в целевую функцию, можно получить какое-то значение. То, насколько это значение удовлетворяет поставленным условиям, определяет характеристику особи, которая называется *приспособленностью (fitness)*. Функция, определяющая приспособленность должна удовлетворять следующему условию: чем "лучше" особь, тем выше приспособленность. Генетические алгоритмы работают с популяцией как правило фиксированного размера, состоящей из особей, заданных при помощи способа, описанного выше. Особи "скрещиваются" между собой с помощью *генетических операторов* (о том как происходит этот процесс – будет описано отдельно), и таким образом получается потомство, причем часть потомков заменяют представителей более старого поколения в соответствии со *стратегией формирования нового поколения*. Выбор

особей для скрещивания проводится согласно *селективной стратегии (selection strategy)*. Заново сформированная популяция снова оценивается, затем выбираются наиболее достойные для скрещивания особи, которые скрещиваются между собой, получаются «дети»и занимают место «престарелых» индивидуумов и т.д.

Все эта продолжается до тех пор пока не найдется особь, гены которой представляют оптимальный набор параметров, при которых значение целевой функции близко к максимуму или минимуму, либо равно ему. Останов работы ГА может произойти также в случае, если популяция вырождается, т.е. если практически нет разнообразия в генах особей популяции, либо если просто вышел лимит времени. Вырождение популяции называют *преждевременной сходимостью (premature convergence)*.

Может создаться впечатление, что ГА являются просто извращенным вариантом случайного поиска. Но приспособленность была введена совсем не зря. Дело в том, что она непосредственно влияет на шанс особи принять участие в скрещивании с последующим «рождением детей». Выбирая каждый раз для скрещивания наиболее приспособленных особей, можно с определенной степенью уверенности утверждать, что потомки будут либо не намного хуже, чем родители, либо лучше их. Приблизительно эту величину уверенности можно оценить с помощью теоремы шаблонов (теоремы шим).

Теоретические аспекты ГА, следующие:

- Представление данных в генах
- Генетические операторы
- Модели ГА
- Тестовые функции
- Стратегии отбора и формирования нового поколения

Где же применяются ГА? Всего применений очень много, поэтому приведенный список не является исчерпывающим.

- Экстремальные задачи (нахождение точек минимума и минимума);

- Задачи о кратчайшем пути;
- Задачи компоновки;
- Составление расписаний;
- Аппроксимация функций;
- Отбор (фильтрация) входных данных;
- Настройка искусственной нейронной сети;
- Моделирование искусственной жизни (Artificial life systems) ;
- Биоинформатика (свертывание белков и РНК);
- Игровые стратегии;
- Нелинейная фильтрация;
- Развивающиеся агенты/машины (Evolvable agents/machines);
- Оптимизация запросов в базах данных
- Разнообразные задачи на графах (задача коммивояжера, раскраска, нахождение паросочетаний)
- Обучение искусственной нейронной сети
- Искусственная жизнь

Некоторые разделы могут содержать подпункты. Так, например, экстремальные задачи включают в мебя целый класс задач линейного и нелинейного программирования.

1.5. Мягкие вычисления

Термин "мягкие вычисления" введен Лотфи Заде в 1994 году.
Мягкие вычисления не являются отдельной методологией. Это понятие объединяет такие области как нечеткая логика, нейровычисления, эволюционные вычисления и вероятностные вычисления с более поздним включением хаотических систем, сетей доверия и разделов теории обучения.

Каждая из составляющих областей имеет много возможностей для ее использования в рамках мягких вычислений. Нечеткая логика лежит в основе методов работы с неточностью, зернистой структурой (гранулированной информацией), приближенных рассуждений и вычислений со словами (computing with words). Нейровычисления отражают способность к обучению, адаптации и идентификации. В случае эволюционных вычислений, речь идет о возможности систематизировать случайный поиск и достигать оптимального значения характеристик. Вероятностные вычисления обеспечивают базу для управления неопределенностью и проведения рассуждений, исходящих из свидетельств[Zadeh].
Системы, в которых нечеткая логика, нейровычисления, генетические алгоритмы и вероятностные вычисления используются в некоторой комбинации, называются гибридными системами. Наиболее известными системами этого типа являются так называемые нейро-нечеткие системы. В настоящее время появляются нечетко-генетические системы, нейро-генетические системы и нейро-нечетко-генетические системы.

Сущность мягких вычислений (Soft Computing) состоит в том, что в отличие от традиционных, жестких вычислений, они нацелены на приспособление к всеобъемлющей неточности реального мира. Руководящим принципом мягких вычислений является: «терпимость к

неточности, неопределенности и частичной истинности для достижения удобства манипулирования, робастности, низкой стоимости решения и лучшего согласия с реальностью». Исходной моделью для мягких вычислений служит человеческое мышление.

Мягкие вычисления не являются отдельной методологией. Это, скорее, объединение, партнерство различных направлений. Главными партнерами в этом объединении являются нечеткая логика, нейровычисления, генетические вычисления и вероятностные вычисления с более поздним включением хаотических систем, сетей доверия и разделов теории обучения.

В ближайшие годы повсеместное распространение интеллектуальных систем несомненно окажет глубокое влияние на сами способы зарождения, конструирования, производства, использования и взаимодействия интеллектуальных систем. Именно в этой перспективе основные вопросы, связанные с мягкими вычислениями и интеллектуальными системами, рассматриваются в этой статье.

1. Введение

Чтобы увидеть эволюцию нечеткой логики в ближайшей перспективе, важно заметить, что мы присутствуем при завершении информационной революции. Продукты этой революции видны всем: Интернет, Всемирная Паутина, мобильные телефоны, факсимильные машины и бортовые компьютеры с мощными возможностями по обработке информации стали частью повседневной жизни. Революция произошла. Центральная роль информации почти во всех наших делах становится фактом, который трудно оспаривать.

Гораздо менее заметной, но потенциально столь же или даже более важной, чем информационная революция, является революция в области интеллектуальных систем. Артефактами этой революции являются созданные человеком системы, которые обладают способностями рассуждать, учиться на своем или чужом опыте и принимать разумные решения без человеческого вмешательства. Термин «коэффициент машинного интеллекта», сокращенно КМИ (MIQ, Machine Intelligence Quotient) введен для характеристики меры

интеллекта создаваемых человеком систем. В этом плане интеллектуальная система может пониматься как система с высоким КМИ.

Более детально мы остановимся на КМИ ниже. Вопрос, который хотелось бы поднять сейчас, состоит в следующем. Мы говорим об искусственном интеллекте уже многие десятилети. Почему же ИИ оказалось нужно так много времени, чтобы получить видимые результаты?

Приведем пример, имеющий отношение к этому вопросу. Когда Заде был преподавателем Колумбийского университета, он написал статью под названием «Думающие машины – новая область в электротехнике», которая была опубликована в студенческом журнале. В первом параграфе этой статьи Заде процитировал ряд заголовков, появившихся в популярной прессе того времени. Один из заголовков гласил: «Электрический мозг, способный переводить иностранные языки, создан». Эта статья была опубликована в январе 1950, примерно за шесть лет до появления самого термина «искусственный интеллект». Сейчас очевидно, что машина- переводчик не могла быть создана в 1950-м году или ранее. Просто необходимые методологии и технологии тогда отсутствовали.

Теперь мы ведем себя гораздо скромнее, чем в то время. Трудности построения систем, которые могли бы имитировать человеческие рассуждения и познавательные способности, оказались значительно большими, чем предполагалось ранее. Даже сегодня, имея в нашем распоряжении широкий набор мощных средств, мы все еще не способны построить машины, которые могли бы делать то, что многие дети делают с легкостью. Например, машины, понимающие волшебные сказки, способные чистить апельсин или есть пищу ножом и вилкой.

Вернемся здесь к понятию КМИ. **Основное отличие между традиционным коэффициентом интеллектуальности КИ (IQ) и КМИ состоит в том, что КИ является более или менее постоянным, тогда как КМИ изменяется со временем и машинно-зависим**. Более того, характеристики КМИ и КИ не совпадают. Например, распознавание речи может быть важной характеристикой для КМИ, но в случае КИ считается тривиальным свойством, всегда имеющимся в

наличии.

Пока мы не имеем даже согласованного множества тестов, чтобы измерять КМИ некоторой системы, созданной человеком, например, портативной видеокамеры. Заде отметил, что такие тесты будут скоро придуманы, и в конце концов понятие КМИ станет играть важную роль в измерении машинного интеллекта.

В действительности, мы только начинаем входить в век интеллектуальных систем. Почему для того, чтобы это случилось, потребовалось так много времени?

По мнению Заде, главная причина такова. **На начальном этапе основные инструментарии в арсенале ИИ были сконцентрированы на манипулировании символами и логике предикатов, в то время как численные методы оказались в немилости. Сегодня стало вполне очевидным, что манипулирование символами и логика предикатов имеют серьезные ограничения при работе со многими проблемами реального мира.** Это касается таких областей как компьютерное зрение, распознавание речи, распознавание рукописей, понимание образов, поиск в мультимедийных базах данных, планирование движений, рассуждения здравого смысла, управление неопределенностью, и многих других областей, связанных с машинным интеллектом.

2. Мягкие вычисления и нечеткая логика

В течение ряда прошедших лет наша способность понимать, конструировать и развивать машины с высоким КМИ значительно усилилась в результате появления мягких вычислений. **Мягкие вычисления (SC) – это не какая-то отдельная методология. Скорее, это консорциум вычислительных методологий, которые коллективно обеспечивают основы для понимания, конструирования и развития интеллектуальных систем. В этом объединении главными компонентами SC являются нечеткая логика (FL), нейровычисления (NC), генетические вычисления (GC) и вероятностные вычисления (PC). Позднее в этот конгломерат были включены рассуждений на базе свидетельств (evidential reasoning), сети доверия (belief networks), хаотические**

системы и разделы теории машинного обучения. По сравнению с традиционными жесткими вычислениями, мягкие вычисления более приспособлены для работы с неточными, неопределенными или частично истинными данными/ знаниями. **Руководящим принципом мягких вычислений является: «терпимость к неточности, неопределенности и частичной истинности для достижения удобства манипулирования, робастности, низкой стоимости решения и лучшего согласия с реальностью».**

В мягких вычислениях весьма важно то, что составляющие их методологии являются в большей степени синергетическими и взаимодополняющими, чем соперничающими. Таким образом, **во многих случаях более высокого КМИ можно достигнуть путем совместного использования FL, NC, GC и PC, чем путем их применения по отдельности**. Более того, существует много проблем, которые не могут быть решены только каким-то одним средством: нечеткой логикой, нейровычислениями, генетическими вычислениями или вероятностными рассуждениями. Это подвергает сомнению позиции тех, кто во всеуслышание заявляет, что его любимый инструментарий, будь то FL, NC, GC или PC, может решить все проблемы. По мере распространения мягких вычислений число приверженцев таких односторонних точек зрения будет неуклонно сокращаться. **Каждая из составляющих методологий имеет много возможностей для ее использования в рамках мягких вычислений. Нечеткая логика лежит в основе методов работы с неточностью, зернистой структурой (гранулированной) информацией, приближенных рассужденияй и, что наиболее важно, вычислений со словами (Computing with Words). Нейровычисления отражают способность к обучению, адаптации и идентификации. В случае генетических вычислений, речь идет о возможности систематизировать случайный поиск и достигать оптимального значения характеристик. Вероятностные вычисления обеспечивают базу для управления неопределенностью и проведения рассуждений, исходящих из свидетельств.**

Системы, в которых FL, NC, GC и PC используются в некоторой комбинации, называются **гибридными системами**. **Наиболее известными системами этого типа являются так называемые нейро-нечеткие системы.** Мы начинаем также строить **нечетко-**

генетические системы, нейро-генетические системы и нейро-нечетко-генетические системы. По мнению Заде, в конечном итоге **большинство систем с высоким КМИ будут гибридными системами**. **В будущем широкое распространение интеллектуальных систем будет иметь глубокое влияние на сами способы, с помощью которых интеллектуальные системы конструируются, производятся и взаимодействуют.**

Каково место нечеткой логики в мягких вычислениях? Прежде, чем ответить на этот вопрос, необходимо прояснить, что такое нечеткая логика и что она должна нам давать. Источником путаницы является то, что термин нечеткая логика используется в двух различных смыслах. В узком смысле, нечеткая логика – это логическая система, являющаяся расширением многозначной логики. Однако, даже для нечеткой логики в узком смысле, список основных операций очень отличается как по духу, так и по содержанию от списка основных операций для систем многозначных логик. В широком смысле слова, который сегодня преобладает, нечеткая логика равнозначна теории нечетких множеств, т.е. классов с неточными, размытыми границами. Таким образом, нечеткая логика, понимаемая в узком смысле, является разделом нечеткой логики в широком смысле.

Важной характеристикой нечеткой логики является то, что любая теория T может быть фаззифицирована (fuzzified) и, следовательно, обобщена путем замены понятия четкого множества в T понятием нечеткого множества. **Таким способом можно прийти к нечеткой арифметике, нечеткой топологии, нечеткой теории вероятностей, нечеткому управлению, нечеткому анализу решений…** Выигрышем от фаззификации является большая общность и лучшее соответствие модели действительности. Однако с нечеткими числами труднее оперировать, чем с четкими числами. Более того, значения большинства нечетких понятий зависят от контекста и/или приложения. Это та цена, которую необходимо заплатить за лучшее согласие с реальностью.

3. Гранулирование информации

В основе методов работы с нечеткими понятиями лежит одна важнейшая особенность. Речь идет о **гранулировании информации** (Information Granulation) и его роли в человеческих рассуждениях,

взаимодействиях и формировании концепций. Ниже мы попытаемся объяснить, почему гранулирование информации играет существенную роль в оперировании нечеткими понятиями и, в частности, в рассуждениях и вычислениях со словами, а не с числами.

Понятие гранулирования информации послужило мотивировкой для написания большинства ранних работ Заде по нечетким множествам и нечеткой логике. **По существу, все человеческие понятия являются нечеткими, так как они получаются в результате группировки (clumping) точек или объектов, объединяемых по сходству. Тогда нечеткость подобных групп (clumps) есть прямое следствие нечеткости понятия сходства.** Простыми примерами таких групп являются понятия «средний возраст», «деловая часть города», «немного облачно», «бестолковый» и др. **Будем называть данную группу «гранулой» (granule).**

В естественном языке (ЕЯ) слова играют роль меток гранул. **В этой ипостаси они служат для сжатия данных. Сжатие данных с помощью слов является ключевым аспектом человеческих рассуждений и формирования понятий.**

В нечеткой логике гранулирование информации лежит в основе понятий лингвистической переменной и нечетких правил типа «если, …, то» . Эти понятия были формально введены Заде в 1973 году в статье "Основы нового подхода к анализу сложных систем и процессов принятия решений". Сегодня почти все приложения нечеткой логики используют эти понятия. С исторической точки зрения, следует заметить, что введение этих понятий было встречено со скептицизмом и враждебностью многими известными членами научного истеблишмента.

Важность нечетких правил связана с тем, что такие правила близки человеческой интуиции. В нечеткой логике нечеткие правила играют центральную роль **в языке нечетких зависимостей и команд** (Fuzzy Dependency and Command Language, FDCL). С неформальной точки зрения, это как раз тот язык, который используется в большинстве приложений нечеткой логики.

При сравнении нечеткой логики с другими методологиями

существенный момент, который часто не осознается, состоит в том, что исходной посылкой в решении, получаемом с помощью нечеткой логики, является человеческое решение. Таким образом, **нечеткое логическое решение обычно представляет собой человеческое решение, выраженное в FDCL.** Здесь вполне понятным примером может служить задача парковки автомобиля, в которой целью является установка автомобиля рядом с обочиной и почти параллельно ей. Нечетко-логическим решением проблемы парковки был бы набор нечетких «если-то»-правил, которые описывают то, как человек паркует автомобиль. Напротив, проблему парковки тяжело решить в контексте классического управления, поскольку в нем отправной точкой является не человеческое решение, а описание конечного состояния, начального состояния, ограничений и уравнений движения.

Другим примером, иллюстрирующим суть гранулирования информации, является следующий. Рассмотрим ситуацию, в которой субъект *A* разговаривает по телефону с субъектом *B*, которого *A* не знает. За короткое время разговора, скажем, за 10-20 секунд, *A* может сформировать грубую оценку возраста *B*, выраженную, например, следующим образом:

“Вероятность того, что *B* очень молодой, очень малая”,

“Вероятность того, что *B* молодой, малая”,

“Вероятность того, что *B* средних лет, большая”,

“Вероятность того, что *B* старый, малая”,

“Вероятность того, что *B* очень старый, очень малая”.

Эти оценки могут интерпретироваться как гранулярное представление вероятностного распределения *P* возраста *B*. В символической форме *P* может быть представлено нечетким графом:

Р = очень малая\очень молодой + малая\молодой +

большая\средних лет + малая\старый + очень малая\очень старый

В этом выражении + означает оператор объединения, а терм типа «малая\старый» означает, что «малая» есть лингвистическая вероятность того, что *B* – «старый». Здесь важным моментом является то, что человек может формировать такие оценки, используя лингвистические, т.е. гранулированные значения возраста и вероятностей. В то же время человек не может думать на основе численных оценок в форме «Вероятность того, что субъекту *B* 25 лет равна 0.012».

Следует заметить, что во многих случаях человек оценил бы возраст В термином «средних лет», опуская его вероятность. Пропуск вероятностей может быть оправданным, если существует так называемое *p*-доминантное значение в вероятностном распределении, т.е. такое значение, вероятность которого доминирует вероятности других значений. Пропуск вероятностей играет ключевую роль в **приближенном рассуждении**.

Гранулирование информации лежит в центре человеческих рассуждений, взаимодействий и формирования понятий. В рамках нечеткой логики оно играет ключевую роль в вычислениях со словами (CW). **Вычисления со словами можно рассматривать как один из наиболее важных результатов нечеткой логики**. Что это такое? Как следует из названия, при вычислениях со словами объектами вычислений являются слова, а не числа, причем слова играют роль меток гранул. **Очень простыми примерами CW являются следующие суждения:**

«Дана молодая, а Танди на несколько лет старше, чем Дана»

(Танди is (молодая + несколько) лет

«Большинство студентов –молодые и большинство молодых студентов одиноки (не имеют семьи)»

(большинство^2 студентов одиноки)

В этих примерах «молодой», «несколько» и «большинство» суть нечеткие числа, + есть операция сложения в нечеткой арифметике, а «большинство^2»–это квадрат от «большинство» в нечеткой арифметике.

В западных культурах существует глубоко заложенная традиция оказывать большее почтение числам, чем словам. Но для любой традиции приходит время, когда ее разумное объяснение оказывается неудовлетворительным и она ставится под сомнение. С точки зрения Заде, пришло время задаться вопросом обоснованности традиции больше верить числам, чем словам.

В этой ситуации необходима система, которая позволяет выражать данные в виде предложений естественного языка. Это как раз то, что пытаются обеспечить вычисления со словами. **Исходным моментом в вычислениях со словами является набор предложений, выраженных на ЕЯ. Этот набор называется множеством исходных данных (Initial Data Set, IDS). Желаемые ответы или заключения также выражаются в терминах ЕЯ. Этот набор называется множеством конечных (терминальных) данных (Terminal Data Set, TDS). Проблема состоит в достижении TDS, стартуя с IDS.**

Очень простым примером является следующий: множество исходных данных состоит из предложения «Большинство шведов высокие», а множеством конечных данных есть ответ на вопрос «Каков средний рост шведов?» Предполагается, что ответ имеет форму «Средний рост шведов есть А», где А–лингвистическое значение роста. **В этом примере цель CW – вычислить А по информации, заложенной во множестве исходных данных.**

В вычислениях со словами слова играют роль нечетких ограничений (constraints), а все предложение интерпретируется как нечеткое ограничение на переменную. Например, предложение «Мэри молода» интерпретируется как нечеткое ограничение на возраст Мэри. В символьной записи:

Мэри is молода = Возраст(Мэри) is молодой

В этом выражении = представляет собой операцию разъяснения (explicitation), Возраст (Мэри) – ограничиваемую переменную, а «молодой» – нечеткое отношение, которое ограничивает Возраст(Мэри).

В общем случае, если *p* – предложение на ЕЯ, то результат разъяснения *p* называется **канонической формой *p*.** В своей основе, каноническая форма предложения *p* делает явным (explicit) неявное (implicit) нечеткое ограничение на *p* и, таким образом, служит для определения значения *p* как ограничения на переменную. **В более общей постановке каноническая форма *p* представляется как**

X isr R,

где *X*–лингвистически ограничиваемая переменная, например, Возраст(Мэри), *R* – ограничивающее нечеткое отношение, например, «молодой», и *isr* – переменная, в которой *r* – дискретная переменная, значения которой определяют роль *R* по отношению к *X*. В частности, если *r=d,* то *isd* обозначается как «*is*», и ограничение «*X is R*» называется дизъюнктивным. В этом случае *R* определяет распределение возможности для *X*. Какова причина трактовать *r* как переменную? Богатство естественных языков делает необходимым использование широкого разнообразия ограничений для представления значений предложения, выраженного средствами ЕЯ. В вычислениях со словами основными типами ограничений, которые используются в дополнение к дизъюнктивному типу, являются: конъюнктивный, вероятностный, обычность (usuality), случайное множество, грубое множество, нечеткий граф и функциональные типы. Каждый из этих типов соответствует конкретному значению *r*.

В вычислениях со словами первый шаг в определении множества конечных данных состоит в разъяснении, т.е. в представлении предложений из IDS в их канонической форме. Следующий шаг включает распространение ограничений (constraint propagation), которое осуществляется в результате использования правил вывода нечеткой логики. В сущности, правила вывода в нечеткой логике могут интерпретироваться как правила распространения ограничений. **Третий и завершающий шаг в вычислении множества конечных данных включает ретрансляцию (retranslation)**

выведенных ограничений в предложения, выраженные на ЕЯ. В нечеткой логике это требует использования лингвистической аппроксимации.

Следует понимать, что отмеченные выше шаги могут требовать широкого использования обычных вычислений с числами. Однако, здесь вычисления со словами находятся на авансцене, тогда как обычные вычисления проходят как бы за занавесом и скрыты от взгляда пользователя.

Итак, что должны обеспечить методы CW? Возможность вывода из множества исходных данных, в котором информация выражается в виде предложений на ЕЯ, открывает пути для формулировки и решения многих важных проблем, в которых имеющаяся информация не является достаточно точной для использования традиционных методов. Для иллюстрации предположим, что имеется задача максимизации функции, описанной словами в виде нечетких «если-то»-правил:

Если X is малое, то Y is малое,

Если X is среднее, то Y is среднее,

Если X is большое, то Y is малое,

в которых значения «малое», «среднее» и «большое» определены с помощью их функций принадлежности.

Другая подобная задача заключается в следующем. Предположим, что ящик содержит десять шаров разного размера, из которых несколько больших, и немного малых. Какова вероятность того, что случайно вытянутый шар не является ни большим, ни малым?

В этих примерах, предложения, содержащиеся в множестве исходных данных, довольно просты. **Действительная проблема состоит в разработке систем вычислений со словами, способных справиться с предложениями значительно большей сложности, которые**

выражают знания о реальном мире.

В этом контексте, вычисления со словами есть раздел нечеткой логики. В представлении Заде, **вычисления со словами** должны превратиться **в важнейшее научное направление**, обеспечивающее эффективную работу с всеобъемлющей неточностью и неопределенностью реального мира. В этой перспективе исходной моделью для вычислений со словами, нечеткой логики и мягких вычислений является человеческий разум.

Понимание, конструирование и развитие информационных/ интеллектуальных систем представляет собой серьезный вызов всем тем, кто вовлечен в разработку и приложения нечеткой логики и мягких вычислений.

1.6. Эволюционные вычисления

Генетические алгоритмы являются частью более общей группы методов, называемой эволюционными вычислениями, которые объединяют различные варианты использования эволюционных принципов для достижения поставленной цели.

Также в ней выделяют следующие направления:

- Эволюционные стратегии
  - Метод оптимизации, основанный на идеях адаптации и эволюции. Степень мутации в данном случае меняется со временем – это приводит к, так называемой, самоадаптации.
- Генетическое программирование
  - Применение эволюционного подхода к популяции программ.
- Эволюционное программирование
  - Было впервые предложено Л.Дж. Фогелем в 1960 году для моделирования эволюции как процесса обучения с целью создания искусственного интеллекта. Он использовал конечные автоматы, предсказывающие символы в цифровых последовательностях, которые, эволюционируя, становились более приспособленными к решению

поставленной задачи.

Генетические алгоритмы применяются для решения следующих задач:

- Оптимизация функций
- Разнообразные задачи на графах (задача коммивояжера, раскраска, нахождение паросочетаний)
- Настройка и обучение искусственной нейронной сети
- Задачи компоновки
- Составление расписаний
- Игровые стратегии
- Аппроксимация функций
- Искусственная жизнь
- Биоинформатика

Насколько осмысленной является идея использования законов эволюции не при имитации самой эволюции, а при решении интеллектуальных задач? На первый взгляд, идея может показаться интересной, но немного сомнительной: не происходит же у нас в голове эволюция, пусть даже виртуальная. Однако эта идея была высказана достаточно давно сразу несколькими авторами и оказалась относительно успешной. Сейчас область исследований, опирающихся на сходные идеи, называется эволюционными вычислениями. Наиболее ранним вкладом в нее в 1960-х годах стали эволюционные стратегии, предложенные Инго Рехенбергом с коллегами, и эволюционное программирование, предложенное Лоуренсом Фогелем, а также генетические алгоритмы Джона Генри Холланда (которые стали известны в 1970-х). Позднее возникла концепция генетического программирования. Все эти разновидности эволюционных вычислений являются альтернативой классическим методам поиска и оптимизации, включая эвристическое программирование.

Существующие методы эволюционных вычислений берут за основу идеи Дарвина. Одна из этих идей гласит: выживает наиболее приспособленный. В действительности, это тавтология, поскольку более приспособленный — это, по определению, тот, кто имеет больше шансов выжить. Эту тавтологичность подчеркивают и сами эволюционисты, чтобы доказать неизбежность эволюции, несмотря на отсутствие у природы каких-либо специальных целей. Не случайно идея

выживания наиболее приспособленных часто называется естественным отбором. Отбор обычно подразумевает целенаправленное действие, но в природе он происходит «автоматически».

Однако неизбежность эволюции не следует лишь из логической истинности естественного отбора. Ведь для того, чтобы эволюция стала «работать», необходимо появление новых альтернатив, из которых осуществляется отбор. В связи с этим вводится дополнительная идея — идея изменчивости видов. То, что виды изменяются со временем, не самоочевидно и может быть установлено только из эмпирических данных, поскольку изменчивость должна обеспечиваться какими-то конкретными механизмами. Так, мы не можем сказать, что сейчас на уровне элементарных частиц идет естественный отбор, поскольку не происходит возникновение все новых и новых видов частиц.

Еще одна идея — это идея наследственности. Если бы виды возникали произвольно, то со временем могли бы появляться все более приспособленные виды, однако появление новых видов не зависело бы от степени приспособленности уже существующих, ведь при возникновении нового вида заранее нельзя сказать, насколько приспособленным он будет (если этот вид не конструируется сознательно). Наследственность же обеспечивает последовательное улучшение существующих решений за счет изменчивости и естественного отбора (стоит отметить, что здесь неявно присутствует предположение непрерывности: небольшое изменение вида обычно не сильно сказывается на степени его приспособленности).

Таким образом, как хорошо известно, базовые элементы дарвиновской концепции эволюции — это наследственность, изменчивость и естественный отбор. Поскольку в процессе эволюции появляются все более приспособленные виды, эволюцию можно трактовать как поиск максимума некоторой функции выживания, или фитнесс-функции. Если предположить, что в процессе эволюции в некоторый момент создан совершенный вид, то на нем эволюция прекращается, так как изменение этого вида может привести только к ухудшению его выживаемости (т. е. к уменьшению значения фитнесс-функции). Оставим на время в стороне вопрос о том, насколько биологическая эволюция соответствует такой модели, и согласимся с определенным сходством между эволюцией и поиском, которое и было замечено

исследователями в области ИИ.

Однако перечисленные выше идеи Дарвина добавляют немногое к классическим методам поиска. Чтобы показать это, достаточно переформулировать упоминавшийся при обсуждении проблем поиска метод градиентного спуска (или подъема в гору, т. е. движения в направлении наискорейшего локального возрастания или убывания функции) в эволюционных терминах. Пусть текущая точка в методе градиентного спуска — это некоторая особь. На каждом шаге градиентного спуска проверяются некоторые точки вблизи текущей (наследственность и изменчивость) и выбирается из них лучшая (отбор). Стохастический градиентный спуск из многих точек обладает еще большим сходством с дарвиновской эволюцией, если ее ограничить лишь указанными идеями. На самом деле, еще сам Дарвин называл отбор в сочетании с изменчивостью «спуском с модификацией». Таким образом, обычный градиентный спуск содержит все базовые элементы эволюции. С одной стороны, это еще раз подчеркивает сходство эволюции с поиском, но с другой стороны, ставит вопрос, что же исследователи ИИ нашли для себя нового в эволюции?

Заинтересовали их не только базовые идеи Дарвина, но и генетические механизмы передачи наследственной информации. В качестве таких механизмов чаще всего рассматриваются перекрест хромосом при скрещивании и генные мутации.

В общем виде алгоритмы эволюционных вычислений, предназначенные для поиска оптимального решения некоторой задачи, могут быть представлены следующим образом. Альтернативные решения (или оптимизируемые объекты) трактуются как особи, степень приспособленности которых, или фитнесс-функция, явно или неявно определяется условиями задачи. Эти особи «эволюционируют» — к ним применяются «генетические операторы», такие как операторы скрещивания, мутации и редукции (селекции или отбора). Такие алгоритмы обычно состоят из следующих шагов.

1. Сгенерировать начальную популяцию (случайную совокупность неоптимальных решений).
2. Выбрать родительские пары.
3. Для каждой родительской пары с использованием оператора

скрещивания породить потомство.
4. К порожденным особям применить оператор мутации, внеся случайные искажения.
5. Произвести отбор особей из популяции по значению их фитнесс-функции, применив оператор редукции.
6. Повторять шаги 2–5, пока не выполнится критерий остановки.

Каждый шаг имеет разные реализации. Кроме того, особенности генетических операторов зависят от конкретной формы эволюционных вычислений. Так, особенностью генетических алгоритмов является то, что в них для представления решений используются битовые строки, трактуемые как генотипы (обычно состоящие из единственной хромосомы), и все генетические операторы применяются к битовым строкам. В эволюционных стратегиях операторы применяются к самим решениям, представленным в их естественной форме. Особенность же эволюционного (генетического) программирования состоит в том, что в них рассматривается оптимизация особых объектов — компьютерных программ. Рассмотрим каждый из шагов чуть подробнее на примере генетических алгоритмов (ГА).

1. Генерация начальной популяции обычно производится равномерно по пространству генотипов. Размер популяции — установочный параметр.
2. Выбор родительских пар может осуществляться различными способами. Обычно он включает два этапа: выбор первого родителя и формирование пары. При выборе первого родителя обычно используется один из следующих способов:
   с равной вероятностью выбирается любая особь из имеющейся популяции;
   особь выбирается случайно с вероятностью, пропорциональной значению фитнесс-функции; т. е. в этом случае значение фитнесс-функции сказывается не только на том, какие особи останутся в популяции в результате отбора, но и на том, сколько потомства они произведут.

Выбор второго родителя осуществляется по одному из следующих критериев:

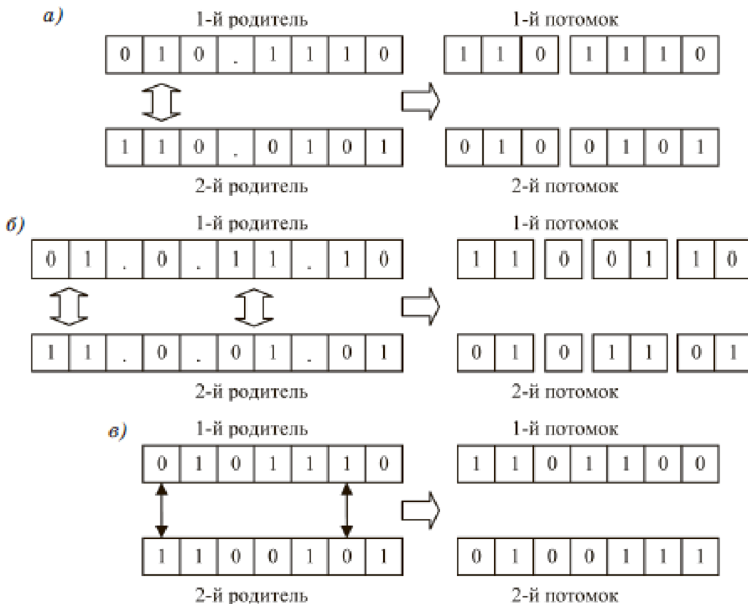
- независимо от уже выбранного родителя (т. е. второй родитель

выбирается абсолютно так же, как и первый); этот вид отбора называется неселективным;
- на основе ближнего родства;
- на основе дальнего родства.

В последних двух случаях выбор одного родителя влияет на выбор другого родителя: с большей вероятностью формируются пары, состоящие из особей, которые больше похожи друг на друга (т. е. ближе находятся в пространстве генотипов) при использовании ближнего родства или меньше похожи при использовании дальнего родства. В генетических алгоритмах в качестве меры близости обычно используется расстояние Хемминга, которое для двух битовых строк вычисляется как число позиций, в которых в двух строках стоят несовпадающие символы (т. е. в одной строке стоит 0, а в другой — 1).

3. **Оператор скрещивания** — это оператор, который определяет, как из генотипов родителей формировать генотипы их потомства. Один из интуитивно очевидных способов заключается в том, чтобы каждый бит генотипа для потомка брать от случайного родителя. Такой способ действительно используется и называется равномерным скрещиванием. Однако в природе гены внутри одной хромосомы являются сцепленными, поэтому случайным (и независимым) образом от родителей берутся целые хромосомы. Также есть механизм, который позволяет рекомбинировать и сцепленным генам. Это механизм кроссинговера, или перекреста хромосом, при котором (гомологичные) хромосомы обмениваются участками.

В ГА этот процесс моделируется так: хромосомы (а как отмечалось выше, обычно в ГА все гены особи располагаются в единственной хромосоме) делятся в некоторой случайной точке и обмениваются этими участками (т. е. все, что идет до этой точки, берется от одного родителя, а все, что после, — от другого). Это одноточечный кроссинговер. В многоточечном кроссинговере таких участков обмена больше. В ГА нередко используется такой оператор скрещивания, при котором формируются генотипы сразу двух потомков, содержащие всю генетическую информацию родителей. Варианты реализации оператора скрещивания представлены ниже на рисунке.



Примеры работы оператора скрещивания: *а* — одноточечного; *б* — многоточечного; *в* — обмена случайными битами

Если бы в ГА геном представлялся в виде нескольких хромосом, то моделировался бы одновременно случайный выбор целых хромосом и рекомбинация генов внутри отдельных хромосом с помощью кроссинговера.

4. В бытовом понимании мутации обычно представляются как случайные искажения генов, и именно так (несмотря на то, что это крайне упрощенное представление) они реализуются в ГА. Действие оператора мутации сводится к случайной замене одного (иногда нескольких) бита генотипа. Настроечный параметр алгоритма — скорость мутации — определяет, как часто эта замена делается. Этот параметр влияет на скорость сходимости и вероятность попадания в локальный экстремум. Естественно, чем чаще мутации, тем медленнее стабилизируется генофонд популяции, но тем меньше шансов, что эволюция «застрянет» в неоптимальном решении.

5. Отбор особей в новую популяцию чаще всего осуществляется в соответствии с одной из двух стратегий:

- пропорционального отбора, при котором вероятность того, что особь останется в следующей популяции, пропорциональна значению ее фитнесс-функции;
- элитного отбора, при котором из популяции отбираются лучшие по значению фитнесс-функции особи, и только они переходят в следующую популяцию.

Формирование новой популяции может осуществляться как на основе потомков и родителей, так и на основе только потомков в зависимости от конкретной реализации.

6. Основные критерии остановки алгоритма базируются либо на числе сменившихся поколений (количестве выполненных итераций), либо на некотором условии стабильности популяции. Заранее сложно предсказать, сколько именно популяций потребуется для сходимости, поэтому этот критерий используется обычно как вспомогательный. Проверка стабильности популяции в общем виде, как правило, требует значительных вычислений, поэтому чаще используется проверка того, что наилучшее по популяции значение фитнесс-функции перестает заметно изменяться от поколения к поколению.

Отличие эволюционных стратегий от генетических алгоритмов заключается в том, что в первых не используются битовые представления. Вместо этого все генетические операторы реализуются в пространстве исходных объектов (или фенотипических признаков) с учетом их структуры. Рассмотрим особенности реализации генетических операторов в эволюционных стратегиях на примере объектов, описаниями которых являются двухкомпонентные векторы вида (x, y), т. е. задача заключается просто в поиске экстремума функции от двух переменных f (x, y).

1. Генерация начальной популяции может осуществляться путем выбора случайных векторов из прямоугольной области $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$, в которой ожидается нахождение экстремума фитнесс-функции. В случае

генетических алгоритмов эта область задается неявно, и она зависит от способа отображения вектора (x, y) в битовую строку.

2. При выборе родителей особенность эволюционных стратегий выражается в способе задания меры родства. В данном случае мерой родства двух особей (x1, y1) и (x2, y2) может служить евклидово расстояние, которое будет заметно отличаться от расстояния Хемминга, используемого в генетических алгоритмах.
3. Результатом скрещивания двух особей в рассматриваемом случае будет являться особь, находящаяся в случайном месте отрезка (x1, y1)—(x2, y2), что, опять же, отличается от результата скрещивания в пространстве генотипов.
4. Результатом мутации особи (x, y) будет являться особь (x + δx, y + δy), где δx, δy — случайные величины, разброс которых определяет скорость мутаций.
5. Операторы отбора и критерии останова в эволюционных стратегиях не имеют особых отличий от тех, которые используются в генетических алгоритмах.

Как видно, реализация генетических операторов на уровне фенотипов допускает более гибкую их настройку, что может оказать помощь в повышении эффективности поиска, но при этом требуются дополнительные усилия со стороны разработчика. При использовании ГА реализация операторов может быть одинаковой для разных задач, для которых нужны только процедуры перевода объектов в битовые строки и обратно. Однако неудачный способ перевода может привести к низкой эффективности ГА.

Оба этих подхода могут применяться при решении самых разнообразных задач — от поиска экстремума функций вещественных переменных до решения изобретательских задач по разработке конструкций технических систем (известны случаи даже патентования и коммерческого использования решений, найденных методами эволюционных вычислений). Они позволяют приближенно решать некоторые NP-полные задачи не хуже, чем методы эвристического программирования, и требуют при этом меньших усилий от разработчика.

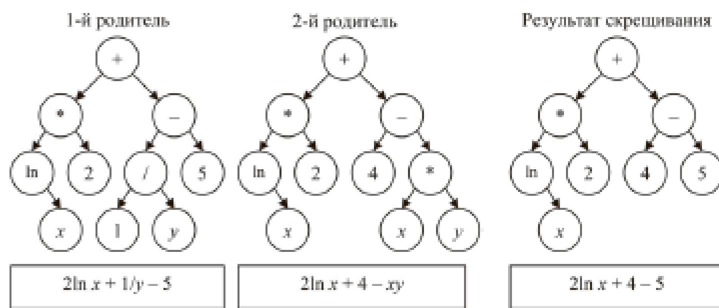
Если оба подхода одинаково применимы на практике, почему в природе используется только один из них? Действительно, эволюционные стратегии в чем-то ближе к концепции не Дарвина, а Ламарка, согласно которой совершенствуются и непосредственно наследуются фенотипические признаки. Эта концепция неплохо работает, когда имеется фиксированный набор фенотипических признаков, для которых можно определить, как их изменения сказываются на фитнесс-функции. Но если оптимизируемые объекты комбинаторно конструируются, перечень их внешних признаков может быть неограниченным и заранее неизвестным. В этом случае применить поиск в пространстве фенотипов не представляется возможным. Именно такая ситуация имеет место в живой природе. Выше мы затрагивали вопрос о сложной связи генов и фенотипических признаков, которые разделяют многие уровни биохимического конструирования, способные породить неограниченный набор внешних признаков. Но за это приходится «платить» тем, что обратный переход от фенотипа к генотипу становится, по крайней мере, NP-полной задачей высокой размерности. Как результат, наследование приобретенных признаков в стиле Ламарка оказывается почти (но не полностью) невозможным.

При использовании генетических алгоритмов разработчик, однако, обычно начинает с фиксированных фенотипических признаков, данных в рамках решаемой оптимизационной задачи, и для них выдумывает способ кодирования в геноме одновременно с обратным преобразованием. Поскольку и в ГА фенотипы оказываются первичными, эволюционные стратегии имеют не меньше прав на применение.

Существует, однако, класс объектов, эволюционная оптимизация которых затруднительна в пространстве их «фенотипических признаков», — это алгоритмы, или программы. Не существует какого-нибудь простого отображения между желаемым выводом программы и ее кодом (выполнение такого отображения — это NP-полная или алгоритмически неразрешимая, в зависимости от деталей постановки, задача). Не удивительно, что попытки автоматического построения программ в рамках эволюционных вычислений выделяются в самостоятельное направление — генетическое (эволюционное) программирование.

Здесь могут применяться аналоги как генетических алгоритмов, так и эволюционных стратегий. В обоих случаях перебираются программы, а не совершаемые ими действия, поэтому различия между ними сводятся к тому, в каком виде представлять код программы — в виде бинарной строки или, скажем, в виде графа. Из-за этого и трудности у этих методов общие, хотя и выражающиеся немного по-разному, например, как реализовать операции скрещивания и мутации, чтобы после их выполнения получались корректные программы (или как закодировать программы в форме бинарной строки для достижения той же цели).

Чаще используются не программы на каком-то обычном языке программирования, а разрабатывается некий язык со специальным синтаксисом, упрощающий работу в рамках эволюционных методов. На практике этот язык зачастую не является алгоритмически полным. В простейшем случае (исходно рассмотренном Фогелем) структура программы фиксирована, а оптимизируются только ее параметры. В современных методах эволюционного программирования рассматриваются представления программ в виде деревьев. При этом в результате скрещивания одно поддерево заменяется другим (все родительские узлы у них должны совпадать). Пример простой (с точки зрения представления программы), но часто встречающейся задачи, — это подбор математического выражения под известные результаты вычислений. На рисунке ниже представлен один из возможных результатов скрещивания двух программ-выражений.



Скрещивание выражений

Такой способ поиска в пространстве программ хорошо сочетается с

подходом к индуктивному выводу на основе алгоритмической сложности. Действительно, если у нас есть некоторый набор данных, для которого нужно построить модель, мы вполне можем применить эволюционное (или генетическое) программирование, при котором фитнесс-функция каждой особи-программы будет оцениваться по критерию минимальной длины описания. Не даст ли это универсального решения задачи индуктивного вывода? Такая возможность, в числе прочих, исследовалась Р. Соломоновым. К сожалению, существующие методы эволюционных вычислений не справляются с NP-полнотой данной проблемы и позволяют получить приемлемое решение лишь для достаточно простых случаев.

Пожалуй, в приложении к проблемам машинного обучения наиболее популярным оказалось применение **эволюционного программирования при обучении нейронных сетей**, что в большей степени используется не в индуктивном выводе, а при синтезе систем управления, в том числе в «Искусственной жизни» и «Адаптивном поведении». Наиболее простым в реализации является случай, когда структура нейронной сети фиксирована и требуется лишь настроить веса ее связей. В геноме тогда кодируются только веса связей, так что длина генотипов оказывается одинаковой у всех особей и не возникает опасности порождения некорректного генетического кода в результате применения генетических операторов. Это, однако, приводит к сильному ограничению пространства поиска. Поиск на менее ограниченном множестве программ является все еще плохо изученной проблемой.

Одна из причин, почему разработчики пытаются ограничиться геномом фиксированного размера, заключается в том, что в противном случае не вполне ясно, как реализовывать оператор скрещивания. Стоит отметить, что фиксированная структура генома приводит к тому, что в эволюционных вычислениях обычно скрещиваются все особи, т. е. эволюционирует один вид. Возможность изменения структуры генома в ходе эволюционных вычислений с соответствующим расщеплением видов практически не исследована.

Порождение ИНС методами эволюционных вычислений лишь заменяет традиционные методы их обучения. В этой связи гораздо больший интерес может представлять порождение нейроглиальных сетей, ведь в

применении к ИНГС эволюционные вычисления превращаются в метаобучение — автоматическое построение самих алгоритмов обучения. Пока, однако, в этом направлении сделаны лишь первые шаги.

Альтернативой эволюционного синтеза нейросетевых систем управления является порождение управляющих конечных автоматов (как классических, так и на основе теории нечетких множеств), где также достигнуты определенные успехи.

Итак, **эволюционные вычисления неплохо себя зарекомендовали в качестве методов поиска как при решении задач классического ИИ (т. е. при решении корректно поставленных NP-полных задач), так и при решении задач машинного обучения, хотя и не дали их полного решения.**

Почему же методы эволюционных вычислений оказываются достаточно успешными для решения проблем поиска? Эволюционные стратегии в применении к поиску экстремума функции от вещественных переменных очень похожи на стохастический градиентный спуск за исключением того, что в них происходит «скрещивание» решений. С генетическими алгоритмами ситуация чуть сложнее, так как в них небольшие мутации генов могут привести к значительному изменению «фенотипа» (примером может служить изменение старшего бита в двоичной записи числа). Но и в ГА основной особенностью является скрещивание, благодаря которому новые решения строятся как фрагменты имеющихся решений. Если задача разбивается на подзадачи и решению каждой подзадачи соответствует отдельный участок генома, то за счет скрещивания подзадачи могут решаться независимо. Действительно, если решение какой-то подзадачи улучшает общее решение независимо от других подзадач, то в генофонде популяции соответствующий участок генома быстро стабилизируется (его вариативность сильно уменьшится). Еще в большей степени этому будет способствовать использование не равномерного скрещивания, а кроссинговера, при котором потомкам передаются длинные фрагменты генотипов родителей. Кроссинговер будет работать хорошо, только если решения отдельных подзадач локализованы в геноме.

Итак, мы видим, что ГА дополнены одним, но очень мощным приемом,

благодаря которому решаемая задача «мягко» (но не адаптивно) разбивается на подзадачи. Следует, однако, иметь в виду, что этот прием не вполне универсален. Если участки генома не соответствуют фрагментам решений почти независимых подзадач (в силу особенностей решаемой NP-полной задачи или в силу неудачного выбора способа кодирования решения в геноме), то ГА будут смесью полного перебора и градиентного спуска, причем реализованного не самым эффективным образом. Отдельные участки генома не будут стабилизироваться независимо друг от друга, и найти решение удастся, только если все гены одновременно рекомбинируют нужным образом, вероятность чего будет очень низка.

Прием, который позволяет эффективно сокращать перебор для многих задач, но в общем случае не гарантирует нахождения оптимального решения, принято называть метаэвристикой. В отличие от простой эвристики метаэвристика не является предметно-специфичной, что делает ее широко применимой, но во многих случаях недостаточной. К примеру, сами по себе ГА не могут использоваться при выборе хода в шахматах, для которых классическое эвристическое программирование пока остается хоть и весьма трудоемким, но практически единственным подходящим средством. Если же в какой-то NP-полной задаче части решения являются почти независимыми (как, например, в задаче коммивояжера), то единственная метаэвристика, заложенная в ГА, может оказаться достаточной для получения приемлемого решения, благодаря чему не нужно будет разрабатывать индивидуальный (под данную задачу) алгоритм обхода дерева варианта со сложными предметно-специфичными эвристиками.

Часто говорят, что эволюционные вычисления весьма эффективны, поскольку они заимствуют основные идеи из естественной эволюции. Но зададимся наивным вопросом: почему генетическое программирование не позволяет создать ИИ? Выше мы не напрасно останавливались на вопросах устройства генов: теперь мы видим, что ГА и аналогичные им методы столь же проще реальных генетических механизмов, сколь и ИНС проще биологических нейронов. Вместе с тем ГА, как и методы обучения ИНС, недостаточны для работы в алгоритмически полном пространстве.

Несмотря на некоторые положительные свойства, методы

эволюционных вычислений оказываются вовсе не универсальными. В связи с этим, если мы рассмотрим эволюционные вычисления как методы поиска в пространстве решений, то становится понятным, почему с их помощью не так просто автоматически получить ИИ. По сути, эти методы сами будут составлять интеллектуальную систему, перед которой ставится задача поиска программы ИИ. Нет ли здесь противоречия? Не получается ли так, что нам уже нужно иметь ИИ, чтобы компьютер смог сам его придумать? Принципиального противоречия здесь нет: более «глупая» программа может вывести более «умную» посредством «грубой силы», т. е. используя очень интенсивный перебор. Конечно, чем глупее эволюционная программа, тем больше грубой силы ей придется приложить. Полным перебором задачу построения ИИ или даже игры в шахматы на практике не решить. Естественно, недостаточно и нескольких простых метаэвристик. Такой взгляд показывает наивность попыток с помощью простой искусственной эволюции вывести интеллектуальные программы. Но как это удалось сделать естественной эволюции?

1.7. Описание генетического алгоритма

Задача кодируется таким образом, чтобы её решение могло быть представлено в виде вектора («хромосома»). Случайным образом создаётся некоторое количество начальных векторов («начальная популяция»). Они оцениваются с использованием «функции приспособленности», в результате чего каждому вектору присваивается определённое значение («приспособленность»), которое определяет вероятность выживания организма, представленного данным вектором. После этого с использованием полученных значений приспособленности выбираются вектора (*селекция*), допущенные к «скрещиванию». К этим векторам применяются «генетические операторы» (в большинстве случаев «скрещивание» - crossover и «мутация» - mutation), создавая таким образом следующее «поколение». Особи следующего поколения также оцениваются, затем производится селекция, применяются генетические операторы и т. д. Так моделируется «эволюционный процесс», продолжающийся несколько жизненных циклов (*поколений*), пока не будет выполнен критерий

остановки алгоритма. Таким критерием может быть:

- нахождение глобального, либо субоптимального решения;
- исчерпание числа поколений, отпущенных на эволюцию;
- исчерпание времени, отпущенного на эволюцию.

Генетические алгоритмы служат, главным образом, для поиска решений в очень больших, сложных пространствах поиска.

Таким образом, можно выделить следующие этапы генетического алгоритма:

1. Создание начальной популяции
2. Вычисление функций приспособленности для особей популяции (оценивание)

- (Начало цикла)

    1. Выбор индивидов из текущей популяции (селекция)
    2. Скрещивание и\или мутация
    3. Вычисление функций приспособленности для всех особей
    4. Формирование нового поколения
    5. Если выполняются условия остального, то (конец цикла), иначе (начало цикла).

Создание начальной популяции

Перед первым шагом нужно случайным образом создать некую начальную популяцию; даже если она окажется совершенно неконкурентоспособной, генетический алгоритм все равно достаточно быстро переведет ее в жизнеспособную популяцию. Таким образом, на первом шаге можно особенно не стараться сделать слишком уж приспособленных особей, достаточно, чтобы они соответствовали формату особей популяции, и на них можно было подсчитать функцию приспособленности (Fitness). Итогом первого шага является популяция

H, состоящая из N особей.

Отбор

На этапе отбора нужно из всей популяции выбрать определенную ее долю, которая останется "в живых" на этом этапе эволюции. Есть разные способы проводить отбор. Вероятность выживания особи h должна зависеть от значения функции приспособленности Fitness (h). Сама доля выживших s обычно является параметром генетического алгоритма, и ее просто задают заранее. По итогам отбора из N особей популяции H должны остаться sN особей, которые войдут в итоговую популяцию H'. Остальные особи погибают.

Размножение

Размножение в генетических алгоритмах обычно половое - чтобы произвести потомка, нужны несколько родителей; обычно, конечно, нужны ровно два. Размножение в разных алгоритмах определяется по-разному - оно, конечно, зависит от представления данных. Главное требование к размножению - чтобы потомок или потомки имели возможность унаследовать черты обоих родителей, "смешав" их каким-либо достаточно разумным способом. Вообще говоря, для того чтобы провести операцию размножения, нужно выбрать (1-s)p/2 пар гипотез из H и провести с ними размножение, получив по два потомка от каждой пары (если размножение определено так, чтобы давать одного потомка, нужно выбрать (1 - s)p пар), и добавить этих потомков в H'. В результате H' будет состоять из N особей. Почему особи для размножения обычно выбираются из всей популяции H, а не из выживших на первом шаге элементов H_0 (хотя последний вариант тоже имеет право на существование)? Дело в том, что главный бич многих генетических алгоритмов - недостаток разнообразия (diversity) в особях. Достаточно быстро выделяется один-единственный генотип, который представляет собой локальный максимум, а затем все элементы популяции проигрывают ему отбор, и вся популяция "забивается" копиями этой особи. Есть разные способы борьбы с таким нежелательным эффектом; один из них - выбор для размножения не самых приспособленных, но вообще всех особей.

Мутации

К мутациям относится все то же самое, что и к размножению: есть некоторая доля мутантов m, являющаяся параметром генетического алгоритма, и на шаге мутаций нужно выбрать mN особей, а затем изменить их в соответствии с заранее определенными операциями мутации.

Пример реализации генетического алгоритма на языке C++

Поиск в одномерном пространстве, без скрещивания.

```
#include <iostream.h>
#include <algorithm.h>
#include <numeric.h>
using namespace std;
int main()
{
   const int N = 1000;
   int a[N];
   //заполняем нулями
   fill(a, a+N, 0);
   for (;;)
   {
      //мутация в случайную сторону каждого элемента:
      for (int i = 0; i < N; ++i)
         if (rand()%2 == 1)
            a[i] += 1;
         else
            a[i] -= 1;
      //теперь выбираем лучших, отсортировав по возрастанию...
      sort(a, a+N);
      //... и тогда лучшие окажутся во второй половине массива.
      //скопируем лучших в первую половину, куда они оставили потомство, а первые умерли:
      copy(a+N/2, a+N,    a /*куда*/);
      //теперь посмотрим на среднее состояние популяции. Как видим, оно всё лучше и лучше.
      cout << accumulate(a, a+N, 0) / N << endl;
```

}

2. Классический генетический алгоритм и его релизация

2.1. Функция приспособленности и кодирование решений

Родителем теории генетических алгоритмов (ГА) считается Холланд (J. Holland), чья работа «Adaptation in Natural and Artificial Systems» (1975), стала классикой в этой области. В ней Холланд впервые ввел термин «генетический алгоритм». Сейчас описанный там алгоритм называют «классический ГА» (иногда «канонический ГА», canonical GA), а понятие «генетические алгоритмы» стало очень широким, и зачастую к ним относятся алгоритмы, сильно отличающиеся от классического ГА.

Ученики Холланда Кеннет Де Йонг (Kenneth De Jong) и Дэвид Голдберг (David E. Goldberg) внесли огромный вклад в развитие ГА. На книгу Голдберга «Genetic algorithms in search optimization and machine learning» (1989), ссылаются авторы практически каждой работы по этой теме.

Как уже было сказано выше, генетические алгоритмы работают по аналогии с природой. Они оперируют с совокупностью «особей», представляющих собой строки, каждая из которых кодирует одно из решений задачи. Приспособленность особи оценивается с помощью специальной функции. Наиболее приспособленные получают шанс скрещиваться и давать потомство. Наихудшие особи удаляются и не дают потомства. Таким образом, приспособленность нового поколения в среднем выше предыдущего.

Итак, пусть перед нами стоит задача оптимизации. Варьируя некоторые параметры, к примеру, если решать задачу размещения, координаты размещаемых элементов, нужно найти такую их комбинацию, чтобы минимизировать занимаемую площадь. Такую задачу можно переформулировать как задачу нахождения максимума некоторой функции $f(x_1, x_2, \ldots, x_n)$. Эта функция называется *функцией приспособленности* (fitness function) и используется для вычисления приспособленности особей. Она должна принимать неотрицательные значения, а область определения параметров должна быть ограничена.

Если нам, к примеру, требуется найти минимум некоторой функции, то достаточно перенести область ее значений на положительную область, а затем инвертировать. Таким образом, максимум новой функции будет соответствовать минимуму исходной.

В генетических алгоритмах никак не используются такие свойства функции, как непрерывность, дифференцируемость и т. д. Она подразумевается как устройство (*blackbox*), которое на вход получает параметры, а на выход выводит результат.

Теперь обратимся к кодировке набора параметров. Закодируем каждый параметр строкой битов. Если он принимает непрерывное множество значений, то выберем длину строки в соответствии с требуемой точностью. Таким образом, параметр сможет принимать только дискретные значения (этих значений будет степень 2), в некотором заданном диапазоне.

Например, пусть переменная x_k принадлежит отрезку $[x_{\min}, x_{\max}]$. Закодируем ее бинарной строкой из l битов. Тогда строка s_k обозначает следующее значение переменной x_k:

$$x_k = x_{\min} + s_k(x_{\max} - x_{\min})/2^l$$

если в формуле s_k обозначает значение бинарного числа, кодируемого этой строкой.

Если же некоторый параметр принимает конечное количество значений, к примеру, целые от 0 до 1000, то закодируем его бинарной строкой

достаточной длины, в данном случае 10. Лишние 23 строки могут повторять уже закодированные значения параметра, либо они могут быть доопределены в функции приспособленности как «худшие», т. е. если параметр кодируется одной из этих строк, то функция принимает заведомо наименьшее значение.

Итак, мы определили для каждого параметра строку, кодирующую его. Особью будет называться строка, являющаяся конкатенацией строк всего упорядоченного набора параметров:

```
 101100 11001011 01101100 1100 1 11101
|  x1  |   x2   |        |    | |  xn |
```

Приспособленность особи высчитывается следующим образом: строка разбивается на подстроки, кодирующие параметры. Затем для каждой подстроки высчитывается соответствующее ей значение параметра, после чего приспособленность особи получается как значение функции приспособленности от полученного набора.

Вообще говоря, от конкретной задачи зависят только такие параметры ГА, как функция приспособленности и кодирование решений. Остальные шаги для всех задач производятся одинаково, в этом проявляется универсальность ГА.

Постановка задачи

Пусть перед нами стоит задача оптимизации, например:

- Задача наилучшего приближения
    - Если рассматривать систему n линейных уравнений с m неизвестными

      $$Ax = b$$

      в случае, когда она переопределена ($n > m$), то иногда оказывается естественной задача о нахождении вектора x, который "удовлетворяет этой системе наилучшим

образом", т. е. из всех "не решений" является лучшим.

- Задача о рационе.
  - Пусть имеется n различных пищевых продуктов, содержащих m различных питательных веществ. Обозначим через a_{ij} содержание (долю) j-го питательного вещества в i-ом продукте, через b_j — суточную потребность организма в j-ом питательном веществе, через c_i — стоимость единицы i-го продукта. Требуется составить суточный рацион питания минимальной стоимости, удовлетворяющий потребность во всех питательных веществах
- Транспортная задача.
  - Эта задача — классическая задача линейного программирования. К ней сводятся многие оптимизационные задачи. Формулируется она так. На m складах находится груз, который нужно развезти n потребителям. Пусть a_i ($i = 1, ..., n$) — количество груза на i-ом складе, а b_j ($j = 1, ..., m$) — потребность в грузе j-го потребителя, c_{ij} — стоимость перевозки единицы груза с i-го склада j-му потребителю. Требуется минимизировать стоимость перевозок.
- Задачи о распределении ресурсов.
  - Общий смысл таких задач — распределить ограниченный ресурс между потребителями оптимальным образом. Рассмотрим простейший пример — *задачу о режиме работы энергосистемы.* Пусть m электростанций питают одну нагрузку мощности p. Обозначим через x_j активную мощность, генерируемую j-ой электростанцией. Техническими условиями определяются возможный минимум m_j и максимум M_j вырабатываемой j-ой электростанцией мощности. Допустим затраты на генерацию мощности x на j-ой электростанции равны $e_j(x)$. Требуется сгенерировать требуемую мощность p при минимальных затратах.

Переформулируем задачу оптимизации как задачу нахождения максимума некоторой функции $f(x_1, x_2, ..., x_n)$, называемой

функцией приспособленности (fitness function). Она должна принимать неотрицательные значения на ограниченной области определения (для того, чтобы мы могли для каждой особи считать её приспособленность, которая не может быть отрицательной), при этом совершенно не требуются непрерывность и дифференцируемость.

Каждый параметр функции приспособленности кодируется строкой битов.

Особью будет называться строка, являющаяся конкатенацией строк упорядоченного набора параметров:

1010 10110 101 … 10101

| x1 | x2 | x3 | …| xn |

Универсальность ГА заключается в том, что от конкретной задачи зависят только такие параметры, как функция приспособленности и кодирование решений. Остальные шаги для всех задач производятся одинаково.

2.2. Классический генетический алгоритм

Основной (классический) генетический алгоритм (также называемый элементарным или простым генетическим алгоритмом) состоит из следующих шагов:
1) инициализация, или выбор исходной популяции хромосом;
2) оценка приспособленности хромосом в популяции;
3) проверка условия остановки алгоритма;
4) селекция хромосом;
5) применение генетических операторов;
6) формирование новой популяции;
7) выбор «наилучшей» хромосомы.
Блок-схема основного генетического алгоритма изображена на рис. 2.1.

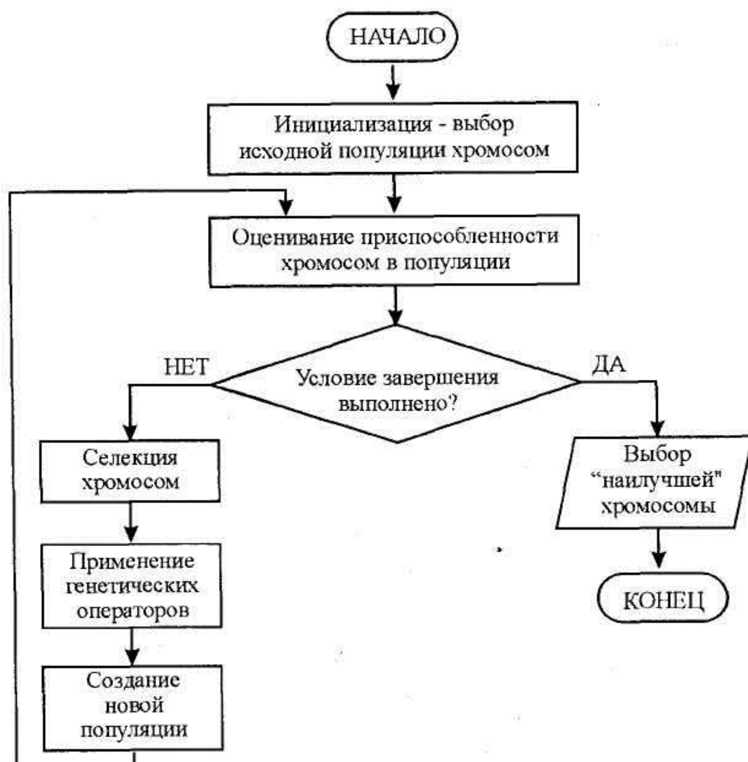


Рис.2.1. Блок-схема генетического алгоритма.

Рассмотрим конкретные этапы этого алгоритма более подробно с использованием дополнительных подробностей, представленных на рис.2.2.

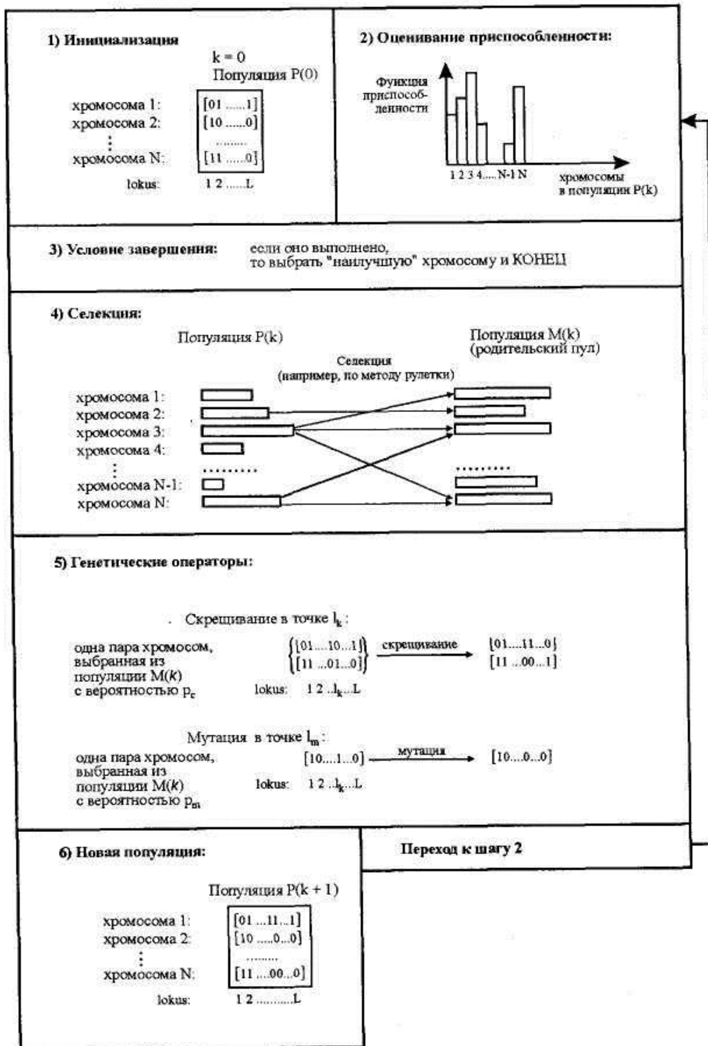


Рис.2.2. Схема выполнения генетического алгоритма.

Инициализация, т.е. формирование исходной популяции, заключается в случайном выборе заданного количества хромосом (особей), представляемых двоичными последовательностями фиксированной

длины.

Оценивание приспособленности хромосом в популяции состоит в расчете функции приспособленности для каждой хромосомы этой популяции. Чем больше значение этой функции, тем выше «качество» хромосомы. Форма функции приспособленности зависит от характера решаемой задачи. Предполагается, что функция приспособленности всегда принимает неотрицательные значения и, кроме того, что для решения оптимизационной задачи требуется максимизировать эту функцию. Если исходная форма функции приспособленности не удовлетворяет этим условиям, то выполняется соответствующее преобразование (например, задачу минимизации функции можно легко свести к задаче максимизации).

Проверка условия остановки алгоритма. Определение условия остановки генетического алгоритма зависит от его конкретного применения. В оптимизационных задачах, если известно максимальное (или минимальное) значение функции приспособленности, то остановка алгоритма может произойти после достижения ожидаемого оптимального значения, возможно - с заданной точностью. Остановка алгоритма также может произойти в случае, когда его выполнение не приводит к улучшению уже достигнутого значения. Алгоритм может быть остановлен по истечении определенного времени выполнения либо после выполнения заданного количества итераций. Если условие остановки выполнено, то производится переход к завершающему этапу выбора «наилучшей» хромосомы. В противном случае на следующем шаге выполняется селекция.

Селекция хромосом заключается в выборе (по расчитанным на втором этапе значениям функции приспособленности) тех хромосом, которые будут участвовать в создании потомков для следующей популяции, т.е. для очередного поколения. Такой выбор производится согласно принципу естественного отбора, по которому наибольшие шансы на участие в создании новых особей имеют хромосомы с наибольшими значениями функции приспособленности. Существуют различные методы селекции. Наиболее популярным считается так называемый *метод рулетки (roulette wheel selection),* который свое название получил по аналогии с известной азартной игрой. Каждой хромосоме может быть сопоставлен сектор колеса рулетки, величина которого устанавливается пропорциональной значению функции приспособленности данной хромосомы. Поэтому чем больше значение

функции приспособленности, тем больше сектор на колесе рулетки. Все колесо рулетки соответствует сумме значений функции приспособленности всех хромосом рассматриваемой популяции. Каждой хромосоме, обозначаемой ch_i для i=1,2,..., N (где N обозначает численность популяции) соответствует сектор колеса $v(ch_i)$, выраженный в процентах согласно формуле

$$v(ch_i) = p_s(ch_i)100\% \,, \quad (2.1)$$

где

$$p_s(ch_i) = \frac{F(ch_i)}{\sum_{i=1}^{N} F(ch_i)} \quad (2.2)$$

причем $F(ch_i)$ - значение функции приспособленности хромосомы ch_i, а $p_s(ch_i)$ - *вероятность селекции* хромосомы ch_i. Селекция хромосомы может быть представлена как результат поворота колеса рулетки, поскольку «выигравшая» (т.е. выбранная) хромосома относится к выпавшему сектору этого колеса. Очевидно, что чем больше сектор, тем больше вероятность «победы» соответствующей хромосомы. Поэтому вероятность выбора данной хромосомы оказывается пропорциональной значению ее функции приспособленности. Если всю окружность колеса рулетки представить в виде цифрового интервала [0, 100], то выбор хромосомы можно отождествить с выбором числа из интервала [a, b], где a и b обозначают соответственно начало и окончание фрагмента окружности, соответствующего этому сектору колеса; очевидно, что $0 \le a < b \le 100$. В этом случае выбор с помощью колеса рулетки сводится к выбору числа из интервала [0, 100], которое соответствует конкретной точке на окружности колеса. Другие методы селекции будут рассматриваться дальше.

В результате процесса селекции создается *родительская популяция,* также называемая *родительским пулом (mating pool)* с численностью N, равной численности текущей популяции.

Применение генетических операторов к хромосомам, отобранным с помощью селекции, приводит к формированию новой популяции потомков от созданной на предыдущем шаге родительской популяции.

В классическом генетическом алгоритме применяются два основных генетических оператора: *оператор скрещивания (crossover)* и *оператор мутации (mutation).* Однако следует отметить, что оператор мутации играет явно второстепенную роль по сравнению с оператором скрещивания. Это означает, что скрещивание в классическом

гянетическом алгоритме производится практически всегда, тогда как мутация - достаточно редко. Вероятность скрещивания, как правило, достаточно велика (обычно $0,5 \le p_c \le 1$), тогда как вероятность мутации устанавливается весьма малой (чаще всего $0 \le p_m \le 0,1$). Это следует из аналогии с миром живых организмов, где мутации происходят чрезвычайно редко.

В генетическом алгоритме мутация хромосом может выполняться на популяции родителей перед скрещиванием либо на популяции потомков, образованных в результате скрещивания.

Оператор скрещивания. На первом этапе скрещивания выбираются пары хромосом из родительской популяции (родительского пула). Это временная популяция, состоящая из хромосом, отобранных в результате селекции и предназначенных для дальнейших преобразований операторами скрещивания и мутации с целью формирования новой популяции потомков. На данном этапе хромосомы из родительской популяции объединяются в пары. Это производится случайным способом в соответствии с вероятностью скрещивания p_c. Далее для каждой пары отобранных таким образом родителей разыгрывается позиция гена *(локус)* в хромосоме, определяющая так называемую *точку скрещивания.* Если хромосома каждого из родителей состоит из L генов, то очевидно, что точка скрещивания k представляет собой натуральное число, меньшее L. Поэтому фиксация точки скрещивания сводится к случайному выбору числа из интервала [1, L-1]. В результате скрещивания пары родительских хромосом получается следующая пара потомков:

1) потомок, хромосома которого на позициях от 1 до k состоит из генов первого родителя, а на позициях от $k+1$ до L - из генов второго родителя;

2) потомок, хромосома которого на позициях от 1 до k состоит из генов второго родителя, а на позициях от $k+1$ до L - из генов первого родителя.

Действие оператора скрещивания будет проиллюстрировано примерами.

Оператор мутации с вероятностью p_m изменяет значение гена в хромосоме на противоположное (т.е. с 0 на 1 или обратно). Например, если в хромосоме [100110101010] мутации подвергается ген на позиции 7, то его значение, равное 1, изменяется на 0, что приводит к

образованию хромосомы [100110001010]. Как уже упоминалось выше, вероятность мутации обычно очень мала, и именно от нее зависит, будет данный ген мутировать или нет. Вероятность p_m мутации может эмулироваться, например, случайным выбором числа из интервала [0, 1] для каждого гена и отбором для выполнения этой операции тех генов, для которых разыгранное число оказывается меньшим или равным значению p_m.

Формирование новой популяции. Хромосомы, полученные в результате применения генетических операторов к хромосомам временной родительской популяции, включаются в состав новой популяции. Она становится так называемой текущей популяцией для данной итерации генетического алгоритма. На каждой очередной итерации рассчитываются значения функции приспособленности для всех хромосом этой популяции, после чего проверяется условие остановки алгоритма и либо фиксируется результат в виде хромосомы с наибольшим значением функции приспособленности, либо осуществляется переход к следующему шагу генетического алгоритма, т.е. к селекции. В классическом генетическом алгоритме вся предшествующая популяция хромосом замещается новой популяцией потомков, имеющей ту же численность.

Выбор «наилучшей» хромосомы. Если условие остановки алгоритма выполнено, то следует вывести результат работы, т.е. представить искомое решение задачи. Лучшим решением считается хромосома с наибольшим значением функции приспособленности.

В завершение следует признать, что генетические алгоритмы унаследовали свойства естественного эволюционного процесса, состоящие в генетических изменениях популяций организмов с течением времени. Главный фактор эволюции - это естественный отбор (т.е. природная селекция), который приводит к тому, что среди генетически различающихся особей одной и той же популяции выживают и оставляют потомство только наиболее приспособленные к окружающей среде. В генетических алгоритмах также выделяется этап селекции, на котором из текущей популяции выбираются и включаются в родительскую популяцию особи, имеющие наибольшие значения функции приспособленности. На следующем этапе, который иногда называется *эволюцией,* применяются генетические операторы скрещивания и мутации, выполняющие рекомбинацию генов в хромосомах.

Операция скрещивания заключается в обмене фрагментами цепочек между двумя родительскими хромосомами. Пары родителей для скрещивания выбираются из родительского пула случайным образом

так, чтобы вероятность выбора конкретной хромосомы для скрещивания была равна вероятности p_c. Например, если в качестве родителей случайным образом выбираются две хромосомы из родительской популяции численностью N, то $p_c = 2N$. Аналогично, если из родительской популяции численностью N выбирается $2z$ хромосом ($z \le N/2$), которые образуют z пар родителей, то $p_c=2z/N$. Обратим внимание, что если все хромосомы текущей популяции объединены в пары до скрещивания, то $p_c =1$. После операции скрещивания родители в родительской популяции замещаются их потомками.
Операция мутации изменяет значения генов в хромосомах с заданной вероятностью p_m способом, представленным при описании соответствующего оператора. Это приводит к инвертированию значений отобранных генов с 0 на 1 и обратно. Значение p_m, как правило, очень мало, поэтому мутации подвергается лишь небольшое количество генов. Скрещивание - это ключевой оператор генетических алгоритмов, определяющий их возможности и эффективность. Мутация играет более ограниченную роль. Она вводит в популяцию некоторое разнообразие и предупреждает потери, которые могли бы произойти вследствие исключения какого-нибудь значимого гена в результате скрещивания.
Основной (классический) генетический алгоритм известен в литературе в качестве инструмента, в котором выделяются три вида операций: *репродукции, скрещивания* и *мутации*. Термины *селекция* и *репродукция* в данном контексте используются в качестве синонимов. При этом репродукция в данном случае связывается скорее с созданием копий хромосом родительского пула, тогда как более распространенное содержание этого понятия обозначает процессформирования новых особей, происходящих от конкретных родителей . Если мы принимаем такое толкование, то операторы скрещивания и мутации могут считаться операторами репродукции, а селекция - отбором особей (хромосом) для репродукции.

Генетические операторы

Частично о генетических операторах мы уже говорили. Остановимся на их описании более подробно. Генетические операторы необходимы, чтобы применить принципы наследственности и изменчивости к виртуальной популяции. Помимо отличительных особенностей, о которых будет рассказано ниже, у них есть такое свойство как *вероятность*. Т.е. описываемые операторы не обязательно применяются ко всем скрещиваемым особям, что вносит

дополнительный элемент неопределенности в процесс поиска решения. В данном случае, неопределенность не подразумевает негативный фактор, а является своеобразной "степенью свободы" работы генетического алгоритма.

Здесь описываются только два самых распространенных и необходимых оператора. Существуют и другие генетические операторы (например, *инверсия*), но они применяются очень редко и поэтому мы о них говорить не будем.

Оператор кроссинговера

Оператор кроссинговера (crossover operator), также называемый кроссовером, является основным генетическим оператором, за счет которого производится обмен генетическим материалом между особями. Оператор моделирует процесс скрещивания особей. Пусть имеются две родительские особи с хромосомами $X=\{x_i, i=1,L\}$ и $Y=\{y_i, i=1,L\}$. Случайным образом определяется точка внутри хромосомы, в которой обе хромосомы делятся на две части и обмениваются ими. Назовем эту точку точкой разрыва. Вообще говоря, в англоязычной литературе она называется *точкой кроссинговера* (crossover point), просто, точка разрыва более образное название и к тому же позволяет в некоторых случаях избежать тавтологии. Описанный процесс изображен на рис.2.3.

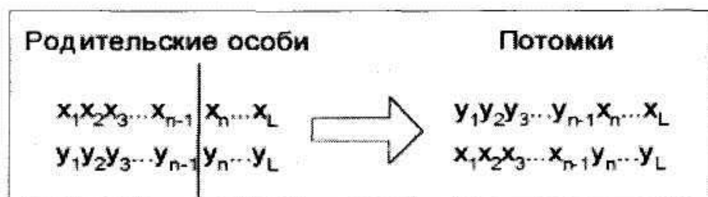


Рис.2.3. Кроссинговер

Данный тип кроссинговера называется *одноточечным*, т.к. при нем родительские хромосомы разрезаются только в одной случайной точке. Также существует *2-х* и *n-точечный* операторы кроссинговера. В 2-х точечном кроссинговере точек разрыва 2, а n-точечный кроссинговер

является своебразным обобщением 1- и 2-точечного кроссинговеров для n > 2.

Кроме описанных типов кроссинговера есть ещё *однородный кроссинговер*. Его особенность заключается в том, что значение каждого бита в хромосоме потомка определяется случаным образом из соответствующих битов родителей. Для этого вводится некоторая величина $0<p_0<1$, и если случайное число больше p_0, то на n-ю позицию первого потомка попадает n-й бит первого родителя, а на n-ю позицию второго - n-й бит второго родителя. В противном случае к первому потомку попадает бит второго родителя, а ко второму - первого. Такая операция проводится для всех битов хромосомы.

Вероятность кроссинговера самая высокая среди генетических операторов и равна обычно 60% и больше.

Оператор мутации

Оператор мутации (mutation operator) необходим для "выбивания" популяции из локального экстремума и способствует защите от преждевременной сходимости. Достигаются это за счет того, что инвертируется случайно выбранный бит в хромосоме, что и показано на рис.2.4.

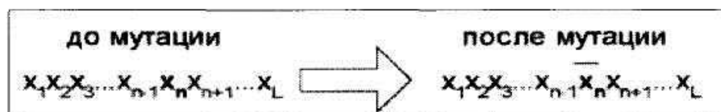


Рис.2.4. Мутация

Так же как и кроссинговер, мутация проводится не только по одной случайной точке. Можно выбирать некоторое количество точек в хромосоме для инверсии, причем их число также может быть случайным. Также можно инвертировать сразу некоторую группу подряд идущих точек. Вероятность мутации значительно меньше вероятности кроссинговера и редко превышает 1%. Среди рекомендаций по выбору вероятности мутации нередко можно встретить варианты 1/L или 1/N, где L - длина хромосомы, N - размер

популяции.

Необходимо также отметить, что некоторые авторы считают, что оператор мутации является основным поисковым оператором и известны алгоритмы, не использующих других операторов (кроссинговер, инверсия и т.д.) кроме мутации.

Пример реализации

Обычно при реализации ГА к скрещиваемым особям сначала применяют оператор кроссинговера, а потом оператор мутации, хотя возможны варианты. Например, оператор мутации можно применять только если к данной паре родительских особей не был применен оператор кроссинговера. В принципе, никто не мешает вообще не использовать вероятность проведения данного генетического оператора и применять кроссинговер ко всем отобранным особям, а мутацию к каждому 100 потомку.

Ниже приведен пример подпрограмм, реализующих операторы кроссинговера и мутации на языке С++ для популяции, содержащей 16-разрядные хромосомы. Размер популяции: 50 особей. Популяция здесь представлена условно .

```
#include <stdlib.h>
unsigned short MutationMask[] = {0x1, 0x2, 0x4, 0x8,
                                0x10, 0x20, 0x40, 0x80,
                                0x100, 0x200, 0x400, 0x800,
                                0x1000, 0x2000, 0x4000, 0x8000};

unsigned short Inds[50],     // Популяция
               SelInds[50];  // Особи отобранные для скрещивания

// оператор 1-точечного кроссинговера
void Cross (int p1, p2, c1, c2) {
  unsigned short left, right;

left =(unsigned short)(15.0 * (double)rand()/(double)RAND_MAX+ 1);
  left = ((unsigned short)0xffff>>left)<<left;
```

```
  right = (unsigned short)0xffff ^ left;

  Inds[c1] = (SelInds[p1] & left) | (SelInds[p2] & right);
  Inds[c2] = (SelInds[p2] & left) | (SelInds[p1] & right);
}

// оператор точечной мутации
void Mutate (int j) {
  int i, L = Inds[j].GetChromosomeLength();
  double rnd;

  for (i=0; i&ltL; i++) {
    rnd = (double)rand()/(double)RAND_MAX + 1);
    if (mutationRate > rnd) {  // mutationRate - вероятность мутации
      Inds [j] = Inds [j] ^ MutationMask [i];
    }
  }
}
```

Находят применение операторы кроссинговера для вещественного кодирования

Предположим, что $C1=\{c1_1,..,c1_n\}$ и $C2=\{c2_1,..,c2_n\}$ - две хромосомы к которым применяется оператор кроссинговера. Будем обозначать потомков как $H1$ и $H2$. Ниже кратко описаны некоторые операторы скрещивания для генетических алгоритмов с вещественным кодированием.

- **Двухточечный кроссинговер (Two-point crossover).** Случайным образом выбираются две точки разрыва i и j из диапазона [1, n-1]. Пусть при этом $i<j$. Тогда потомки определяются путем обмена соответствующих частей родительских хромосом: $H1 = \{c1_1, c1_2,.., c2_i, c2_{i+1},.., c2_j, c1_{j+1},.., c1_n\}$, $H2 = \{c2_1, c2_2,.., c1_i, c1_{i+1},.., c1_j, c2_{j+1},.., c2_n\}$.
- **Однородный кроссинговер (Uniform crossover).** Работает как однородный кроссинговер для бинарного

кодирования с той лишь разницей, что участок хромосомы, для которого разыгрывается вероятность попасть к первому или ко второму потомку не отдельный разряд, а значение параметра целиком. Т.е. разыгрывается не бит, а весь ген. Иными словами, для каждого гена, если случайная бюинарная переменная *Rnd*=0, то ген от 1-го родителя попадает к первому потомку, а ген второго родителя - ко второму. Если же *Rnd*=1, то наоборот.

Используются также кроссинговеры:

- **Арифметический кроссинговер (Arithmetical crossover).**
- **Геометрический кроссинговер (Geometrical crossover,).**
- **BLX-a кроссинговер (BLX-a crossover).**
- **Имитированный бинарный кроссинговер (Simulated binary crossover).**
- **Нечеткая рекомбинация (Fuzzy recombination).**

Стратегии формирования нового поколения

После скрещивания особей необходимо решить проблему о том какие из новых особей войдут в следующее поколение, а какие - нет, и что делать с их предками. Есть два способа:

1. Новые особи (потомки) занимают места своих родителей. После чего наступает следующий этап, в котором потомки оцениваются, отбираются, дают потомство и уступают место своим "детям".
2. Создается промежуточная популяция, которая включает в себя как родителей, так и их потомков. Члены этой популяции оцениваются, а затем из них выбираются N самых лучших, которые и войдут в следующее поколение.

Те кто знаком с эволюционными стратегиями, то с этими способами вы уже встречались. Какой из этих двух вариантов лучше – ответить однозначно затруднительно. Видимо второй вариант практичнее (т.к. не позволяет заменять приспособленных родителей на "неизвестно кого"), но тут может быть больше проблем с преждевременной сходимостью, чем в первом варианте. К тому же он требует сортировки массива

размером (как минимум) 2*N.

Вообще говоря, можно комбинировать стратегии отбора и формирования следующего поколения как угодно - ограничений нет никаких.

Принцип "элитизма"

Суть этого принципа заключается в том, что в новое поколение включаются лучшие родительские особи. Их число может быть от 1 и больше. Использование "элитизма" позволяет не потерять хорошее промежуточное решение, но, в то же время, из-за этого алгоритм может "застрять" в локальном экстремуме. Однако, опыт использования принципа "элитизма" , позволяет сделать вывод, что в большинстве случаев "элитизм" нисколько не вредит поиску решения, и главное - это предоставить алгоритму возможность анализировать *разные* строки из пространства поиска.

Элитизм

В классическом описании генетического алгоритма подразумевается создание популяции потомков и замещение родительских особей их "детьми". Такой подход достаточно естественнен, но не особенно эффективен, т.к. потомки, полученные в результате генетических преобразований, могут быть хуже родительских особей. В результате появилось несколько подходов, "исправляющих" данное явление, которые можно объединить, используя понятие "элитизм" (elitism) (иногда "стратегия элитизма" (elitist strategy). Ниже будет приведено краткое описание этих подходов. Отметим, что краткое описание элитизма уже было сделано раньше (см. Стратегии отбора и формирования нового поколения), однако считают необходимым уделить ему отдельное внимание. т.

Условно элитизм можно разделить на два класса-подхода, назовем их конкурентным и неконкурентным. Коротко их суть в следующем:

1. *Конкурентный подход*. Родительские особи "состязаются" с потомками и победители (или победитель) переходят в

следующее поколение.

2. *Неконкурентный подход*. В этом случае часть родительской подпопуляции (случайная либо определенная по заданному правилу) переходит в новое поколение без каких-либо возражений со стороны электората.

Иными словами, в первом классе потомки после создания имеют, в общем, равные права с родителями на то, чтобы перейти в новое поколение и определяющую роль здесь играет приспособленность особи, а не ее положение на генеалогическом древе. Во втором классе престарелые индивиды имеют определенный приоритет и даже если все потомки будут лучше любого родителя, какая-то часть родительской подпопуляции неизбежно будет присутствовать в новом поколении.

Рассмотрим каждый класс более подробно. При этом будем по возможности принимать во внимание наработки в области эволюционных стратегий (evolutionary strategies). Свою историю эволюционные стратегии ведут с, как минимум, середины 60-х годов и принятые в них обозначения весьма удобны для указания различных подходов к элитизму.

1. Конкурентный подход

Выделим в классе конкурентных подходов 2 подкласса (в скобках приведены жаргонные названия, рекомендуемые для употребления в псевдо-, квази- и мета-научной литературе):

- глобальное состязание (массовое побоище, жестокое и беспощадное)
- локальное состязание (бои удельных князьков на кулачках)

При глобальном состязании сначала создаются все потомки (ключевое слово "все"), которые затем соревнуются на общих основаниях со всеми родителями (ключевое слово "всеми") и те, кто окажется лучше, независимо от возраста, переходят в новое поколение. Т.е. после создания потомков определяется их приспособленность и, зная приспособленность в текущей популяции (т.е. в популяции, из которой выбирались родительские особи), можно, отсортировав особей из

текущей популяции и потомков, сформировать популяцию для следующего поколения. В данном случае оптимист скажет, что выбрали лучшую часть, а пессимист скажет, что отбросили худшую часть, и оба будут правы, точно также как и в случае со стаканом с водой. Потомки не обязательно соревнуются со всеми особями из текущей популяции, можно устроить чемпионат взяв только родительских особей и их потомков. Главное в глобальных состязаниях, чтобы особи соревновались вместе.

Несколько иной подход используется в локальных состязаниях. Рассмотрим достаточно распространенный случай, когда две родительских особи используются для создания двух потомков. Тогда, непосредственно после создания, производится оценка потомков, а затем потомки соревнуются только со своими родителями. Здесь проблемы семьи решаются внутри семьи. Таким образом популяция нового поколения формируется из победителей многочисленных локальных состязаний.

С точки зрения эволюционных стратегий конкурентный подход соответствует (*nu* + lambda) стратегии (которая называется "плюс-стратегия" (plus strategy)), где *nu* -- количество родительских особей, а lambda -- количество созданных потомков.

2. Неконкурентный подход

В неконкурентном подходе все значительно проще. После оценки текущей популяции выбираются несколько наилучших особей (т.е. особей имеющих максимальную приспособленность) и они "автоматически" попадают в следующее поколение. При этом элитные особи могут принимать участие в скрещивании наравне с остальными особями. В силу того, что, таким образом, родительские особи не соревнуются в той или иной форме со потомками, такой подход и получил название неконкурентный.

В эволюционных стратегиях неконкурентный подход соответствует (mu, lambda) стратегии (дословный перевод выглядит как "запятая-стратегия" (comma strategy)), смысл переменных mu и lambda остается

прежним.

Здесь мы проанализируем последствия использования, либо не использования того или иного подхода к элитизму. Что же влечет использование элитизма? Очевидно, что при элитизме медленнее обновляется генетическая информация в популяции, т.к. часть особей, точнее их геномы, остаются неизменными при смене поколений. И хотя такое возможно и без элитизма (например, скрещивание двух "похожих" родительских особей может привести к созданию идентичных с ними потомков) в случае с элитизмом вероятность этого события существенно увеличивается, прежде всего, в силу специфики самого подхода к элитизму. Хорошо это или плохо? Попытаемся проанализировать.

Для этого необходимо понять, когда (не)выгодно интенсивное обновление геномов популяции. Очевидно, что если популяция "приближается" к глобальному оптимуму (т.е. находится на сравнительно небольшом от него удалении), то резкие изменения особей (большинство из которых обладают высокой приспособленностью) нежелательны, т.к. могут ухудшить соответствующие этим особям решения. Поэтому, в данном случае, элитизм, как способ сохранения "хороших" особей, необходим. Нужно отметить, что описанная ситуация, как правило, не наблюдается в начале эволюции, за исключением случая, когда решается очень простая задача, но для них ГА, по большому счету, не нужны. Если же эволюционный поиск находится в самом начале, то, очевидно, необходимо активно исследовать пространство поиска, чтобы выделить в нем потенциальные области, при попадании в которые, популяция будет иметь высокую приспособленность. Однако, если в результате генетических преобразований получается особь, которая находится в одной из таких областей (и допустим обладает максимальной приспособленностью по сравнению с остальными особями в популяции) то необходимо, чтобы характерные для этой особи генотипические (и, соответственно, фенотипические) признаки закрепились в популяции, иначе данная область на неопределенное время будет "потеряна".

Небольшое отступление. Генотип особи -- это, как правило, ее генетическая информация, а фенотип особи -- соответствующее этой

особи решение поставленной задачи, т.е., в простейшем случае, фенотип -- декодированный из генетического представления набор оптимизируемых параметров. Другими словами, генотип -- это ДНК. Изменение генотипа, получаемое в результате скрещивания и мутации, влечет изменение соответствующего фенотипа и для случая прямого кодирования здесь все очевидно, т.к. генотип однозначно соответствует фенотипу (и обратно, для каждого фенотипа можно найти единственный генотип). Для кодирования Грея отображение генотип-фенотип также однозначно, хотя и не так наглядно. Однако при неоднозначном соответствии генотипа и фенотипа могут возникать проблемы в адекватной оценке особи.

Дело в том, что и приспособленная особь может исчезнуть, например, потому что генератор случайных чисел выдал "неудачную" для этой особи последовательность, в результате чего она либо вообще не приняла участия в скрещивании (в частности, могут "задавить числом" неприспособленные особи), либо ее потомки на нее не "похожи", либо же они сильно мутировали. При использовании элитизма шансы, что генетические свойства рассматриваемой (потенциально многострадальной) особи сохранятся и распространятся в популяции, выглядят обнадеживающе. Получается, что и на начальных этапах элитизм нужен.

Более сложная картина для неоднозначного отображения генотип-фенотип. Например, когда в генотипе закодирована структура искусственной нейронной сети (ИНС). Тогда одной и той же особи (структуре ИНС) могут соответствовать различные по характеристикам ИНС, в следствие различных значений весов связей. В этом случае, даже если приспособленности некоторой особи высока, это не дает достаточных оснований для преимущества перед менее приспособленными объктами, т.к. все рассуждения о приспособленности ведутся в условиях неполной информации. В некоторой степени ситуацию можно выправить, если для каждой особи исследовать несколько различных соответствующих ей фенотипов (в данном примере ИНС с различными весами, но одинаковой структурой). Тем не менее, неопределенность, хоть и меньшая, остается. В силу этих рассуждений необходимость в элитизме в данном случае остается под вопросом. Вполне возможно, что лучше вообще запретить конкуренцию между особями, имеющими существенно

различные по составу генотипы (которым соответствуют сильно отличающиеся структуры ИНС). Сделать это можно, например, с помощью "нишинга (niching)".

В любом случае, не стоит забывать о "формуле успеха" в генетических алгоритмах, которую можно сформулировать следующим образом: "Необходимо соблюдать баланс между исследованием пространства поиска (exploration) и использованием найденных хороших решений (exploitation)". Т.е. если используется элитизм, другими словами, более активно используются уже найденные хорошие особи, то необходимы меры, которые будут компенсировать влияние элитизма, например, увеличение вероятности мутации, либо увеличение размера популяции, либо запрет на существование "особей-близнецов" в популяции. В противном случае существенно увеличивается риск преждевременной сходимости.

Использование элитизма, в целом, позволяет существенно улучшить результаты работы ГА. Тем не менее, использовать элитизм необходимо с определенной осторожностью, чтобы избежать преждевременной сходимости. В результате классификации известных автору методов элитизма предлагается выделять следующие подходы:

1. Конкурентный подход с локальным состязанием.

2. Конкурентный подход с глобальным состязанием.

3. Неконкурентный подход.

2.3. Принцип и алгоритм работы ГА

Принцип работы ГА

Генетические алгоритмы оперируют совокупностью особей (популяцией), которые представляют собой строки, кодирующие одно из решений задачи. Этим ГА отличается от большинства других алгоритмов оптимизации, которые оперируют лишь с одним решением, улучшая его.

С помощью функции приспособленности среди всех особей популяции выделяют:

- наиболее приспособленные (более подходящие решения), которые получают возможность скрещиваться и давать потомство
- наихудшие (плохие решения), которые удаляются из популяции и не дают потомства

Таким образом, приспособленность нового поколения в среднем выше предыдущего.

В классическом ГА:

- начальная популяция формируется случайным образом
- размер популяции (количество особей N) фиксируется и не изменяется в течение работы всего алгоритма
- каждая особь генерируется как случайная L-битная строка, где L — длина кодировки особи
- длина кодировки для всех особей одинакова

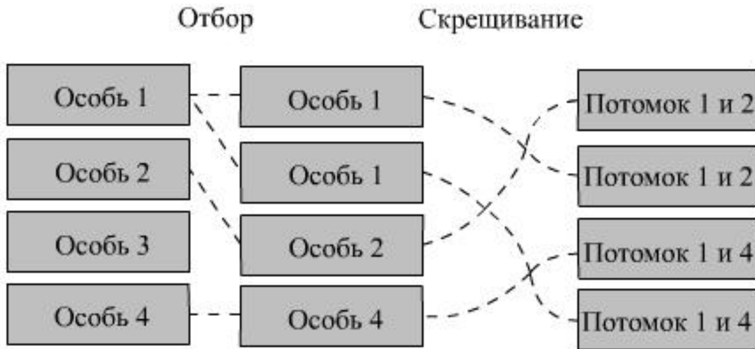
Алгоритм работы

На рисунке изображена схема работы любого генетического алгоритма:

Шаг алгоритма состоит из трех стадий:

1. генерация промежуточной популяции (*intermediate generation*) путем отбора (*selection*) текущего поколения
2. скрещивание (*recombination*) особей промежуточной популяции путем *кроссовера* (*crossover*), что приводит к формированию нового поколения
3. мутация нового поколения

Первые две стадии (отбор и скрещивание):



Отбор

Промежуточная популяция — это набор особей, получивших право размножаться. Наиболее приспособленные особи могут быть записаны туда несколько раз, наименее приспособленные с большой вероятностью туда вообще не попадут.

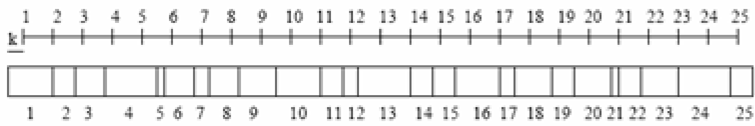
В классическом ГА вероятность каждой особи попасть в промежуточную популяцию пропорциональна ее приспособленности, т.е. работает *пропорциональный отбор* (*proportional selection*).

Существует несколько способов реализации данного отбора:

- *stochastic sampling*. Пусть особи располагаются на колесе рулетки так, что размер сектора каждой особи пропорционален ее приспособленности. N раз запуская рулетку, выбираем

требуемое количество особей для записи в промежуточную популяцию.

- *remainder stochastic sampling*. Для каждой особи вычисляется отношение ее приспособленности к средней приспособленности популяции. Целая часть этого отношения указывает, сколько раз нужно записать особь в промежуточную популяцию, а дробная показывает её вероятность попасть туда ещё раз. Реализовать такой способ отбора удобно следующим образом: расположим особи на рулетке так же, как было описано. Теперь пусть у рулетки не одна стрелка, а *N*, причем они отсекают одинаковые сектора. Тогда один запуск рулетки выберет сразу все *N* особей, которые нужно записать в промежуточную популяцию. Такой способ иллюстрируется следующим рисунком:



Скрещивание

Особи промежуточной популяции случайным образом разбиваются на пары, потом с некоторой вероятностью скрещиваются, в результате чего получаются два потомка, которые записываются в новое поколение, или не скрещиваются, тогда в новое поколение записывается сама пара.

В классическом ГА применяется одноточечный оператор кроссовера (*1-point crossover*): для родительских строк случайным образом выбирается точка раздела, потомки получаются путём обмена отсечёнными частями.

```
011010.01010001101   ->   111100.01010001101
111100.10011101001        011010.10011101001
```

Мутация

К полученному в результате отбора и скрещивания новому поколению применяется оператор мутации, необходимый для "выбивания" популяции из локального экстремума и способствующий защите от преждевременной сходимости.

Каждый бит каждой особи популяции с некоторой вероятностью инвертируется. Эта вероятность обычно очень мала, менее 1%.

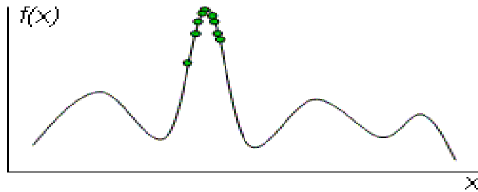
`101100110`**`0`**`101101 -> 101100110`**`1`**`101101`

Можно выбирать некоторое количество точек в хромосоме для инверсии, причем их число также может быть случайным. Также можно инвертировать сразу некоторую группу подряд идущих точек. Среди рекомендаций по выбору вероятности мутации нередко можно встретить варианты 1/L или 1/N.

Критерии останова

Такой процесс эволюции, вообще говоря, может продолжаться до бесконечности. Критерием останова может служить заданное количество поколений или *схождение* (*convergence*) популяции.

Схождением называется состояние популяции, когда все строки популяции находятся в области некоторого экстремума и почти одинаковы. То есть кроссовер практически никак не изменяет популяции, а мутирующие особи склонны вымирать, так как менее приспособлены. Таким образом, схождение популяции означает, что достигнуто решение близкое к оптимальному.



Итоговым решением задачи может служить наиболее приспособленная особь последнего поколения.

2.4. Применение генетических алгоритмов

Генетические алгоритмы применяются для решения следующих задач:

1. Оптимизация функций
2. Оптимизация запросов в базах данных
3. Разнообразные задачи на графах (задача коммивояжера, раскраска, нахождение паросочетаний)
4. Настройка и обучение искусственной нейронной сети
5. Задачи компоновки
6. Составление расписаний
7. Игровые стратегии
8. Теория приближений
9. Искусственная жизнь
10. Биоинформатика (фолдинг белков)
11. Синтез конечных автоматов
12. Настройка ПИД регуляторов
13. Синтез конструкций антенн

Пример простой реализации на C++

Поиск в одномерном пространстве, без скрещивания.

```
#include <cstdlib>
#include <ctime>
```

```
#include <algorithm>
#include <iostream>
#include <numeric>

int main()
{
    srand((unsigned int)time(NULL));
    const size_t N = 1000;
    int a[N] = { 0 };
    for ( ; ; )
    {
        //мутация в случайную сторону каждого
элемента:
        for (size_t i = 0; i < N; ++i)
            a[i] += ((rand() % 2 == 1) ? 1 : -1);

        //теперь выбираем лучших, отсортировав по
возрастанию
        std::sort(a, a + N);
        //и тогда лучшие окажутся во второй
половине массива.
        //скопируем лучших в первую половину, куда
они оставили потомство, а первые умерли:
        std::copy(a + N / 2, a + N, a);
        //теперь посмотрим на среднее состояние
популяции. Как видим, оно всё лучше и лучше.
        std::cout << std::accumulate(a, a + N, 0) /
N << std::endl;
    }
}
```

Пример простой реализации на Delphi

Поиск в одномерном пространстве с вероятностью выживания, без скрещивания. *(проверено на Delphi XE)*

```
program Program1;

{$APPTYPE CONSOLE}
```

```
{$R *.res}

uses
  System.Generics.Defaults,
  System.Generics.Collections,
  System.SysUtils;

const
  N = 1000;
  Nh = N div 2;
  MaxPopulation = High(Integer);
var
  A: array [1..N] of Integer;
  I, R, C, Points, BirthRate: Integer;
  Iptr: ^Integer;
begin
  Randomize;
  // Частичная популяция
  for I := 1 to N do
    A[I] := Random(2);
  repeat
    // Мутация
    for I := 1 to N do
      A[I] := A[I] + (-Random(2) or 1);
    // Отбор, лучшие в конце
    TArray.Sort<Integer>(A,
TComparer<Integer>.Default);
    // Предустановка
    Iptr := Addr(A[Nh + 1]);
    Points := 0;
    BirthRate := 0;
    // Результаты скрещивания
    for I := 1 to Nh do
    begin
      Inc(Points, Iptr^);
      // Случайный успех скрещивания
      R := Random(2);
      Inc(BirthRate, R);
      A[I] := Iptr^ * R;
      Iptr^ := 0;
```

```
      Inc(Iptr,1);
    end;
    // Промежуточный итог
    Inc(C);
  until (Points / N >= 1) or (C >= MaxPopulation);
  Writeln(Format('Population %d (rate:%f)
score:%f', [C, BirthRate / Nh, Points / N]));
end.
```

2.5. Пример выполнения классического генетического алгоритма

Рассмотрим выполнение описанного ранее классического генетического алгоритма на как можно более простом примере . Проследим последовательность выполнения его этапов, соответствующих блок-схеме на рис. 2.1.

Пример 1

Рассмотрим сильно упрощенный и довольно искусственный пример, состоящий в нахождении хромосомы с максимальным количеством единиц. Допустим, что хромосомы состоят из 12 генов, а популяция насчитывает 8 хромосом. Понятно, что наилучшей будет хромосома, состоящая из 12 единиц. Посмотрим, как протекает процесс решения этой весьма тривиальной задачи с помощью генетического алгоритма.

Инициализация, или выбор исходной популяции хромосом. Необходимо случайным образом сгенерировать 8 двоичных последовательностей длиной 12 битов. Это можно достигнуть, например, подбрасыванием монеты (96 раз, при выпадении «орла» приписывается значение 1, а в случае «решки» - 0). Таким образом можно сформировать исходную популяцию

ch_1, = [111001100101]
ch_2 = [001100111010]
ch_3 = [011101110011]
ch_4 = [001000101000]
ch_5 = [010001100100]
ch_6 = [010011000101]
ch_7 = [101011011011]
ch_8 = [000010111100]

Оценка приспособленности хромосом в популяции. В рассматриваемом упрощенном примере решается задача нахождения

такой хромосомы, которая содержит наибольшее количество единиц. Поэтому функция приспособленности определяет количество единиц в хромосоме. Обозначим функцию приспособленности символом *F*. Тогда ее значения для каждой хромосомы из исходной популяции будут такие:

$F(ch_1) = 7$
$F(ch_2) = 6$
$F(ch_3) = 8$
$F(ch_4) = 3$
$F(ch_5) = 4$
$F(ch_6) = 5$
$F(ch_7) = 8$
$F(ch_8) = 5$

Хромосомы ch_3 и ch_7 характеризуются наибольшими значениями функции принадлежности. В этой популяции они считаются наилучшими кандидатами на решение задачи. Если в соответствии с блок-схемой генетического алгоритма условие остановки алгоритма не выполняется, то на следующем шаге производится селекция хромосом из текущей популяции.

Селекция хромосом. Селекция производится методом рулетки. На основании формул (2.1) и (2.2) для каждой из 8 хромосом текущей популяции (в нашем случае - исходной популяции, для которой N = 8) получаем секторы колеса рулетки, выраженные в процентах (рис. 2.4)

$v(ch_1)= 15,22$
$v(ch_2) = 13,04$
$v(ch_3)= 17,39$
$v(ch_4)= 6,52$
$v(ch_5)= 8,70$
$v(ch_6) = 10,87$
$v(ch_7)= 17,39$
$v(ch_8)= 10,87$

Розыгрыш с помощью колеса рулетки сводится к случайному выбору числа из интервала [0, 100], указывающего на соответствующий сектор на колесе, т.е. на конкретную хромосому. Допустим, что разыграны следующие 8 чисел:

79 44 9 74 44 86 48 23

Это означает выбор хромосом

ch_7 ch_3 ch_1 ch_7 ch_3 ch_7 ch_4 ch_2

Как видно, хромосома ch_7 была выбрана трижды, а хромосома ch_3 - дважды. Заметим, что именно эти хромосомы имеют наибольшее

значение функции приспособленности. Однако выбрана и хромосома ch_4 с наименьшим значением функции приспособленности. Все выбранные таким образом хромосомы включаются в так называемый родительский пул.

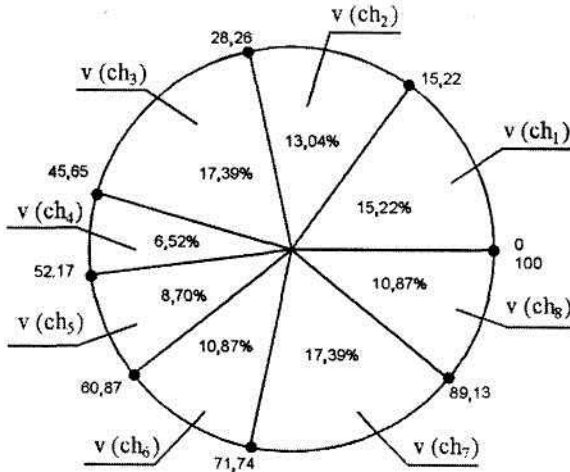


Рис.2.4. Колесо рулетки для селекции в примере 2.1.

Применение генетических операторов. Допустим, что ни одна из отобранных в процессе селекции хромосом не подвергается мутации, и все они составляют популяцию хромосом, предназначенных для скрещивания. Это означает, что вероятность скрещивания $p_c = 1$, а вероятность мутации $p_m = 0$. Допустим, что из этих хромосом случайным образом сформированы пары родителей

ch_2 и ch_7 ch_1 и ch_7 ch_3 и ch_4 ch_3 и ch_7

Для первой пары случайным образом выбрана точка скрещивания $k = 4$, для второй $k = 3$, для третьей $k = 11$, для четвертой $Ik = 5$. При этом процесс скрещивания протекает так, как показано на рис.2.5.

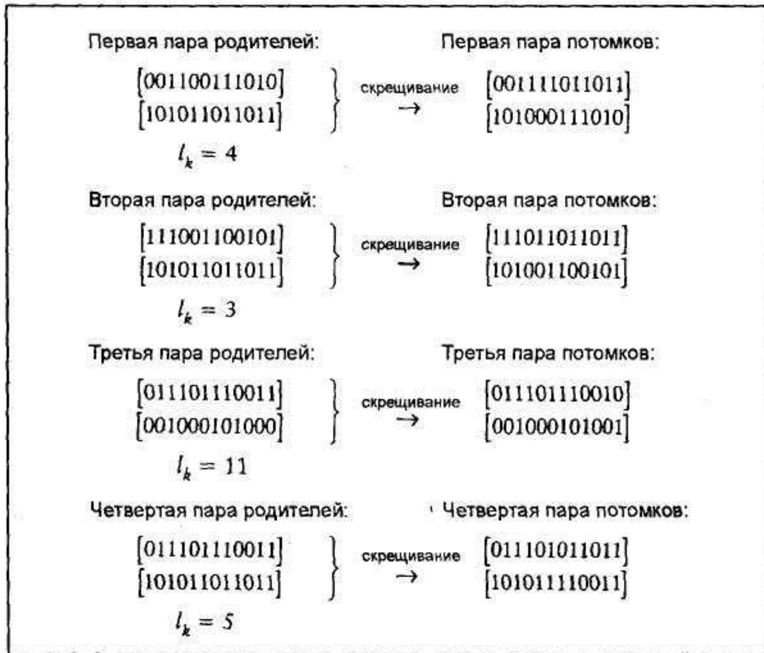


Рис.2.5. Процесс скрещивания хромосом в примере 2.1.

В результате выполнения оператора скрещивания получаются 4 пары потомков.
Если бы при случайном подборе пар хромосом для скрещивания были объединены, например, ch_3 с ch_3 и ch_4 с ch_7 вместо ch_3 с ch_4 и ch_3 с ch_7, а другие пары остались без изменения, то скрещивание ch_3 с ch_3 дало бы две такие же хромосомы независимо от разыгранной точки скрещивания. Это означало бы получение двух потомков, идентичных своим родителям. Заметим, что такая ситуация наиболее вероятна для хромосом с наибольшим значением функции приспособленности, т.е. именно такие хромосомы получают наибольшие шансы на переход в новую популяцию.
Формирование новой популяции. После выполнения операции скрещивания мы получаем (согласно рис.2.5) следующую популяцию потомков:

Ch_1 = [001111011011]
Ch_2 = [101000111010]
Ch_3 = [111011011011]
Ch_4 = [101001100101]
Ch_5 = [011101110010]
Ch_6 = [001000101001]
Ch_7 = [011101011011]
Ch_8 = [101011110011]

Для отличия от хромосом предыдущей популяции обозначения вновь сформированных хромосом начинаются с заглавной буквы С. Согласно блок-схеме генетического алгоритма (рис.2.1) производится возврат ко второму этапу, т.е. к оценке приспособленности хромосом из вновь сформированной популяции, которая становится текущей. Значения функций приспособленности хромосом этой популяции составляют
$F(Ch_1) = 8$
$F(Ch_2) = 6$
$F(Ch_3) = 9$
$F(Ch_4) = 6$
$F(Ch_5) = 7$
$F(Ch_6) = 4$
$F(Ch_7) = 8$
$F(Ch_8) = 8$

Заметно, что популяция потомков характеризуется гораздо более высоким средним значением функции приспособленности, чем популяция родителей. Обратим внимание, что в результате скрещивания получена хромосома Ch_3 с наибольшим значением функции приспособленности, которым не обладала ни одна хромосома из родительской популяции. Однако могло произойти и обратное, поскольку после скрещивания на первой итерации хромосома, которая в родительской популяции характеризовалась наибольшим значением функции приспособленности, могла просто «потеряться». Помимо этого «средняя» приспособленность новой популяции все равно оказалась бы выше предыдущей, а хромосомы с большими значениями функции приспособленности имели бы шансы появиться в следующих поколениях.

2.6. Представление данных в генах

Для решения некоторой задачи с помощью ГА её данные необходимо представить в виде генов особи. Для этого прежде всего необходимо определить какие параметры задачи необходимо настроить. Например, если мы пытаемся аппроксимировать с помощью ГА набор точек функцией вида f(x)=A*exp(k*x), где A, k - константы, x - независимая переменная, то в качестве параметров будут выступать A и k, т.к. от их значения зависит вид и поведение функции. Итак, имеем два параметра и, следовательно, два гена.

Следующий шаг - это выбор числа разрядов в каждом гене. Здесь необходимо учитывать, что с одной стороны, чем больше разрядов, тем лучше, т.к. выше точность и т.д., но с другой - большое число разрядов влечет увеличение времени поиска решения (сходимости). Стоит отметить, что с ростом числа параметров (в некоторых задачах они исчисляются сотнями) "лишние" разряды сказываются все сильнее и сильнее. Поэтому необходимо найти компромисс между точностью и скоростью. Возьмем для задачи аппроксимации 16-разрядные гены, при этом параметры будут изменяться от -32,768 до 32,767 с шагом в 0,001. Данные числа получены, исходя из того, что 16-разрядное число может принимать одно из 2^{16}= 65536 значений от 0 до 65535. Если предположить, что 0 будет в середине этого интервала, причем каждое значение разделится затем на 1000, то получим интервал изменения параметров и шаг изменения.

После того, как выбраны параметры, их число и разрядность, необходимо решить, как непосредственно записывать данные. Можно использовать обычное кодирование, когда $1011=1011_2=11_{10}$, либо коды Грея, когда $1011=1110_2=14_{10}$. Несмотря на то, что коды Грея влекут неизбежное кодирование/декодирование данных, они позволяют избежать некоторых проблем, которые появляются в результате обычного кодирования. Можно лишь сказать, что преимущество кода Грея в том, что если два числа различаются на 1, то и их двоичные коды различаются только на один разряд, а в двоичных кодах не все так просто. Стоит отметить, что кодировать и декодировать в коды Грея довольно удобно, описать это можно так: сначала копируется самый старший разряд, затем:

- Из двоичного кода в код Грея: G[i] = XOR(B[i+1], B[i])
- Из кода Грея в двоичный код: B[i] = XOR(B[i+1], G[i])

Здесь, G[i] - i-й разряд кода Грея, а B[i] - i-й разряд бинарного кода. Например, последовательность чисел от 0 до 7 в двоичном коде: {000, 001, 010, 011, 100, 101, 110, 111}, а в кодах Грея: {000, 001, 011, 010, 110, 111, 101, 100}.

Итак, теперь заданы все необходимые характеристики генов, остается только представить все это на каком-нибудь языке программирования. Например, на C++ одна особь с хромосомой из двух 16-разрядных генов может "выглядеть" так:

```
1.  short int Individual[2];

2.    class Individual {
3.      short int Genes[2];
4.    };
5.  unsigned char Individual [32];
```

Последний пример на первый взгляд кажется как минимум странным: зачем для задания одного бита использовать целый байт? Однако бывают случаи, когда используются небинарные алфавиты, т.е. каждый разряд может принимать не 2 различных значения "0" или "1", а больше, например, "1", "2", "4", "8". Как раз для таких ситуаций последний пример и удобен. В дальнейшем речь будет идти только о бинарном алфавите. Кроме того, в последнем способе легче использовать коды Грея. Для того чтобы задать популяцию из 50 особей, можно поступить одним из следующих способов:

```
1.  short int Population[50][2];

2.    class Population {
3.      Individual Inds[50];
4.    };
5.  unsigned char Population[50][32];
```

Следует также отметить, что в задаче аппроксимации, взятой в качестве

примера, в процессе "эволюции" будут учавствовать все гены особи, т.е. генетические операторы будут применяться к каждому гену. Однако, это не всегда необходимо, например, рассмотрим задачу компоновки, т.е. когда на некоторой известной поверхности требуется разместить N элементов разной площади так, чтобы они не накладывались друг на друга. Для простоты будем считать, что все элементы имеют прямоугольную форму. Настраиваемыми (оптимизируемыми) параметрами будут координаты каждого элемента, т.е. каждая хромосома будет содержать два гена, соответствующие координатам левого верхнего угла элемента. Но помимо этого необходимо знать, какой элемент представляет данная особь. Для этого необходимо добавить в набор генов особи ещё один ген, содержащий номер элемента. При этом нам нужно сохранить значение этого гена неизменным в течении всего процесса поиска решения. Таким образом, получаем особь с тремя генами, два из которых обрабатываются генетическими операторами, а один нет.

Помимо рассмотренных задач аппроксимации и компоновки можно привести варианты кодирования для некоторых других задач:

- Оптимизация функций: гены - независимые переменные;
- Настройка весов искусственной нейронной сети: гены - синаптические веса нейронов;
- Искусственная жизнь (Artificial Life): гены - характеристики особи (сила, скорость, и т.д.), также должны быть неизменные гены, обозначающие тип особи (растение или животное);
- Задача о кратчайшем пути: гены - пункты передвижения. Вся хромосома целиком представляет из себя маршрут из начальной точки в конечную, причем не всегда существующий.

И последнее, разрядность генов необязательно должна быть фиксированной, многие современные алгоритмы самостоятельно подстраивают число разрядов для каждого гена в зависимости от промежуточных результатов поиска решения.

2.7. Примеры кодирования параметров задачи в генетическом алгоритме

Выбор исходной популяции связан с представлением параметров задачи в форме хромосом, т.е. с так называемым хромосомным представлением. Это представление определяется способом кодирования. В классическом генетическом алгоритме применяется двоичное кодирование, т.е. аллели всех генов в хромосоме равны 0 или 1. Дпина хромосом зависит от условий задачи, точнее говоря - от количества точек в пространстве поиска.

Генетические алгоритмы находят применение главным образом в задачах оптимизации. Пример 2.2 демонстрирует выполнение классического генетического алгоритма, аналогичного рассмотренному в примере 2.1, но для случая оптимизации функции. Для простоты примем, что это функция одной переменной. В новом примере хромосомы выступают в роли закодированной формы соответствующих фенотипов, а оптимизируется сама функция приспособленности. В примере 2.3 оптимизируется та же функция, однако внимание читателя акцентируется на другом способе кодирования хромосом для иной области определения переменной x.

Пример 2.2

Рассмотрим очень простой пример - задачу нахождения максимума функции, заданной выражением (2.1) для целочисленной переменной x, принимающей значения от 0 до 31.

Для применения генетического алгоритма необходимо прежде всего закодировать значения переменной x в виде двоичных последовательностей. Очевидно, что целые числа из интервала [0, 31] можно представить последовательностями нулей и единиц, используя их представление в двоичной системе счисления. Число 0 при этом записывается как 00000, а число 31 - как 11111. В данном случае хромосомы приобретают вид двоичных последовательностей, состоящих из 5 битов, т.е. цепочками длиной 5.

Также очевидно, что в роли функции приспособленности будет выступать целевая функция $f(x)$, заданная выражением (2.1). Тогда приспособленность хромосомы ch_i, $i = 1,2.....N$ будет определяться значением функции $f(x)$ для x, равного фенотипу, соответствующему генотипу ch_i. Обозначим эти фенотипы ch_i^*. В таком случае значение функции приспособленности хромосомы ch_i (т.е. $F(ch_i)$) будет равно $f(ch_i^*)$.

Выберем случайным образом исходную популяцию, состоящую из 6 кодовых последовательностей (например, можно 30 раз подбросить монету); при этом N=6. Допустим, что выбраны хромосомы

$ch_1 = [10011]$

ch_2 = [00011]
ch_3 = [00111]
ch_4 = [10101]
ch_5 = [01000]
ch_6 = [11101]
Соответствующие им фенотипы - это представленные ниже числа из интервала от 0 до 31:
$ch_1^* = 19$
$ch_2^* =3$
$ch_3^*= 7$
$ch_4^* = 21$
ch_5^*, = 8
$ch_6^* = 29$
По формуле (2.1) рассчитываем значения функции приспособленности для каждой хромосомы в популяции и получаем
$F(ch_1) = 723$
$F(ch_2)= 19$
$F(ch_3) = 99$
$F(ch_4) = 883$
$F(ch_5) = 129$
$F(ch_6)=1683$
Селекция хромосом. Методом рулетки (также, как и в примере2.1), выбираем 6 хромосом для репродукции. Колесо рулетки представлено на рис.2.6.

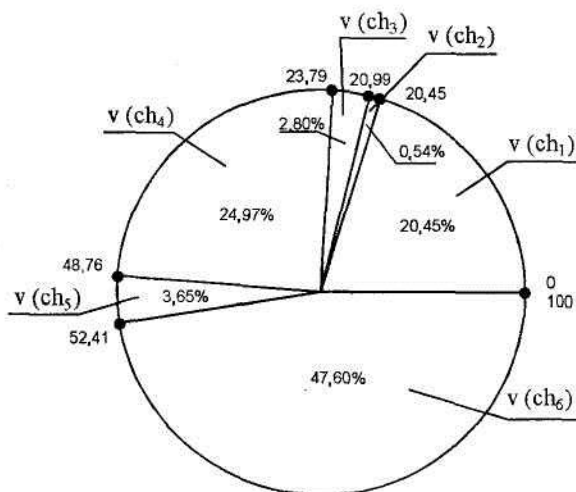


Рис.2.6. Колесо рулетки для селекции в примере 2.1.

Допустим, что выбраны числа
97 26 54 13 31 88.
Это означает выбор хромосом
ch6 ch_4 ch_6 ch_1 ch_4 ch_6
Пусть скрещивание выполняется с вероятностью $p_c = 1$. Допустим, что для скрещивания сформированы пары
ch_1 и ch_4 ch_4 и ch_6 ch_6 и ch_6
Кроме того, допустим, что случайным образом выбрана точка скрещивания, равная 3 для хромосом ch_1, и ch_4, а также точка скрещивания, равная 2 для хромосом ch_4 и ch_6 (рис.2.7).

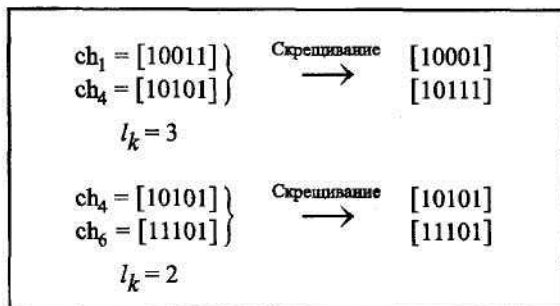


Рис.2.7. Процесс скрещивания хромосом в примере2.2.

При условии, что вероятность мутации $p_m = 0$, в новую популяцию включаются хромосомы

Ch_1 = [10001]
Ch_2 = [10111]
Ch_3 = [10101]
Ch_4 = [11101]
Ch_5 = [11101]
Ch_6 = [11101]

Для расчета значений функции приспособленности этих хромосом необходимо декодировать представляющие их двоичные последовательности и получить соответствующие им фенотипы. Обозначим их Ch_i*. В результате декодирования получаем числа (из интервала от 0 до 31)

Ch_1*= 17
Ch_2* = 23
Ch_3* = 21
Ch_4* = 29
Ch_5* = 29
Ch_6* = 29

Соответственно, значения функции приспособленности хромосом новой популяции, рассчитанные по формуле (2.1), составят

$F(Ch_1)$= 579
$F(Ch_2)$=1059
$F(Ch_3)$= 883
$F(Ch_4)$=1683
$F(Ch_5)$ = 1683
$F(Ch_6)$=1683

Легко заметить, что в этом случае среднее значение приспособленности возросло с 589 до 1262.

Обратим внимание, что если на следующей итерации будут сформированы для скрещивания пары хромосом, например, Ch_4 и Ch_2, Ch_5 и Ch_2 или Ch_6 и Ch_2 с точкой скрещивания 2 или 3, то среди прочих будет получена хромосома [11111] с фенотипом, равным числу 31, при котором оптимизируемая функция достигает своего максимума. Значение функции приспособленности для этой хромосомы оказывается наибольшим и составляет 1923. Если такое сочетание пар в данной итерации не произойдет, то можно будет ожидать образования хромосомы с наибольшим значением функции приспособленности на следующих итерациях. Хромосома [11111] могла быть получена и на текущей итерации в случае формирования для скрещивания пары Ch_1 и Ch_6 с точкой скрещивания 3.

Отметим, что при длине хромосом, равной 5 битам, пространство поиска очень мало и насчитывает всего $2^5 = 32$ точки. Представленный пример имеет исключительно демонстрационный характер. Применение генетического алгоритма для такого простого примера практически нецелесообразно, поскольку его оптимальное решение может быть получено мгновенно. Однако этот пример пригоден для изучения функционирования генетического алгоритма.

Также следует упомянуть, что в малых популяциях часто встречаются ситуации, когда на начальном этапе несколько особей имеют значительно большие значения функции принадлежности, чем остальные особи данной популяции. Применение метода селекции на основе «колеса рулетки» позволяет в этом случае очень быстро выбрать «наилучшие» особи, иногда - на протяжении «жизни» одного поколения. Однако такое развитие событий считается нежелательным, поскольку оно становится главной причиной преждевременной сходимости алгоритма, называемой сходимостью к неоптимальному решению. По этой причине используются и другие методы селекции, отличающиеся от колеса рулетки, либо применяется масштабирование функции приспособленности.

Пример 2.3

Рассмотрим задачу, аналогичную задаче из примера 2.2, т.е.
будем искать максимум функции, заданной формулой (2.1), но для переменной x, принимающей действительные значения из интервала $[a, b]$, где $a = 0$, $b = 3{,}1$. Допустим, что нас интересует решение с точностью до одного знака после запятой.

Поиск решения сводится к просмотру пространства, состоящего из 32 точек 0,0 0,1 ... 2,9 3,0 3,1. Эти точки (фенотипы) можно представить в виде хромосом (генотипов), если использовать бинарные пятизвенные цепочки, поскольку с помощью 5 битов можно получить $2^5=32$ различных кодовых комбинации. Следовательно, можно использовать такое же множество кодовых последовательностей, как и в примере 2.2, причем хромосома [00000] будет соответствовать числу 0,0, хромосома [00001] - числу 0,1 и т.д., вплоть до хромосомы [11111], соответствующей числу 3,1.

Таким образом, мы можем воспроизвести последовательность этапов генетического алгоритма (так же, как в примере 2.2), не забывая, что конкретным хромосомам (генотипам) в данном примере соответствуют другие фенотипы. Те кодовые последовательности, которые в примере 2.2 представляли фенотипы 0, 1, ..., 29, 30, 31, в рассматриваемой ситуации обозначают значения x, равные 0,0 0,1 ... 2,9 3,0 3,1. В связи с тем, что генетический алгоритм основан на случайном выборе исходной популяции и хромосом для последующего преобразования методом колеса рулетки, а также родительских пар для скрещивания и точки скрещивания, то генетический алгоритм в текущем примере будет выполняться аналогично, но не идентично предыдущему примеру.

В результате выполнения этого алгоритма будет выбрано наилучшее решение, которое представляется хромосомой [11111] со значением фенотипа 3,1. Функция приспособленности этой хромосомы равна 20,22; это максимально возможное значение.

Заметим, что если бы в примере 2.3 нас интересовало решение с точностью, превышающей один знак после запятой, то интервал [0, 3,1] необходимо было бы разбить на большее количество подинтервалов, и для кодирования соответственно большего количества чисел потребовались более длинные хромосомы (с длиной, превышающей 5 битов). Аналогично, расширение области определения переменной x также потребует применения более длинных хромосом. Из этих наблюдений можно сделать вывод, что длина хромосом зависит от ширины области определения x и от требуемой точности решения.

Представим теперь задачу из примера 2.3 в более общем виде. Допустим, что ищется максимум функции $f(x_1, x_2.....x_n) > 0$ для $x_i \in [a_i, b_i] \subset R$; $i = 1, 2, ..., n$ и требуется найти решение с точностью до q знаков после запятой для каждой переменной x_i. В такой ситуации необходимо разбить интервал $[a_i, b_i]$ на $(b_i - a_i) \cdot 10^q$ одинаковых подинтервалов. Это означает применение дискретизации с шагом $r = 10^{-q}$. Наименьшее натуральное число $m_i,$

удовлетворяющее неравенству

$(b_i\text{-}a_i)\bullet 10^q \leq 2^{m_i}$ -1 , (2.4)

определяет необходимую и достаточную длину двоичной последовательности, требуемой для кодирования числа из интервала [a_i, b_i] с шагом r. Каждой такой двоичной последовательности соответствует десятичное значение числа, представляемого данным кодом (с учетом правил перевода десятичных чисел в двоичную форму). Пусть y_i- обозначает десятичное значение двоичной последовательности, кодирующей число x_i. Значение x_i можно представить выражением

$$x_i = a_i + y_i \frac{b_i - a_i}{2^{m_i} - 1} \quad . \qquad (2.5)$$

Таким способом задаются фенотипы, соответствующие кодовым последовательностям с длиной m_i. Пример 2.3 - это частный случай задачи в данной постановке при условии, что i= 1 и q= 1. Выражение (2.5) - это следствие из простого линейного отображения интервала [a_i, b_i] на интервал [0, 2^{m_i} -1], где 2^{m_i} - десятичное число, закодированное двоичной последовательностью длиной m_i, и составленной исключительно из единиц, а 0 - это, очевидно, десятичное значение двоичной последовательности длиной m_i, составленной только из нулей. Обратим внимание, что если a_i = - 25, b_i = 25 и применяется шаг r=0,05, то согласно формуле (2.4) получаем m_i=10, а с помощью формулы 2.5) можно проверить значения фенотипов для генотипов, представленных в табл. 2.1.

3. Основы теории ГА

Рассмотрим несколько теоретических фактов, которые помогут более чётко понять природу генетических алгоритмов и примерно объяснить, почему же они работают.

3.1. Шаблоны

Шаблоном (*schema*) называется строка длины L (т. е. той же длины, что и любая строка популяции), состоящая из символов {0, 1, *} (где * — «don't care» символ). Будем говорить, что строка является представителем данного шаблона, если в позициях, где знак шаблона

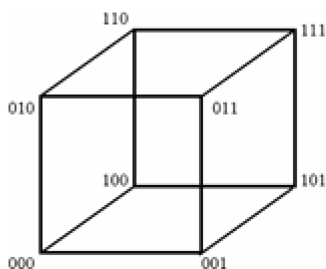
равен 0 или 1, она имеет тот же символ. Например, у шаблона 01*0*110 следующие представители:

- 01**0**0**0**110
- 01**0**0**1**110
- 01**1**0**0**110
- 01**1**0**1**110

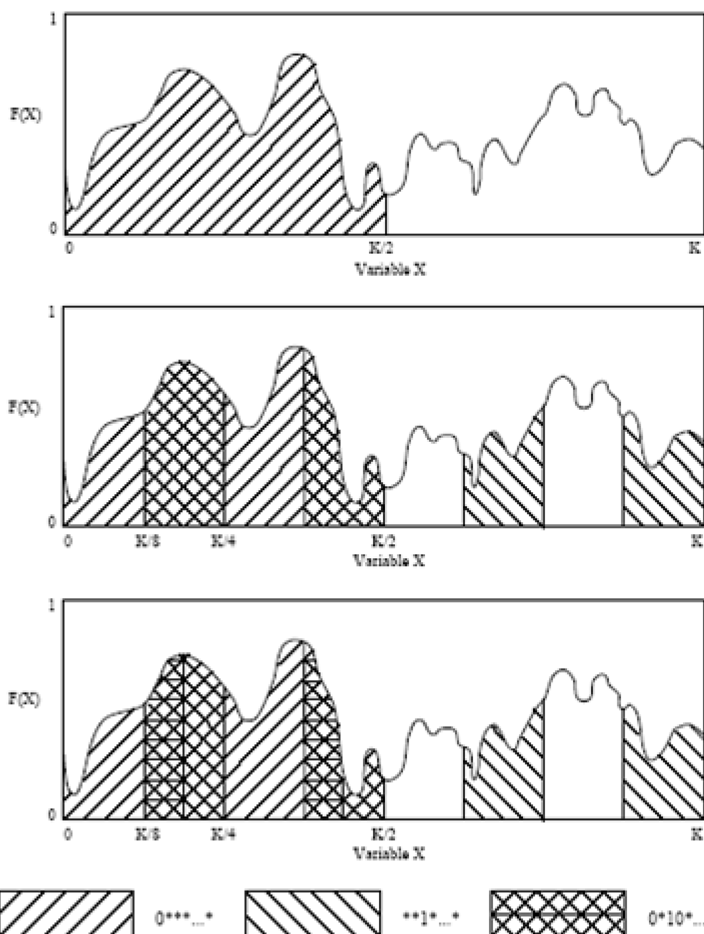
Порядком (*order*) шаблона называется количество фиксированных битов в нем. *Определяющей длиной* (*defining length*) шаблона называется расстояние между его крайними фиксированными битами. Например, для шаблона *1***01* порядок $o = 3$, а определяющая длина $\Delta = 5$.

Очевидно, что количество представителей шаблона H равно $2^{L-o(H)}$, а количество шаблонов равно 3^L (действительно, шаблоны — это строки, у которых на каждой позиции может находиться один из трех символов).

Если представить пространство поиска в виде гиперкуба, то строки это его вершины, а шаблон определяет в нем гиперплоскость. К примеру, шаблон **1 определяет правую грань этого трехмерного куба:



Поэтому термины «гиперплоскость» и «шаблон» взаимозаменяемы. Следующий рисунок изображает другое представление шаблонов:



На нем видно, что некоторые шаблоны имеют с среднем по всему пространству поиска большую приспособленность, чем другие.

Приспособленностью шаблона называется средняя приспособленность строк из популяции, являющихся его представителями. Следует заметить, что эта величина зависит от популяции, и поэтому меняется со временем.

Внешне кажется, что генетический алгоритм при отборе выбирает строку, однако при этом неявным образом происходит выборка шаблонов, представителем которых она является. Это означает, что на каждом поколении количество представителей шаблона изменяется в соответствии с текущей приспособленностью этого шаблона. У «хороших» шаблонов представители в среднем более приспособленные, а значит, они чаще будут выбираться в промежуточную популяцию. «Плохие» шаблоны имеют много шансов вымереть. Одна строка является представителем сразу многих шаблонов (а именно 2^L: на каждой позиции мы либо оставляем бит строки, либо заменяем его на «*»). Поэтому при отборе одной строки отбирается сразу целое множество шаблонов. Это явление получило название *неявный параллелизм* (*implicit parallelism*).

Теорема шаблонов

Теорема шаблонов (*The Schema Theorem*) была приведена в упомянутой выше работе Холланда и является первой попыткой объяснить, почему генетические алгоритмы работают. Она показывает, как изменяется доля представителей шаблона в популяции.

Пусть $M(H, t)$ — число представителей шаблона H в t-ом поколении. В силу того, что при построении промежуточной популяции используется пропорциональный отбор, в ней количество представителей данного шаблона будет

$$M(H, t + intermediate) = M(H, t)\, f(H, t) / \langle f(t) \rangle$$

где $f(H, t)$ — приспособленность шаблона H в t-ом поколении, а $\langle f(t) \rangle$ — средняя приспособленность t-го поколения.

Особи промежуточной популяции с вероятностью p_c подвергаются кроссоверу. Одноточечный кроссовер может разрушить шаблон, что означает, что один из родителей был представителем рассматриваемого шаблона, но ни один из детей уже таковым являться не будет. Вероятность разрушения меньше, чем

$$\Delta(H)\ (1 - P(H, t)\, f(H, t) / \langle f(t) \rangle) / (L-1)$$

где $P(H, t)$ — доля представителей шаблона H в t-ом поколении. Первый множитель произведения равен вероятности точки раздела попасть между фиксированными битами шаблона, а второй — вероятности выбрать в пару представителя другого шаблона.

Действительно, кроссовер разрушает шаблон не чаще, чем когда второй родитель (а он выбирается в промежуточной популяции) не является представителем этого шаблона, и при этом точка раздела попадает между фиксированными битами шаблона. Даже в этих ситуациях он не обязательно разрушается, например, если мы рассматриваем шаблон 11****, а кроссоверу подвергаются строки 110101 и 100000, и точка раздела попадает между первыми двумя битами, то, хотя вторая строка не является представителем нужного шаблона, все равно один из потомков окажется подходящим и шаблон не разрушится.

Таким образом, после кроссовера, переходя от количества представителей к их доле, получаем следующее неравенство:

$$P(H, t+1) \geq P(H, t)\, f(H, t)\, [1 - p_c\, \Delta(H)\, (1 - P(H, t)\, f(H, t) / \langle f(t) \rangle) / (L-1)] / \langle f(t) \rangle$$

Теперь учтем влияние мутации. Для каждого фиксированного бита вероятность того, что он не будет инвертирован, равна $(1 - p_m)$. Поскольку всего в шаблоне фиксированных битов $o(H)$, то верна следующая итоговая формула теоремы шаблонов:

$$P(H, t+1) \geq P(H, t)\, f(H, t)\, [1 - p_c\, \Delta(H)\, (1 - P(H, t)\, f(H, t) / \langle f(t) \rangle) / (L-1)]\, (1 - p_m)^{o(H)} / \langle f(t) \rangle$$

Полученное выражение не слишком удачно для анализа работы генетического алгоритма. Во-первых, в нем присутствует знак неравенства, связанный также с тем, что мы не учитывали случаи, когда рассматриваемый шаблон получается в результате кроссовера пары строк, не являющихся его представителями. Во-вторых, приспособленность шаблона и средняя приспособленность популяции быстро изменяются от поколения к поколению, поэтому полученное неравенство хорошо описывает ситуацию только для следующего

поколения.

Тем не менее, теорема шаблонов является хоть каким-то теоретическим обоснованием работы классического генетического алгоритма (следует заметить, что она верна только для классического ГА с его пропорциональным отбором и одноточечным кроссовером). На данный момент существуют более точные версии этой теоремы, а также другие рассуждения, доказывающие целесообразность использования генетических алгоритмов.

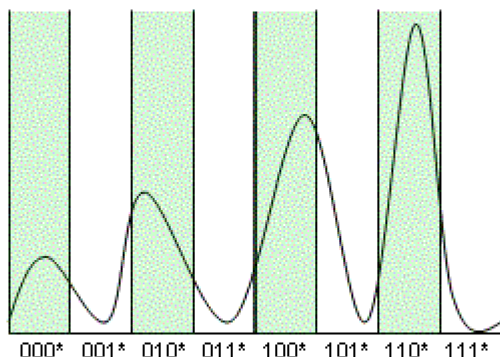
Строительные блоки

Из полученного в теореме шаблонов выражения видно, что шаблоны с малым порядком и малой определяющей длиной менее подвержены разрушению в результате кроссовера или мутации, поэтому рост (или уменьшение) их доли в популяции происходит динамичнее. Шаблоны с высокой приспособленностью, малым порядком и малой определяющей длиной называются *строительными блоками* (*building blocks*).

Холланд (1992) показал, что в то время, как ГА обрабатывает N строк на каждом поколении, в то же время неявно обрабатываются порядка N^3 гиперплоскостей. Это доказывается с рассчетом на реально применимые размеры популяции и длины строки. Практически это означает, что большая популяция имеет возможность локализовать решение быстрее, чем маленькая. Для оценки рекомендуемого размера популяции в зависимости от длины строки можно вспомнить, что всего гиперплоскостей 3^L.

Еще один аргумент в пользу больших популяций: в случае, если разброс приспособленностей представителей блока большой, то вероятность выбрать некоторое количество представителей блока с меньшей приспособленностью вместо представителей более хорошего достаточно велика, поскольку отдельные особи «слабого» блока могут оказаться лучше, чем многие особи «сильного». Увеличение размера популяции увеличит количество осуществляемых при генерации промежуточной популяции выборок, и вероятность сделать в итоге выбор неверного блока окажется достаточно малой.

В гипотезе о строительных блоках считается, что в процессе приближения популяции к глобальному оптимуму порядок и приспособленность строительных блоков увеличиваются. Это легко видно на простом примере:



Все локальные максимумы приведенной функции приходятся на блок **0*, а минимумы — на **1*, поэтому очевидно, что после отбора основная часть особей будут представителями первого блока. Левая половина графика в среднем ниже правой, поэтому доля блока 1*** будет преобладать над долей 0***. Получается, что основная масса особей окажутся представителями блока 1*** и в то же время **0*, значит, большое их количество будут представителями блока 1*0*. Теперь, выбирая между блоками 100* и 110*, получаем, что второй блок будет преобладать над первым. Таким образом, можно сказать, что хорошие строительные блоки малого порядка сложились в приспособленные блоки большего порядка, и в результате мы оказались в области глобального максимума, чем приблизились к решению задачи.

3.2. Настройка ГА

Генетический алгоритм производит поиск решений двумя методами одновременно: отбором гиперплоскостей (*hyperplane sampling*) и методом *hill-climbing*. Кроссовер осуществляет первый из них, поскольку комбинирует и совмещает шаблоны родителей в их детях. Мутация обеспечивает второй метод: особь случайным образом

изменяется, неудачные варианты вымирают, а если полученное изменение оказалось полезным, то, скорее всего, эта особь останется в популяции.

Возникает вопрос: какой же из этих методов лучше осуществляет поиск хороших решений? Исследования показали, что на простых задачах, таких, как максимизация унимодальной функции, ГА с мутацией (и без кроссовера) находят решение быстрее. Также для такого метода требуется меньший размер популяции. На сложных многоэкстремальных функциях лучше использовать ГА с кроссовером, поскольку этот метод более надежен, хотя и требует большего размера популяции.

С точки зрения теоремы шаблонов, мутация только вредит росту количества представителей хороших шаблонов, поскольку лишний раз их разрушает. Однако мутация просто необходима для ГА с малым размером популяции. Дело в том, что для малочисленных популяций свойственна *преждевременная сходимость* (*premature convergence*). Это ситуация, когда в некоторых позициях все особи имеют один и тот же бит, но такой набор битов не соответствует глобальному экстремуму. При этом кроссовер практически не изменяет популяции, т. к. все особи почти одинаковы. В этом случае мутация способна инвертировать «застрявший» бит у одной из особей и вновь расширить пространство поиска.

Введем понятие *давления отбора* (*selection pressure*) — это мера того, насколько различаются шансы лучшей и худшей особей популяции попасть в промежуточную популяцию. Для пропорционального отбора эта величина имеет свойство уменьшаться с увеличением средней приспособленности популяции. Действительно, при этом для каждой особи отношение $f/\langle f\rangle$ стремится 1, а значит шансы плохой и хорошей особей создать потомство уравниваются.

При увеличении p_c или p_m и при уменьшении давления отбора (например, за счет использования других стратегий отбора) размножение представителей приспособленных шаблонов замедляется, но зато происходит интенсивный поиск других шаблонов. Обратно, уменьшение p_c или p_m и увеличение давления отбора ведет к интенсивному использованию найденных хороших шаблонов, но

меньше внимания уделяется поиску новых. Таким образом, для эффективной работы генетического алгоритма необходимо поддерживать тонкое равновесие между *исследованием и использованием*. Это можно сформулировать также как необходимость *сбалансированной сходимости* ГА: быстрая сходимость может привести к схождению к неоптимальному решению, а медленная сходимость часто приводит к потере найденной наилучшей особи.

Методология управления сходимостью классического ГА до сих пор не выработана.

3.3. Другие модели ГА

Классический ГА хорош для понимания работы генетических алгоритмов, однако он не является наиболее эффективным из них. Сейчас мы рассмотрим различные варианты кодировки, генетические операторы и стратегии отбора, а также другие модели ГА.

Кодирование

Если сравнивать кодирование бинарным алфавитом и небинарным, то первый вариант обеспечивает лучший поиск с помощью гиперплоскостей, т. к. предоставляет максимальное их количество. Действительно, если требуется закодировать 2^L значений, то для бинарного алфавита количество гиперплоскостей будет 3^L, тогда как при использовании, к примеру, четырехзначного алфавита длина слов будет в 2 раза меньше, и гиперплоскостей будет $5^{L/2}$, т. е. меньше.

Еще один аргумент в пользу бинарных алфавитов — это то, что для встречаемости каждого символа в каждой позиции им требуется меньший размер популяции. Действительно, даже если имеется всего две строки, есть вероятность, что на каждой позиции в популяции есть и 0, и 1 (т. е. одна строка является результатом инвертирования другой). Если же алфавит большей мощности, то популяция из двух строк заведомо не будет содержать в каждой позиции несколько символов, и до применения мутации большая часть пространства поиска будет недоступна с точки зрения кроссовера. После применения мутации станет недоступна другая часть.

С другой стороны, небинарные алфавиты зачастую обеспечивают более наглядное представление решений задачи.

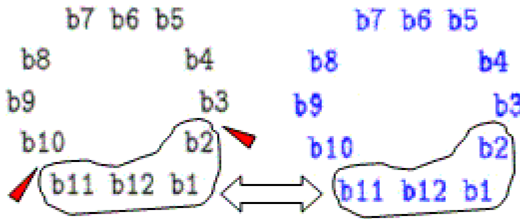
Исследования показали, что для большинства функций генетические алгоритмы будут работать лучше, если закодировать параметры в строку *кодом Грея*, а не прямым бинарным кодом. Это связано с т. н. *Hamming cliffs*, когда, к примеру, числа 7 и 8 различаются на 4 бита. Бинарное кодирование добавляет дополнительные разрывы, что осложняет поиск. Это можно показать на примере: пусть требуется минимизировать функцию $f(x) = x^2$. Если в популяции изначально преобладали отрицательные хорошие решения, то с большой вероятностью она сойдется к $-1 = 11\ldots1$. При этом достигнуть глобального минимума будет практически невозможно, поскольку любые изменения одного бита будут приводить к ухудшению решения. При кодировании кодом Грея такой проблемы не возникает.

Кодирование с плавающей точкой тоже является более удачным, чем прямое бинарное. На вопрос, лучше ли оно, чем кодирование кодом Грея, можно ответить, что на каких-то задачах лучше работает первый вариант, на других — второй. Как определить, какой вариант использовать для конкретной задачи, пока неизвестно.

Кроссовер

Одноточечный кроссовер мы рассмотрели выше.

При *двухточечном кроссовере* для родительской пары случайным образом выбираются 2 точки раздела, и родители обмениваются промежутками между ними. В результате получаются два ребенка. Удобно в этом случае представить строки в виде колец:



Определяющая длина в этом случае тоже измеряется в кольце, поэтому для такого шаблона, как 1*****1, при одноточечном кроссовере определяющая длина равна 6, и точка раздела всегда попадает между крайними фиксированными битами, а при двухточечном эта длина равна 1.

Следует заметить, что одноточечный кроссовер является частным случаем двухточечного, когда одна из точек раздела фиксирована.

Однородный кроссовер осуществляется следующим образом: один из детей наследует каждый бит с вероятностью p_0 у первого родителя, а иначе у второго. Второй ребенок получает все остальные не унаследованные биты. Обычно $p_0 = 0.5$. Для однородного кроссовера не важна определяющая длина шаблона, и вообще в большинстве случаев шаблон разрушается. Такой агрессивный оператор плохо предназначен для отбора гиперплоскостей, однако его применение оправдано при малом размере популяции, т. к. он препятствует преждевременному схождению, свойственному таким популяциям.

Стратегии отбора

Как мы уже отмечали выше, для пропорционального отбора свойственно уменьшение давления отбора с увеличением средней приспособленности популяции. Исправить этот недостаток можно с помощью масштабирования (*scaling*): на каждом поколении нулем приспособленности можно считать наихудшую особь.

Ранковый отбор (*rank selection*) отличается от пропорционального тем, что для каждой особи ее вероятность попасть в промежуточную

популяцию пропорциональна ее порядковому номеру в отсортированной по возрастанию приспособленности популяции. Такой вид отбора не зависит от средней приспособленности популяции.

Турнирный отбор (*tournament selection*) осуществляется следующим образом: из популяции случайным образом выбирается t особей, и лучшая из них помещается в промежуточную популяцию. Этот процесс повторяется N раз, пока промежуточная популяция не будет заполнена. Наиболее распространен вариант при $t = 2$. Турнирный отбор является более агрессивным, чем пропорциональный.

Отбор усечением (*truncation selection*): популяция сортируется по приспособленности, затем берется заданная доля лучших, и из них случайным образом N раз выбирается особь для дальнейшего развития.

Стратегии формирования нового поколения

Выделяют два типа формирования нового поколения после получения множества детей в результате кроссовера и мутации:

1. дети замещают родителей;
2. новое поколение составляется из совокупности и детей, и их родителей, например, выбором N лучших.

Также для формирования нового поколения возможно использование принципа *элитизма*: в новое поколение обязательно включается заданное количество лучших особей предыдущего поколения (часто одна лучшая особь). Использование второй стратегии и элитизма оказывается весьма полезным для эффективности ГА, т. к. не допускает потерю лучших решений. К примеру, если популяция сошлась в локальном максимуме, а мутация вывела одну из строк в область глобального, то при первой стратегии весьма вероятно, что эта особь в результате скрещивания будет потеряна, и решение задачи не будет получено. Если же используется элитизм, то полученное хорошее решение будет оставаться в популяции до тех пор, пока не будет найдено еще лучшее.

3.4. Некоторые модели генетических алгоритмов

Классический ГА был рассмотрен выше. Напомним, что его создал Holland (1975).

Genitor

Этот алгоритм был создан Уитли (D. Whitley). *Genitor*-подобные алгоритмы отличаются от классического ГА следующими тремя свойствами:

- На каждом шаге только *одна* пара случайных родителей создает только *одного* ребенка.
- Этот ребенок заменяет не родителя, а одну из худших особей популяции (в первоначальном *Genitor* — самую худшую).
- Отбор особи для замены производится по ее ранку (рейтингу), а не по приспособленности.

Утверждается (Syswerda, 1991), что в *Genitor* поиск гиперплоскостей происходит лучше, а сходимость быстрее, чем у классического ГА.

CHC

CHC расшифровывается как *Cross generational elitist selection, Heterogenous recombination, Cataclysmic mutation*. Этот алгоритм был создан Eshelman (1991) и характеризуется следующими параметрами:

- Для нового поколения выбираются *N* лучших *различных* особей среди родителей и детей. Дублирование строк не допускается.
- Для скрещивания выбирается случайная пара, но не допускается, чтобы между родителями было мало Хэммингово расстояние или мало расстояние между крайними различающимися битами.
- Для скрещивания используется разновидность однородного кроссовера *HUX* (*Half Uniform Crossover*): ребенку переходит ровно половина битов каждого родителя.
- Размер популяции небольшой, около 50 особей. Этим оправдано использование однородного кроссовера.

- СНС противопоставляет агрессивный отбор агрессивному кроссоверу, однако все равно малый размер популяции быстро приводит ее к состоянию, когда создаются только более или менее одинаковые строки. В таком случае СНС применяет *cataclysmic mutation*: все строки, кроме самой приспособленной, подвергаются сильной мутации (изменяется около трети битов). Таким образом, алгоритм перезапускается и далее продолжает работу, применяя только кроссовер.

Hybrid Algorithms

Идея *гибридных алгоритмов* (*hybrid algorithms*) заключается в сочетании генетического алгоритма с некоторым другим методом поиска, подходящим в данной задаче (зачастую это бывает *hill-climbing*). На каждом поколении каждый полученный потомок оптимизируется этим методом, после чего производятся обычные для ГА действия. При использовании *hill-climbing* получается, что каждая особь достигает локального максимума, вблизи которого она находится, после чего подвергается отбору, скрещиванию и мутации.

Такой вид развития называется Ламарковой эволюцией, при которой особь способна обучаться, а затем полученные навыки записывать в собственный генотип, чтобы потом передать их потомкам. И хотя такой метод ухудшает способность алгоритма искать решение с помощью отбора гиперплоскостей, однако на практике гибридные алгоритмы оказываются очень удачными. Это связано с тем, что обычно велика вероятность того, что одна из особей попадет в область глобального максимума и после оптимизации окажется решением задачи.

Генетический алгоритм способен быстро найти во всей области поиска хорошие решения, но он может испытывать трудности в получении из них наилучших. Такой метод, как *hill-climbing* быстро достигает локального максимума, однако не может искать глобальный. Сочетание этих двух алгоритмов способно использовать преимущества обоих.

3.5. Параллельные ГА

В природе все процессы происходят параллельно и независимо друг от

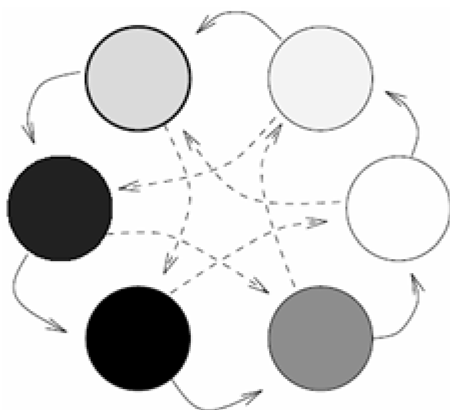
друга. Генетические алгоритмы тоже можно организовать как несколько параллельно выполняющихся процессов, и это увеличит их производительность.

Сделаем из классического ГА параллельный. Для этого будем использовать турнирный отбор. Заведем $N/2$ процессов (здесь и далее процесс подразумевается как некоторая машина, процессор, который может работать независимо). Каждый из них будет выбирать случайно из популяции 4 особи, проводить 2 турнира, и победителей скрещивать. Полученные дети будут записываться в новое поколение. Таким образом, за один цикл работы одного процесса будет сменяться целое поколение.

Island Models

Островная модель (*island model*) — это тоже модель параллельного генетического алгоритма. Она заключается в следующем: пусть у нас есть 16 процессов и 1600 особей. Разобьем их на 16 подпопуляций по 100 особей. Каждая их них будет развиваться отдельно с помощью некого генетического алгоритма. Таким образом, можно сказать, что мы расселили особи по 16-ти изолированным островам.

Изредка (например, каждые 5 поколений) процессы (или острова) будут обмениваться несколькими хорошими особями. Это называется миграция. Она позволяет островам обмениваться генетическим материалом.



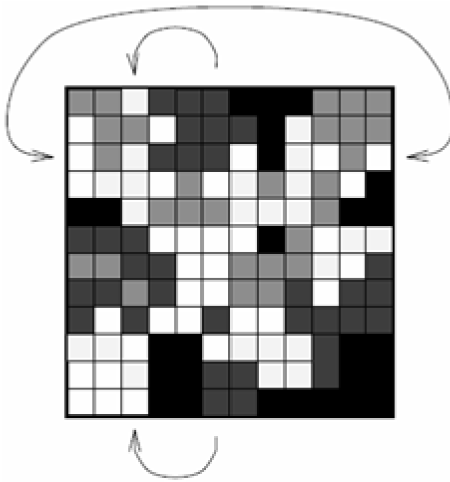
An Island Model Genetic Algorithm

Так как населенность островов обычно бывает невелика, подпопуляции будут склонны к преждевременной сходимости. Поэтому важно правильно установить частоту миграции. Чересчур частая миграция (или миграция слишком большого числа особей) приведет к смешению всех подпопуляций, и тогда островная модель будет несильно отличаться от обычного ГА. Если же миграция будет слишком редкой, то она не сможет предотвратить преждевременного схождения подпопуляций.

Генетические алгоритмы стохастичны, поэтому при разных его запусках популяция может сходиться к разным решениям (хотя все они в некоторой степени «хорошие»). Островная модель позволяет запустить алгоритм сразу несколько раз и пытаться совмещать «достижения» разных островов для получения в одной из подпопуляций наилучшего решения.

Cellular Genetic Algorithms

Cellular Genetic Algorithms — модель параллельных ГА. Пусть дано 2500 процессов, расположенных на сетке размером 50×50 ячеек, замкнутой, как показано на рисунке (левая сторона замыкается с правой, верхняя с нижней, получается тор).



A Cellular Genetic Algorithm

Каждый процесс может взаимодействовать только с четырьмя своими соседями (сверху, снизу, слева, справа). В каждой ячейке находится ровно одна особь. Каждый процесс будет выбирать лучшую особь среди своих соседей, скрещивать с ней особь из своей ячейки и одного полученного ребенка помещать в свою ячейку вместо родителя.

По мере работы такого алгоритма возникают эффекты, похожие на островную модель. Сначала все особи имеют случайную приспособленность (на рисунке она определяется по цвету). Спустя несколько поколений образуются небольшие области похожих особей с близкой приспособленностью. По мере работы алгоритма эти области растут и конкурируют между собой.

Другие модели ГА

До сих пор мы рассматривали ГА с фиксированными параметрами, такими как размер популяции, длина строки, вероятность кроссовера и мутации. Однако существуют генетические алгоритмы, в которых эти параметры могут изменяться и подстраиваться.

К примеру, пусть вероятность мутации для каждой особи будет отдельной. Можно добавить к строке особи подстроку, кодирующую эту вероятность. При вычислении приспособленности эта подстрока будет игнорироваться, но она будет подвергаться кроссоверу и мутации так же, как и остальная часть строки. Вероятность каждого бита данной особи быть инвертированным при мутации будет равна значению, кодируемому добавленной подстрокой. Инициализируются вероятности мутации случайным образом.

Thomas Back (1992) в своей работе заметил, что для унимодальных функций вариант с глобальной вероятностью мутации работает лучше, однако для многоэкстремальных функций использование адаптивной мутации дает лучшие результаты.

3.6. Наблюдения

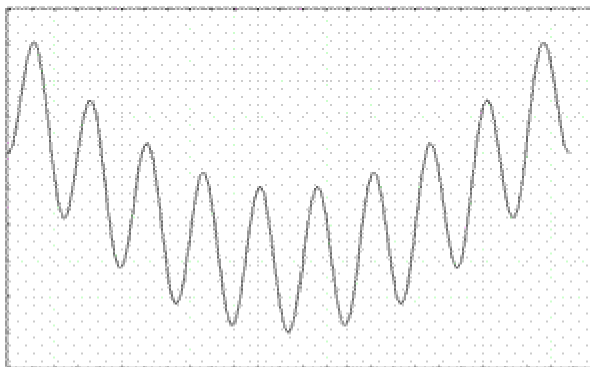
Укажем некоторые наблюдения, полученные исследователями генетических алгоритмов.

Факторы, создающие сложность для ГА

Как и для любого алгоритма оптимизации, для генетических алгоритмов есть некоторые типы функций, с которыми им работать сложнее, чем с другими. Обычно ГА тестируют именно на таких функциях. Ниже приведены свойства функций приспособленности, создающие сложность для ГА.

- *Многоэкстремальность*: это проблема для любого метода поиска, т. к. создается множество ложных аттракторов. Пример — функция Растригина (Растригин Л.А.):

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



Минимум достигается при всех $x_i = 0$. Количество аргументов функции можно варьировать, тем самым повышая или понижая сложность задачи поиска минимума. На картинке изображен график функции Растригина с одним аргументом.

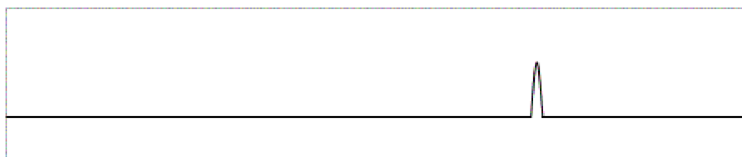
Обманчивость (*deception*) — это характеристика функции, построенной так, что шаблоны малого порядка уводят популяцию к локальному экстремуму. Пример: пусть строка состоит из 10-ти четырехбитных подстрок. Пусть u_i равно количеству единиц в i-той подстроке. Зададим функцию $g(u)$ следующей таблицей:

u	0	1	2	3	4
***g*(*u*)**	3	2	1	0	4

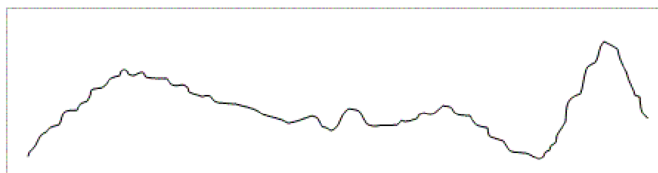
- и пусть функция приспособленности равна сумме $g(u_i)$ по всем $i = 1..10$:
  $$f = \sum_{i=1}^{10} g(u_i)$$
-
- Локальный максимум достигается при всех битах, равных 0, глобальный — при всех 1. В большинстве случаев при добавлении единицы в подстроку приспособленность особи будет падать (за исключением случая, когда все остальные биты подстроки уже равны 1). При замене 1 на 0 она будет

расти. Поэтому с большой вероятностью популяция сойдется к решению, при котором большинство подстрок будут состоять из всех нулей, и лишь некоторые из всех единиц. Однако это не будет глобальным максимумом. Из этого решения попасть в глобальный максимум, т. е. заменить все нули единицами, для ГА будет сложно.

- *Изолированность* («поиск иголки в стоге сена») — также проблема для любого метода оптимизации, поскольку функция не предоставляет никакой информации, подсказывающей, в какой области искать максимум. Лишь случайное попадание особи в глобальный экстремум может решить задачу.



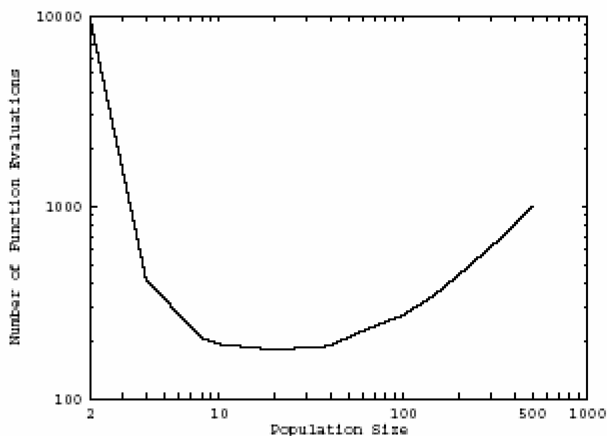
- *Дополнительный шум* (*noise*) разбрасывает значения приспособленности шаблонов, поэтому часто даже хорошие гиперплоскости малого порядка не проходят отбор, что замедляет поиск решения.



Размер популяции

Для того, чтобы получить хорошие результаты, необходимо правильно выбрать размер популяции. На графике изображена зависимость количества вычислений функции приспособленности для нахождения максимума унимодальной функции от размера популяции. Видно, что существует оптимальный размер популяции. Действительно, если популяция мала, то при заданном ограничении количества вычислений

функции приспособленности (а значит, фиксированном времени вычислений) она успеет создать большее количество поколений, но вероятнее всего преждевременно сойдется. Слишком большая популяция должна найти решение, но она может не успеть достичь этого момента, т. к. ей отведено малое количество поколений.



Для алгоритмов с кроссовером (т. е. без мутации) существуют оценки оптимального размера, а для ГА с мутацией (и без кроссовера) их пока нет. Однако эксперименты показывают, что для них оптимальный размер популяции тоже существует. В любом случае, он зависит от задачи.

3.7. Основная теорема о генетических алгоритмах

Для того чтобы лучше понять функционирование генетического алгоритма, будем использовать понятие *схема* и сформулируем основную теорему, относящуюся к генетическим алгоритмам и называемую теоремой о схемах . Понятие схема было введено Холландом и используется для анализа работы ГА. В частности рассматриваются процессы конструирования и разрушения определенной схемы в течение развития популяции (schema dynamics).

Схемой называется строка вида

$(a_1, a_2, ..., a_i, ..., a_l), a_i \in \{0, 1, *\}$.

Символом "*" в некотором разряде обозначается то, что там может быть как 1, так и 0. Например, для двух бинарных строк "111000111000" и "110011001100" схема будет выглядеть следующим образом: "11*0****1*00". Т.е. с помощью схем можно как бы выделять общие участки двоичных строк и маскировать различия. Имея в составе схем m символов "*" можно закодировать (обобщить) 2^m двоичных строк. Так, например, схема "01*0*1" описывает набор строк

{"010001", "010011", "011001", "011011"}.

Опеределяющей длиной схемы (schema defining length) называется расстояние между двумя крайними символами "0" и/или "1". Для схемы "01*0*1" определяющая длина равна 5, а для схемы "**0**1*" определяющая длина равна 3. *Порядок схемы (schema order)* - это ещё одна характеристика схемы и равна она числу фиксированных позиций в строке, т.е. общему числу "0" и "1". Для схем "01*0*1" и "**0**1*" порядки равны 4 и 2 соответственно.

Теперь о том, какое отношение схема имеет к генетическим алгоритмам. Дело в том, что хромосома, по сути, является двоичной строкой. В то же время особи, которой принадлежит хромосома, содержащая набор генов-параметров задачи, поставлена в соответствие величина, характеризующая её приспособленность. Т.к. схема является обобщением нескольких бинарных строк (хромосом), то можно говорить, что особи, обладающие хромосомами, которые соответствуют одной схеме более приспособленны, а особи с хромосомами, соответствующими другой схеме - менее приспособлены.

Можно сказать, что смысл работы ГА заключается в поиске двоичной строки определенного вида из всего множества бинарных строк. Пространство поиска составляет 2^L строк, а его мерность равна L (L-мерное пространство), где L - длина хромосомы. Схема соответствует некоторой гиперплоскости в этом пространстве. Данное утверждение можно проиллюстрировать следующим образом. Пусть разрядность

хромосомы равна 3, тогда всего можно закодировать $2^3=8$ строк. Представим куб в 3-мерном пространстве. Обозначим вершины этого куба 3-разрядными бинарными строками так, чтобы метки соседних вершин отличались ровно на один разряд, причем вершина с меткой "000" находилась бы в начале координат. Вариант обозначения изображен на рис.1.

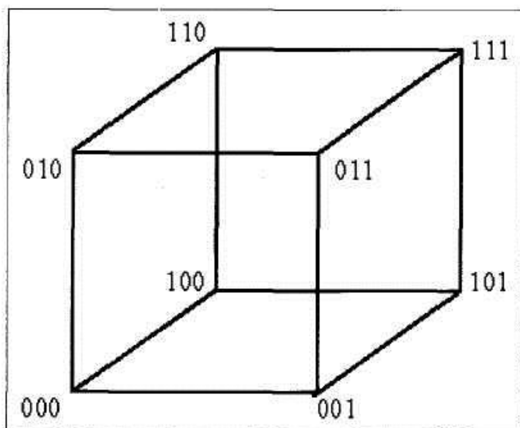


Рис.1. 3-мерный куб

Если взять схему вида "**0", то она опишет левую грань куба, а схема "*10" - верхнее ребро этой грани. Очевидно, что схема "***" соответствует всему пространству. Если взять двоичные строки длиной 4 разряда, то разбиение пространства схемами можно изобразить на примере 4-мерного куба с поименованными вершинами (рис.2).

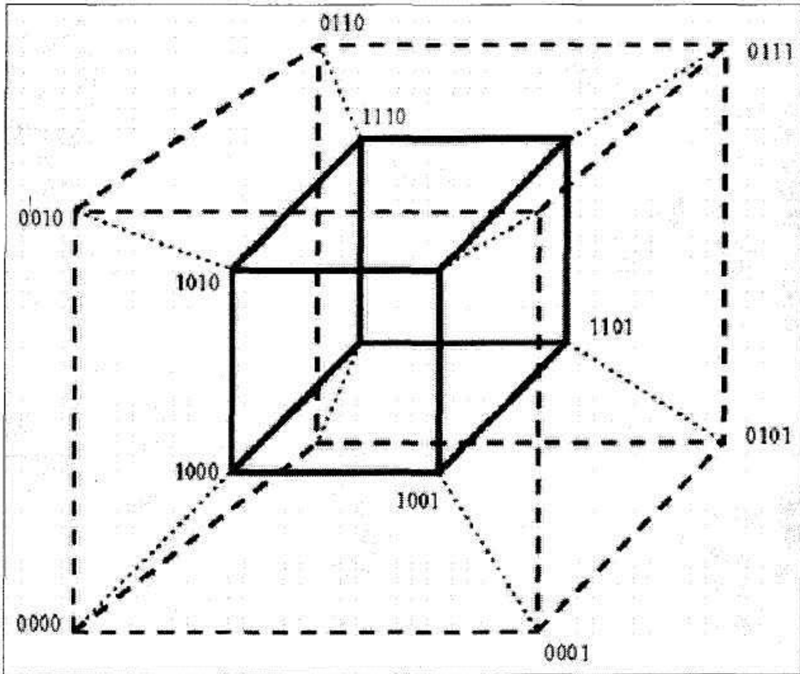


Рис.2. 4-мерный куб

Здесь схеме "*1**" соответствует гиперплоскость, включающая задние грани внешнего и внутреннего куба, а схеме "**10" - гиперплоскость с верхними ребрами левых граней обоих кубов.

Разбиение пространства поиска можно представить и по другому. Представим координатную плоскость, в которой по одной оси мы будем откладывать значения двоичных строк, а по другой - значение целевой функции (рис. 3).

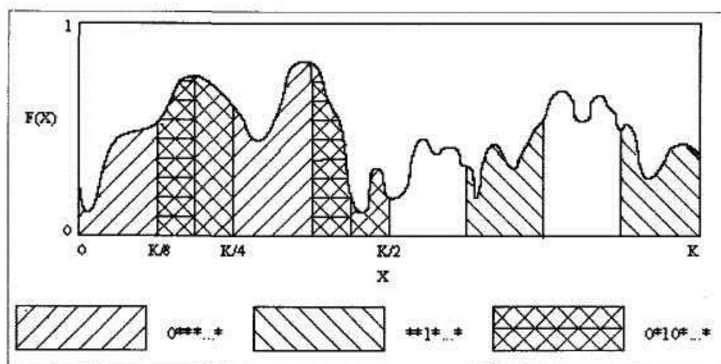


Рис. 3. Разбиение пространства

Участки пространства, заштрихованные разным стилем, соответствуют разным схемам. Число К в правой части горизонтальной оси соответствует максимальному значению бинарной строки - "111...111". Из рисунка видно, что схема "0***...*" покрывает всю левую половину отрезка, схема "**1*...*" - 4 участка шириной в одну восьмую часть, а схема "0*10*...*" - левые половины участков, которые находятся на пересечении первых двух схем. Таким образом и происходит разбиение пространства в этом случае.

Как мы уже говорили, понятие *схема* было введено для определения множества хромосом, обладающих некоторыми общими свойствами, т.е. подобных друг другу. Если аллели принимают значения 0 или 1 (рассматриваются хромосомы с двоичным алфавитом), то схема представляет собой множество хромосом, содержащих нули и единицы на некоторых заранее определенных позициях. При рассмотрении схем удобно использовать расширенный алфавит {0, 1, *}, в который помимо 0 и 1 введен дополнительный символ *, обозначающий любое допустимое значение, т.е. 0 или 1; символ * в конкретной позиции означает «все равно» *(don't care).* Например,
10*1 ={1001, 1011}
*01*10 = {001010, 001110, 101010,101110}
Считается, что *хромосома принадлежит к данной схеме,* если для каждой j-й позиции (локуса), j = 1, 2, ..., L, где L - длина хромосомы;

символ, занимающий *j*-ю позицию хромосомы, соответствует символу, занимающему *j*-ю позицию схемы, причем символу * соответствуют как 0, так и 1. То же самое означают утверждения *хромосома соответствует схеме* и *хромосома представляет схему.* Отметим, что если в схеме присутствует *m* символов *, то эта схема содержит 2^m хромосом. Кроме того, каждая хромосома (цепочка) длиной *L* принадлежит к 2^L схемам. В таблицах 2 и 3 представлены схемы, к которым принадлежат цепочки длиной 2 и 3 соответственно.

Таблица 2. Схемы, к которым принадлежат цепочки длиной 2

Звенья	Схемы			
	1	2	3	4
00	**	*0	0*	00
01	**	*1	0*	01
10	**	*0	1*	10
11	**	*1	1*	11

Таблица 3. Схемы, к которым принадлежат цепочки длиной 3

Звенья	Схемы							
	1	2	3	4	5	6	7	8
000	***	**0	*0*	0**	*00	0*0	00*	000
001	***	**1	*0*	0**	*01	0*1	00*	001
010	***	**0	*1*	0**	*10	0*0	01*	010
011	***	**1	*1*	0**	*11	0*1	01*	011
100	***	**0	*0*	1**	*00	1*0	10*	100
101	***	**1	*0*	1**	*01	1*1	10*	101
110	***	**0	*1*	1**	*10	1*0	11*	110
111	***	**1	*1*	1**	*11	1*1	11*	111

Цепочки длиной 2 соответствуют четырем различным схемам, а цепочки длиной 3 - восьми схемам.

Генетический алгоритм базируется на принципе трансформации наиболее приспособленных особей (хромосом). Пусть P(0) означает исходную популяцию особей, а *P(k)* - текущую популяцию (на *k*-й итерации алгоритма). Из каждой популяции *P(k), k* = 0, 1, ... методом селекции выбираются хромосомы с наибольшей приспособленностью,

которые включаются в так называемый родительский пул *(mating pool)* *M(k).* Далее, в результате объединения особей из популяции *M(k)* в родительские пары и выполнения операции скрещивания с вероятностью p_c, а также операции мутации с вероятностью p_m формируется новая популяция $P(k+1)$, в которую входят потомки особей из популяции *M(k).*

Следовательно, для любой схемы, представляющей хорошее решение, было бы желательным, чтобы количество хромосом, соответствующих этой схеме, возрастало с увеличением количества итераций *k.*

На соответствующее преобразование схем в генетическом алгоритме оказывают влияние 3 фактора: селекция хромосом, скрещивание и мутация. Проанализируем воздействие каждого из них с точки зрения увеличения ожидаемого количества представителей отдельно взятой схемы.

Обозначим рассматриваемую схему символом *S*, а количество хромосом популяции *P(k),* соответствующих этой схеме - *c(S, k).* Следовательно, *c(S, k)* можно считать количеством элементов (т.е. мощностью) множества $P(k) \cap S$.

Начнем с исследования влияния селекции. При выполнении этой операции хромосомы из популяции *P(k)* копируются в родительский пул *M(k)* с вероятностью, определяемой выражением (3.3). Пусть *F(S, k)* обозначает среднее значение функции приспособленности хромосом из популяции *P(k),* которые соответствуют схеме S. Если

$P(k)S = \{ch_1.....ch_{c(S,k)}\}$,

то

$$c(S,k)=\frac{\sum_{i=1}^{c(S,k)} F(ch_i)}{c(S,k)} \quad (3.6)$$

Величина *F(S, k)* также называется приспособленностью схемы S на k-й итерации.

Пусть $\Im$ (k) обозначает сумму значений функций приспособленности хромосом из популяции *P(k)* мощностью N, т.е.

$$\Im(k) = \sum_{i=1}^{N} F(ch_i^{(k)}) \quad (3.7)$$

Обозначим через $\overline{F}$ (k) среднее значение функции приспособленности хромосом этой популяции, т.е.

$$\overline{F}(k) = \frac{1}{N}\Im(k). \quad (3.8)$$

Пусть $ch_i^{(k)}$ обозначает элемент родительского пула *M(k).* Для каждого $ch_r^{(k)} \in M(k)$ и для каждого $i = 1,..., c(S, к)$ вероятность того, что $ch_r^{(k)}=ch_i$ определяется отношением $F(ch_i)/ F(k)$. Поэтому ожидаемое количество хромосом в популяции *M(k),* которые равны ch_i, составит

$$N\frac{F(ch_i)}{\Im(k)} = \frac{F(ch_i)}{\overline{F}(k)}$$

Таким образом, ожидаемое количество хромосом из множества $P(k) \cap S$, отобранных для включения в родительский пул *M(k),* будет равно

$$\sum_{i=1}^{c(S,k)} \frac{F(ch_i)}{\overline{F}(k)} = c(S,k)\frac{F(S,k)}{\overline{F}(k)},$$

что следует из выражения (3.6).

Поскольку каждая хромосома из родительского пула *M(k)* одновременно принадлежит популяции *P(k),* то хромосомы, составляющие множество $M(k) \cap S$ - это те же самые особи, которые были отобраны из множества $P(k) \cap S$ для включения в популяцию *M(k).* Если количество хромосом родительского пула *M(k),* соответствующих схеме S (т.е. количество элементов множества $M(k) \cap S$), обозначить *b(S, k),* то из приведенных рассуждений можно сделать следующий вывод:

Вывод 3.1

Ожидаемое значение *b(S, k),* т.е. ожидаемое значение количества хромосом родительского пула *M(к),* соответствующих схеме S, определяется выражением

$$E[b(S,k)] = c(S,k)\frac{F(S,k)}{\overline{F}(k)}. \quad (3.9)$$

Из этого следует, что если схема S содержит хромосомы со значением функции приспособленности, превышающим среднее значение (т.е. приспособленность схемы S на k-й итерации оказывается большей, чем среднее значение функции приспособленности хромосом из популяции *P(k),* и поэтому

$F(S, k)/F(k)>1$), то ожидаемое количество хромосом из родительского пула *M(k),* соответствующих схеме S, будет больше количества

хромосом из популяции *P(k),* соответствующих схеме S. Поэтому можно утверждать, что селекция вызывает распространение схем с приспособленностью «лучше средней» и исчезновение схем с «худшей» приспособленностью.

Прежде чем приступить к анализу влияния генетических операторов скрещивания и мутации на хромосомы из родительского пула, определим необходимые для дальнейших рассуждений понятия *порядка* и *охвата схемы.* Пусть *L* обозначает длину хромосом, соответствующих схеме S.

Определение 3.1

Порядок (order) схемы *S,* иначе называемый счетностью схемы и обозначаемый *o(S)* - это количество постоянных позиций в схеме, т.е. нулей и единиц в случае алфавита {0, 1, *}. Например,

o(10*1) = 3 *o*(*01*10) = 4 *o*(**0*Г) = 2 *o*(*101**) = 3

Порядок схемы *o*(S) равен длине *L* за вычетом количества символов *, что легко проверить на представленных примерах (для *L* = 4 с одним символом * и для *L* = 6 с двумя, четырьмя и тремя символами *). Легко заметить, что порядок схемы, состоящей исключительно из символов *, равен нулю, т.е.

o(****) = 0, а порядок схемы без единого символа * равен *L;* например, *o*(10011010) = 8. Порядок схемы *o*(S) - это всегда целое число из интервала [0, *L*].

Определение 3.2

Охват (defining length) схемы S, называемый также *длиной* схемы (не путать с длиной *L)* и обозначаемый *d(S)* - это расстояние между первым и последним постоянным символом (т.е. разность между правой и левой крайними позициями, содержащими постоянные символы). Например,

d(10*1) = 4-1 =3

d(**0*1*) = 5-3 = 2

d(*01*10) = 6-2 = 4

d(*101**) = 4-2 = 2

Охват схемы *d(S)* - это целое число из интервала [0, *L* - 1]. Отметим, что охват схемы с постоянными символами на первой и последней позиции равен *L* - 1 (как в первом из приведенных примеров). Охват схемы с единственной постоянной позицией равен нулю, в частности, *d(**1*)*=0. Охват характеризует содержательность информации, заключенной в схеме.

Перейдем к рассуждениям о влиянии операции скрещивания на обработку схем в генетическом алгоритме. Прежде всего отметим, что

одни схемы оказываются более подверженными уничтожению в процессе скрещивания, чем другие. Например, рассмотрим схемы S_1 = 1****0* и S_2 = **01***, а также хромосому ch=[1001101], соответствующую обеим схемам. Видно, что схема S_2 имеет больше шансов «пережить» операцию скрещивания, чем схема S_1, которая больше подвержена «расщеплению» в точках скрещивания 1, 2, 3, 4 и 5. Схему S_2 можно разделить только при выборе точки скрещивания, равной 3. Обратим внимание на охват обеих схем, который - это очевидно - оказывается существенным в процессе скрещивания.
В ходе анализа влияния операции скрещивания на родительский пул *M(k)* рассмотрим некоторую хромосому из множества
$M(k) \cap S$, т.е. хромосому из родительского пула, соответствующую схеме S. Вероятность того, что эта хромосома будет отобрана для скрещивания, равна p_c. Если ни один из потомков этой хромосомы не будет принадлежать к схеме S, то это означает, что точка скрещивания должна находиться между первым и последним постоянным символом данной схемы. Вероятность этого равна $d(S)l(L$-1). Из это можно сделать следующие выводы:

Вывод 3.2 (влияние скрещивания)

Для некоторой хромосомы из $M(k) \cap S$ вероятность того, что она будет отобрана для скрещивания и ни один из ее потомков не будет принадлежать к схеме S, ограничена сверху величиной

$$p_c \frac{d(S)}{L-1}$$

Эта величина называется *вероятностью уничтожения схемы S.*

Вывод 3.3

Для некоторой хромосомы из $M(k) \cap S$ вероятность того, что она не будет отобрана для скрещивания либо, что хотя бы один из ее потомков после скрещивания будет принадлежать к схеме S, ограничена снизу величиной

$$1-p_c \frac{d(S)}{L-1}$$

Эта величина называется *вероятностью выживания схемы S.*

Легко показать, что если данная хромосома принадлежит к схеме S и отбирается для скрещивания, а вторая родительская хромосома также принадлежит к схеме S, то оба их потомка тоже будут принадлежать к схеме S. Выводы 3.2 и 3.3 подтверждают значимость показателя охвата

схемы *d(S)* для оценки вероятности уничтожения или выживания схемы.

Рассмотрим теперь влияние мутации на родительский пул *M(k)*. Оператор мутации с вероятностью p_m случайным образом изменяет значение в конкретной позиции с 0 на 1 и обратно. Очевидно, что схема переживет мутацию только в том случае, когда все ее постоянные позиции останутся после выполнения этой операции неизменными.

Хромосома из родительского пула, принадлежащая к схеме S (т.е. хромосома из множества $M(k) \cap S$) останется в этой схеме тогда и только тогда, когда ни один символ этой хромосомы, соответствующий постоянным символам схемы S, не изменится в процессе мутации. Вероятность такого события равна

$(1\text{-}p_m)^{o(S)}$. Этот результат можно представить в форме следующего вывода:

Вывод 3.4 (влияние мутации)

Вероятность того, что некоторая хромосома из $M(k) \cap S$ будет принадлежать к схеме S после операции мутации, определяется выражением

$(1\text{-}p_m)^{o(S)}$.

Эта величина называется *вероятностью выживания схемы S после мутации.*

Вывод 3.5

Если вероятность мутации p_m мала ($p_m \ll 1$), то можно считать, что вероятность выживания схемы S после мутации, определенная в выводе 3.4, приближенно равна

$1 - p_m\, o(S)$.

Эффект совместного воздействия селекции, скрещивания и мутации (выводы 3.1 - 3.4) с учетом факта, что если хромосома из множества $M(k) \cap S$ дает потомка, соответствующего схеме S, то он будет принадлежать к $P(k+1) \cap S$, ведет к построению следующей схемы репродукции :

$$E[c(S,k + 1)] \geq c(S,k)\, \frac{F(S,k)}{\overline{F}(k)} \left(1\text{-}p_c \frac{d(S)}{L-1}\right)(1\text{-}p_m)^{o(s)} . \quad (3\ 10)$$

Зависимость (3.10) показывает, как изменяется от популяции к популяции количество хромосом, соответствующих данной схеме. Это изменение вызывается тремя факторами, представленными в правой части выражения (3.10), в частности: *F(S,k)IF(k)* отражает роль среднего значения функции приспособленности, $1\text{-}p_c d(S)/(L\text{-}1)$ показывает влияние скрещивания и $(1\text{-}p_m)^{o(S)}$ - влияние мутации. Чем

больше значение каждого из этих факторов, тем большим оказывается ожидаемое количество соответствий схеме S в следующей популяции. Вывод 3.5 позволяет представить зависимость (3.10) в виде

$$E[c(S,k+1)] \geq c(S,k)\ \frac{F(S,k)}{\overline{F}(k)}\left(1-p_c\frac{d(S)}{L-1}-p_m o(s)\right). \quad (3.11)$$

Для больших популяций зависимость (3.11) можно аппроксимировать выражением

$$c(S,k+1) \geq c(S,k)\ \frac{F(S,k)}{\overline{F}(k)}\left(1-p_c\frac{d(S)}{L-1}-p_m o(s)\right). \quad (3.12)$$

Из формул (3.11) и (3.12) следует, что ожидаемое количество хромосом, соответствующих схеме S в следующем поколении, можно считать функцией от фактического количества хромосом, принадлежащих этой схеме, относительной приспособленности схемы, а также порядка и охвата схемы. Заметно, что схемы с приспособленностью выше средней и с малым порядком и охватом характеризуются возрастанием количества своих представителей в последующих популяциях. Подобный рост имеет показательный характер, что следует из выражения (3.9). Для больших популяций эту формулу можно заменить рекуррентной зависимостью вида

$$c(S,k+1) = c(S,k)\ \frac{F(S,k)}{\overline{F}(k)} \quad (3.13)$$

Если допустить, что схема S имеет приспособленность на ε% выше средней, т.е.

$$F(S,k) = \overline{F}(k) + \varepsilon\overline{F}(k), \quad (3.14)$$

то при подстановке выражения (3.12) в неравенство (3.11) в предположении, что ε не изменяется во времени, при старте от $k = 0$ получаем

$$c(S, k) = c(S, 0)(1 + \varepsilon)^k,$$

$$\varepsilon = (F(S,k) - \overline{F}(k))/\ \overline{F}(k), \quad (3.15)$$

т.е. $\varepsilon > 0$ для схемы с приспособленностью выше средней и $\varepsilon < 0$ - в противном случае.

Равенство (3.15) описывает геометрическую прогрессию. Из этого следует, что в процессе репродукции схемы, оказавшиеся лучше (хуже) средних, выбираются на очередных итерациях генетического алгоритма в показательно возрастающих (убывающих) количествах. Обратим внимание, что зависимости (3.9) - (3.13) основаны на

предположении, что функция приспособленности F принимает только положительные значения. При использовании генетических алгоритмов для решения оптимизационных задач, в которых целевая функция может принимать и отрицательные значения, необходимы некоторые дополнительные соотношения между оптимизируемой функцией и функцией приспособленности. Конечный результат, получаемый из выражений (3.10) - (3.12), можно сформулировать в форме теоремы. Это основная теорема генетических алгоритмов, иначе называемая *теоремой о схемах* .

Теорема 3.1

Схемы малого порядка, с малым охватом и с приспособленностью выше средней формируют показательно возрастающее количество своих представителей в последующих поколениях генетического алгоритма.

В соответствии с приведенной теоремой важным вопросом становится кодирование, которое должно обеспечивать построение схем малого порядка, с малым охватом и с приспособленностью выше средней. Косвенным результатом теоремы 3.1 (о схемах) можно считать следующую гипотезу, называемую *гипотезой о кирпичиках* (либо о *строительных блоках)*.

Гипотеза 3.1

Генетический алгоритм стремится достичь близкого к оптимальному результата за счет комбинирования хороших схем (с приспособленностью выше средней) малого порядка и малого охвата. Такие схемы называются *кирпичиками* (либо *строительными блоками).*

Гипотеза о строительных блоках выдвинута на основании теоремы о схемах с учетом того, что генетические алгоритмы исследуют пространство поиска с помощью схем малого порядка и малого охвата, которые впоследствии участвуют в обмене информацией при скрещивании.

Несмотря на то, что для доказательства этой гипотезы предпринимались определенные исследования, однако в большинстве нетривиальных приложений приходится опираться на эмпирические результаты. В течение последних двадцати лет опубликованы многочисленные работы, посвященные применениям генетических алгоритмов, подтверждающим эту гипотезу. Если она считается истинной, то проблема кодирования приобретает критическое значение для генетического алгоритма; кодирование должно реализовать концепцию малых строительных блоков. Качество, которое

обеспечивает генетическим алгоритмам явное преимущество перед другими традиционными методами, несомненно заключается в обработке большого количества различных схем.
Обратимся снова к примерам 1 и 2 и на их основе проанализируем обработку схем генетическим алгоритмом.

Пример 4.

В условиях примера 1 рассмотрим схему

S_0 = **********11

и покажем, как изменяется количество представителей этой схемы и приспособленность в процессе выполнения генетического алгоритма. Длина L = 12, а охват и порядок схемы S_0 составляют соответственно $d(S_0)=1$ и $o(S_0)=2$. В исходной популяции из примера 1 схеме S_0 соответствуют две следующие хромосомы:

ch_3 = [011101110011]

ch_7 = [101011011011]

Из формулы (3.10) следует, что после селекции и скрещивания количество хромосом, соответствующих схеме S_o, должно быть больше или равно 2,5. Напомним, что вероятности скрещивания и мутации считаются равными соответственно $p_c = 1$ и $p_m = 0$. Приспособленность схемы S_o в исходной популяции, обозначаемая $F(S_0, 0)$, равна 8 и превышает среднюю приспособленность всех хромосом этой популяции F = 5,75, что легко рассчитать по формулам (3.6) - (3.8).
В примере 1 после селекции и скрещивания в новой популяции получены четыре хромосомы, соответствующие схеме S_o:

Ch_1 = [001111011011]

Ch_3 = [111011011011]

Ch_7 = [011101011011]

Ch_8 = [101011110011]

Приспособленность схемы S_o в новой популяции, т.е. $F(S_0, 1)$, составит 8,25, тогда как средняя приспособленность хромосом этой популяции F(1)=7, что также следует из формул (3.6) - (3.8). Новая популяция характеризуется большим средним значением функции приспособленности особей по сравнению с предыдущей (исходной) популяцией, что уже отмечалось в примере 1. Кроме того, в новой популяции приспособленность схемы S_o оказывается лучшей, а количество представителей этой схемы - большим по сравнению с предыдущей популяцией.

Пример 5

В условиях примера 2 рассмотрим схему

S_1=1****

и проследим ее обработку при выполнении генетического алгоритма.
В этом случае L=5, а охват и порядок схемы S_1, составляют $d(S_1) = 0$ и $o(S_1) = 1$ соответственно. В исходной популяции из примера 2 этой схеме соответствуют три хромосомы
ch_1 =[10011]
ch_4 =[10101]
ch_6 =[11101]
Приспособленность схемы S_1, в исходной популяции
$F(S_1, 0)$=1096 и превышает среднюю приспособленность особей этой популяции $F(0) = 589$, что следует из выражений (3.6) - (3.8). На основе формулы (3.9) легко рассчитать ожидаемое количество хромосом родительского пула, соответствущих схеме S_1. Оно составит 3 * 1096/589 = 5,58. В примере 2 по результатам селекции в родительский пул включены 6 таких хромосом: ch_6, ch_4, ch_6, ch_1, ch_4, ch_6. Ожидаемое количество хромосом, соответствущих схеме S_1, после скрещивания с вероятностью p_c=1 (вероятность мутации p_m=0), как легко рассчитать по формуле (3.10), должно превышать 5,58. В новую популяцию включены 6 представителей схемы S_1. Это все хромосомы данной популяции.

Пример 6

В условиях примера 2 рассмотрим схему
S_2 = 11***
и проследим ее обработку при выполнении генетического алгоритма.
Длина $L = 5$, а охват и порядок схемы S_2 составляют $d(S_2)= 1$ и $o(S_2)=2$ соответственно. В исходной популяции из примера 2 этой схеме соответствует одна хромосома
ch_6 = [11101].
Поэтому приспособленность схемы S_2 в исходной популяции равна функции приспособленности хромосомы ch_6 и составляет 1683. Она превышает среднюю приспособленность особей исходной популяции, равную 589. По формуле (3.9) рассчитываем ожидаемое количество хромосом родительского пула, соответствущих схеме S_2. Оно составит 1683/589 = 2,86. В примере 2 по результатам селекции в родительский пул включены 3 одинаковых хромосомы [11101], соответствущих схеме S_2. Ожидаемое количество хромосом в новой популяции, соответствущих схеме S_2, после скрещивания с вероятностью
$p_c = 1$ (вероятность мутации $p_m = 0$), должно превышать 5,58. В примере 2 в новую популяцию включены 3 хромосомы, соответствущих схеме S_2. Это
Ch_4=Ch_5 = Ch_6 = [11101].

Пример 7

В условиях примера 2 рассмотрим схему

S_3 = ***11

и проследим ее обработку при выполнении генетического алгоритма. Длина L= 5, а охват и порядок схемы S_3 составляют $d(S_3) = 1$ и $o(S_3) = 2$ соответственно. В исходной популяции из примера 2 этой схеме соответствуют три хромосомы

ch_1 = [10011]

ch_2 = [00011]

ch_3 = [00111]

В отличие от примеров 5 и 6 приспособленность схемы S_3 в исходной популяции оказывается меньше средней приспособленности особей этой популяции $\overline{F}(0)=589$ и составляет $F(S_3, 0)=280$. Ожидаемое количество хромосом родительского пула, соответствующих схеме S_3 и рассчитанное по формуле (3.9), равно 3 * 280/589 = 1,426. В примере 2 в родительский пул была включена одна хромосома [10011], соответствующая схеме S_3. На основе формулы (3.10) получаем значение 1,068, определяющее количество представителей схемы S_3 в новой популяции. В примере 2 после скрещивания с вероятностью $p_c = 1$ (вероятность мутации $p_m = 0$) в новую популяцию была включена одна хромосома, соответствующая схеме S_3, т.е. Ch_2 = [10111].

Пример 8

В условиях примера 2 рассмотрим схему

S_4 = *10**

и проследим ее обработку при выполнении генетического алгоритма. Длина $L = 5$, а охват и порядок схемы S_4 составляют $d(S_4)=1$ и $o(S_A) = 2$ соответственно. В исходной популяции из примера 2 этой схеме соответствует только одна хромосома ch_5 = [01000]

Поэтому приспособленность схемы S_4 в исходной популяции равна значению функции приспособленности хромосомы ch_5 и составляет 129. Аналогично примеру 7, она меньше средней приспособленности особей начальной популяции, которая равна 589. Ожидаемое количество представителей схемы S_4 в родительском пуле составляет 129/589 = 0,22, что следует из формулы (3.9). В примере 2 родительский пул (после селекции) не содержит ни одной хромосомы, соответствующей схеме S_4. При расчете ожидаемого количества

представителей схемы S_4 в новой популяции по формуле (3.10) для вероятности скрещивания $p_c = 1$ и вероятности мутации $p_m = 0$ получаем значение 0,165. В примере 2 после скрещивания в новую популяцию не была включена ни одна хромосома, соответствующая схеме S_4.

Из примеров 4-8, посвященных обработке схем, можно сделать следующие выводы. Эти примеры иллюстрируют основную теорему генетических алгоритмов - теорему о схемах. Они затрагивают обработку схем низкого (малого) порядка с малым охватом. Примеры 4-6 демонстрируют увеличение количества представителей данной схемы в следующем поколении для случая, когда приспособленность этой схемы превышает среднюю приспособленность всех особей популяции. Примеры 7 и 8 показывают ситуацию, когда приспособленность схемы оказывается меньше средней приспособленности особей популяции. Количество представителей таких схем в следующих поколениях не увеличивается, а наоборот - наблюдается уменьшение количества соответствующих им хромосом.

При анализе подобных примеров для схем большего порядка и большего охвата также не регистрируется увеличение количества их представителей в следующем поколении, что согласуется с теоремой о схемах.

Графическая интерпретация схем, обсуждавшихся в примерах 5 и 8, представлена на рис.4; аналогичным образом можно проиллюстрировать схемы из примеров 6 и 7, равно как и любые другие.

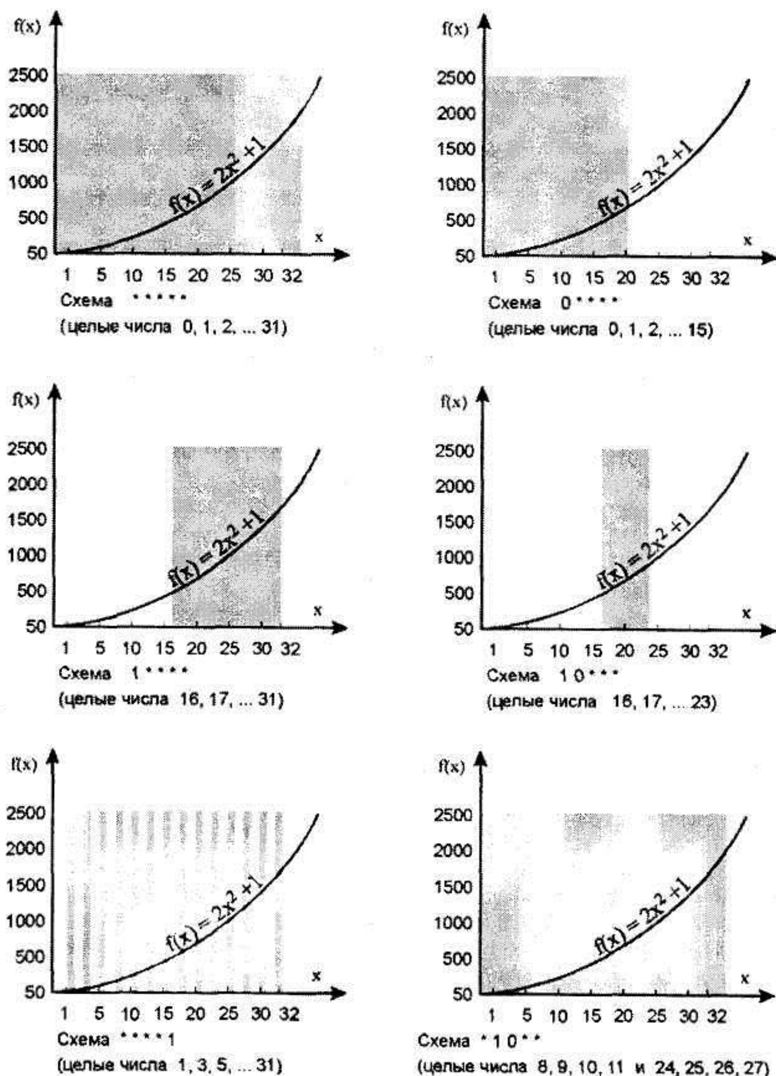


Рис.4. Графическое представление схем для целочисленных значений x от 0 до 31, закодированных в форме 5-битовых двоичных последовательностей для оптимизации функции $f(x) = 2x^2 + 1$ (примеры 1, а также 5 и 8).

На рис. 4 видно, что к схеме 1*** (пример 5) в исходной популяции из примера 2 принадлежат хромосомы ch_1, ch_4 и ch_6 с фенотипами 19, 21, 29 соответственно, а после селекции и скрещивания к этой схеме уже принадлежат все включенные в новую популяцию хромосомы, т.е. Ch_1, Ch_2, Ch_3, Ch_4, Ch_5, и Ch_6 с фенотипами 17, 23, 21, 29, 29, 29 соответственно. В то же время к схеме *10** (пример 8) в исходной популяции из примера 2 принадлежит только одна хромосома ch_5, фенотип которой равен 8; в следующей популяции уже нет ни одной хромосомы, принадлежащей этой схеме. Обратим внимание (рис. 4), что оптимальное решение, которое максимизирует функцию, заданную выражением (3.1), принадлежит к схеме 1**** и не соответствует схеме *10**.

Выполнение генетических алгоритмов основано на обработке схем. Схемы малого порядка, с малым охватом и приспособленностью выше средней выбираются, размножаются и комбинируются, в результате чего формируются все лучшие кодовые последовательности. Поэтому оптимальное решение строится (в соответствии с гипотезой кирпичиков) путем объединения наилучших из полученных к текущему моменту частичных решений. Простое скрещивание не слишком часто уничтожает схемы с малым охватом, однако ликвидирует схемы с достаточно большим охватом. Однако невзирая на губительность операций скрещивания и мутации для схем высокого порядка и охвата, количество обрабатываемых схем настолько велико, что даже при относительно низком количестве хромосом в популяции достигаются весьма неплохие результаты выполнения генетического алгоритма.

Количество эффективно обрабатываемых схем, рассчитанное Холландом , составляет $O(N^3)$. Это означает, что для популяции мощностью N количество обрабатываемых в каждом поколении схем имеет порядок N^3.

3.8. Строительные блоки (Building blocks)

Строительный блок (СБ) (*building block (BB)*) - это одно из ключевых понятий в теории генетических алгоритмов, наряду со схемой и теоремой схем. К строительным блокам (как правило) принято относить схемы с малой определяющей длиной, малым порядком и высокой приспособленностью. Приспособленность СБ чаще всего

определяется как средняя приспособленность особей, хромосомы которых содержат данный строительный блок.

В гипотезе о строительных блоках (*building blocks hypothesis*) считается, что в процессе работы генетического алгоритма, по мере приближения к глобальному оптимуму, порядок и приспособленность строительного блока увеличиваются. Т.е. в начале эволюции относительно высокой приспособленностью обладают строительные блоки малой определяющей длины, затем, в результате отбора и рекомбинации, появляются более приспособленные и "длинные" строительные блоки. Таким образом, говорят об обработке строительных блоков (*building blocks processing*) в результате работы генетического алгоритма.

Выделяют следующие проблемы в обработке строительных блоков:
1. Идентификация.
2. Смешивание.

Под идентификацией понимается проблема нахождения (локализации) строительного блока. Иными словами, какой именно участок хромосомы можно считать строительным блоком. Очевидно, что минимальный по размерам СБ представляет один разряд, а максимальный - всю хромосому. Но это в простейшем варианте, а в случае "не граничных условий" количество "претендентов" на звание строительного блока очень велико. В частности, для хромосомы длины n может существовать $C(n, 2)$ различных позиций для строительных блоков 2-го порядка, $C(n, 3)$ различных позиций для строительных блоков третьего порядка и т.д., где $C(n, m)$ - количество сочетаний из n по m (вычисляется как $n!/(m!(n-m)!)$, где "!" - факториал, т.е. $n! = 1*2*...*(n-1)*n$, где "*" - обозначает операцию умножения). Проблема смешивания заключается в том, что, даже если строительные блоки найдены, трудно определить, какие из них стоит смешивать (грубо говоря, помещать в одну хромосому), а какие – нет, т.к. высокая приспособленность различных строительных блоков не гарантирует высокой приспособленности хромосом, полученных в результате их комбинации. В силу обозначенных проблем уделяется довольно много внимания разработке операторов и подходов, которые позволили бы обеспечить эффективную (*BB-wise*) обработку строительных блоков. Также существует мнение, что эффективная идентификация и

смешивание являются критическими условиями для обеспечения успешной работы генетического алгоритма.

Помимо идентификации и смешивания существует проблема неопределенности приспособленности строительного блока (*building block fitness noise*). Поскольку один и тот же строительный блок может входить как в хорошие (с точки зрения решаемой задачи) хромосомы, так и в не очень, то приспособленность этого строительного блока практически всегда определяется с некоторым "шумом". Это осложняет задачу выбора между двумя строительными блоками, т.к. даже если приспособленность одного из них выше, чем приспособленность другого, всегда есть вероятность сделать неправильный выбор. Говоря по-другому, в результате селекции может быть выбрана особь с менее приспособленным строительным блоком. Графически это можно представить следующим образом (рис.1)

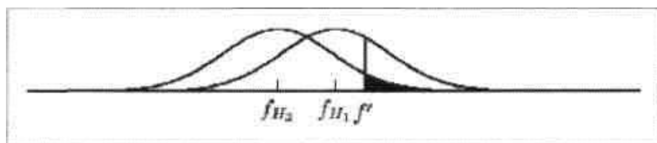


Рис.1.

Пусть *H1* и *H2* два строительных блока, приспособленности которых распределены по нормальному закону, причем f_{H_1} и f_{H_2} соответственно средние приспособленности блоков 1 и 2. Пусть также приспособленность H_1 больше, чем приспособленность H_2. Если имеется небольшое число хромосом, содержащих рассматриваемые блоки, то их приспособленности определены с достаточно большой погрешностью, иными словами, дисперсия распределений велика. Таким образом, площадь взаимного "наложения" распределений также значительна, и, следовательно, значительна вероятность выбрать в результате селекции менее приспособленный строительный блок. В случае, представленном на рис.1, вероятность выбрать хромосому с приспособленностью больше f' и содержащую схему H_2, равна площади закрашенной части распределения приспособленности 2-й схемы.

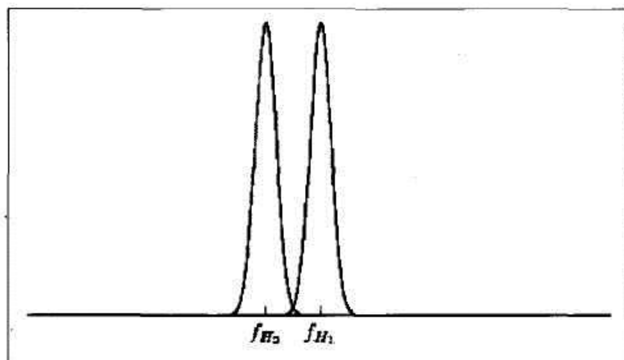


Рис.2.

Если увеличить количество хромосом содержащих рассматриваемые строительные блоки (например, увеличив размер популяции), то тогда распределения приспособленностей обоих блоков "сжимаются", т.к. значения приспособленностей определяются более точно (рис.2). Тем самым уменьшается вероятность выбора менее приспособленного строительного блока.

Несмотря на то, что изначально строительные блоки определялись как схемы с определенными свойствами, т.е. применительно к ГА с бинарным кодированием, аналоги схем, а следовательно и строительных блоков, можно найти и в других видах эволюционных вычислений (не только в генетических алгоритмах). Видимо, концепция строительных блоков давно уже живет "своей жизнью", отдельно от теоремы схем, т.к. за СБ можно считать любой участок хромосомы определенного вида вне зависимости от того, возможно ли описание хромосом, содержащих этот блок, одной схемой или нельзя.

3.9. Модификации классического генетического алгоритма

В классическом генетическом алгоритме используется двоичное представление хромосом, селекция методом колеса рулетки и точечное

скрещивание (с одной точкой скрещивания). Для повышения эффективности его работы создано множество модификаций основного алгоритма. Они связаны с применением других методов селекции, с модификацией генетических операторов (в первую очередь оператора скрещивания), с преобразованием функции приспособленности (путем ее масштабирования), а также с иными способами кодирования параметров задачи в форме хромосом. Существуют также версии генетических алгоритмов, позволяющие находить не только глобальный, но и локальные оптимумы. Это алгоритмы, использующие так называемые ниши, введенные в генетические алгоритмы по аналогии с природными экологическими нишами. Другие версии генетических алгоритмов служат для многокритериальной оптимизации, т.е. для одновременного поиска оптимального решения для нескольких функций. Встречаются также специальные версии генетического алгоритма, созданные для решения проблем малой размерности, не требующих ни больших популяций, ни длинных хромосом. Их называют *генетическими микроалгоритмами.*

3.9.1. Методы селекции

Основанный на принципе колеса рулетки метод селекции считается для генетических алгоритмов основным методом отбора особей для родительской популяции с целью последующего их преобразования генетическими операторами, такими как скрещивание и мутация. Несмотря на случайный характер процедуры селекции, родительские особи выбираются пропорционально значениям их функций приспособленности, т.е. согласно вероятности селекции, определяемой по формуле (3.3). Каждая особь получает в родительском пуле такое количество своих копий, какое устанавливается выражением

$e(ch_i) = p_s(ch_i)\cdot/V$, (3.16)

где N - количество хромосом ch_i, i=1,2.....N в популяции, а $p_s(ch_i)$ - вероятность селекции хромосомы ch_i, рассчитываемая по формуле (3.3). Строго говоря, количество копий данной особи в родительском пуле равно целой части от $e(ch_i)$. При использовании формул (3.3) и (3.16) необходимо обращать внимание на то, что $e(ch_1) = F(ch_i)/\overline{F}$,

где $\overline{F}$ - среднее значение функции приспособленности в популяции. Очевидно, что метод рулетки можно применять тогда, когда значения функции приспособленности положительны. Этот метод может

использоваться только в задачах максимизации функции (но не минимизации).

Очевидно, что проблему минимизации можно легко свести к задаче максимизации функции и обратно. В некоторых реализациях генетического алгоритма метод рулетки применяется для поиска минимума функции (а не максимума). Это результат соответствующего преобразования, выполняемого программным путем для удобства пользователей, поскольку в большинстве прикладных задач решается проблема минимизации (например, затрат, расстояния, погрешности и т.п.). В качестве примера такой реализации можно назвать программу **FlexTool**. Однако возможность применения метода рулетки всего лишь для одного класса задач, т.е. только для максимизации (или только для минимизации) можно считать его несомненным недостатком. Другая слабая сторона этого метода заключается в том, что особи с очень малым значением функции приспособленности слишком быстро исключаются из популяции, что может привести к преждевременной сходимости генетического алгоритма. Для предотвращения такого эффекта применяется масштабирование функции приспособленности (п. 3.9.5).

С учетом отмеченных недостатков метода рулетки созданы и используются альтернативные алгоритмы селекции. Один из них называется *турнирным методом (tournament selection).* Наряду с методом рулетки и ранговым методом он применяется как один из основных алгоритмов селекции в программе **FlexTool.** Представим эти методы подробнее.

При турнирной селекции все особи популяции разбиваются на подгруппы с последующим выбором в каждой из них особи с наилучшей приспособленностью. Различаются два способа такого выбора: *детерминированный выбор (deterministic tournament selection)* и *случайный выбор (stochastic tournament selection).* Детерминированный выбор осуществляется с вероятностью, равной 1, а случайный выбор - с вероятностью, меньшей 1. Подгруппы могут иметь произвольный размер, но чаще всего популяция разделяется на подгруппы по 2 - 3 особи в каждой.

Турнирный метод пригоден для решения задач как максимизации, так и минимизации функции. Помимо того, он может быть легко распространен на задачи, связанные с многокритериальной оптимизацией, т.е. на случай одновременной оптимизации нескольких функций. В турнирном методе допускается изменение размера подгрупп, на которые подразделяется популяция *(tournament size).* Исследования подтверждают, что турнирный метод действует эффективнее, чем метод рулетки.

На рис. 1 представлена схема, которая иллюстрирует метод турнирной селекции для подгрупп, состоящих из двух особей. Такую схему легко обобщить на подгруппы большего размера. Это одно из возможных приложений рассматриваемого алгоритма селекции (оно используется в программе **FlexTool**).

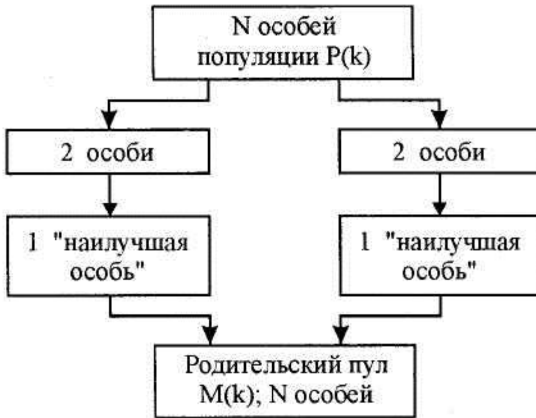


Рис. 1. Схема турнирной селекции для подгрупп, состоящих из двух особей.

При *ранговой селекции (ranking selection)* особи популяции ранжируются по значениям их функции приспособленности. Это можно представить себе как отсортированный список особей, упорядоченных по направлению от наиболее приспособленных к наименее приспособленным (или наоборот), в котором каждой особи приписывается число, определяющее ее место в списке и называемое *рангом (rank).* Количество копий *M(k)* каждой особи, введенных в родительскую популяцию, рассчитывается по априорно заданной функции в зависимости от ранга особи. Пример такой функции показан на рис. 2.

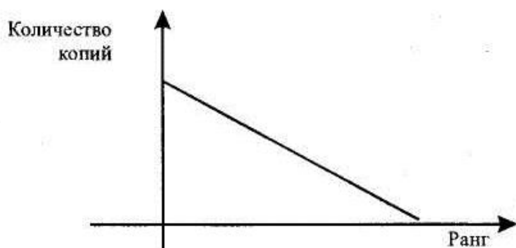


Рис. 2. Пример функции, определяющей зависимость количества копий особи в родительском пуле от его ранга при ранговой селекции.

Достоинство рангового метода заключается в возможности его применения как для максимизации, так и для минимизации функции. Он также не требует масштабирования из-за проблемы преждевременной сходимости, актуальной для метода рулетки. Существуют различные варианты алгоритмов селекции. Представленные ранее методы (рулетки, турнирный и ранговый) применяются чаще всего. Другие методы представляют собой либо их модификации, либо комбинации - например, метода рулетки с турнирным методом, когда пары родительских хромосом выбираются случайным образом, после чего из каждой пары выбирается хромосома с наибольшим значением функции приспособленности. Большинство методов селекции основано на формулах (3.3) и (3.16), по которым рассчитывается вероятность селекции и количество копий, вводимых в родительский пул. В так называемом детерминированном методе каждая особь получает число копий, равное целой части от $e(ch_i)$, после чего популяция упорядочивается в соответствии с дробной частью $e(ch_i)$, а остальные хромосомы, необходимые для пополнения новой популяции, последовательно выбираются из верхней части сформированного таким образом списка. В другом методе (называемом случайным) дробные части $e(ch_i)$ рассматриваются как вероятности успеха по Бернулли и, например, хромосома ch_i, для которой $e(ch_i) = 1,5$, получает одну копию гарантированно и еще одну - с вероятностью 0,5. В еще одном методе для устранения расхождения между расчетным значением $e(ch_i)$ и количеством копий хромосом ch_i, выбираемым по методу рулетки, производится модификация $e(ch_i)$ путем увеличения или уменьшения его значения для каждой хромосомы, выбранной для скрещивания и/или мутации.

3.9.2. Особые процедуры репродукции

В качестве особых процедур репродукции можно рассматривать так называемую *элитарную стратегию* и *генетический алгоритм с частичной заменой популяции.*

Элитарная стратегия (elitist strategy) заключается в защите наилучших хромосом на последующих итерациях. В классическом генетическом алгоритме самые приспособленные особи не всегда переходят в следующее поколение. Это означает, что новая популяция $P(k+1)$ не всегда содержит хромосому с наибольшим значением функции приспособленности из популяции $P(k)$. Элитарная стратегия применяется для предотвращения потери такой особи. Эта особь гарантированно включается в новую популяцию.

Генетический алгоритм с частичной заменой популяции, иначе называемый *генетическим алгоритмом с зафиксированным состоянием (steady-state),* характеризуется тем, что часть популяции переходит в следующее поколение без каких-либо изменений. Это означает, что входящие в эту часть хромосомы не подвергаются операциям скрещивания и мутации. Часто в конкретных реализациях алгоритма данного типа на каждой итерации заменяются только одна или две особи вместо скрещивания и мутации в масштабе всей популяции. Именно такой подход принят, например, в программе **Evolver** . В других программах, в частности, во **FlexTool** , пользователь может сам установить - какая часть популяции (в соответствии со значениями функции приспособленности) должна передаваться без изменений в следующее поколение. Это подмножество хромосом не подвергается регулярной селекции и без изменений включается в новую популяцию.

3.9.3. Генетические операторы

В классическом генетическом алгоритме операция скрещивания представляет собой так называемое точечное скрещивание. Также применяются и другие виды скрещивания: двухточечное, многоточечное и равномерное.

Двухточечное скрещивание (two-point crossover), как следует из его названия, отличается от точечного скрещивания тем, что потомки наследуют фрагменты родительских хромосом, определяемые двумя случайно выбранными точками скрещивания. Для пары хромосом из примера 1 скрещивание в точках 4 и 6 показано на рис.1. Обратим

внимание, что такое скрещивание не приводит к уничтожению схемы 1**********1, которую представляет родитель 2.

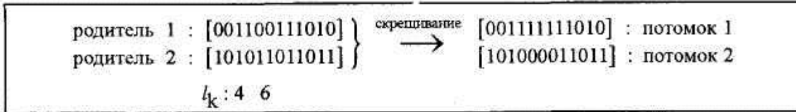


Рис.1. Пример двухточечного скрещивания.

Многоточечное скрещивание (multiple-point crossover) представляет собой обобщение предыдущих операций и характеризуется соответственно большим количеством точек скрещивания. Например, для трех точек скрещивания, равных 4, 6 и 9, и такого же количества родителей, как на рис.1, результаты скрещивания показаны на рис.2.

Рис.2. Пример трехточечного скрещивания.

Аналогично производится скрещивание для пяти или большего нечетного количества точек. Очевидно, что одноточечное скрещивание может считаться частным случаем многоточечного скрещивания.
Пример двухточечного скрещивания, представленный на рис.1, можно проиллюстрировать способом, показанным на рис. 3.

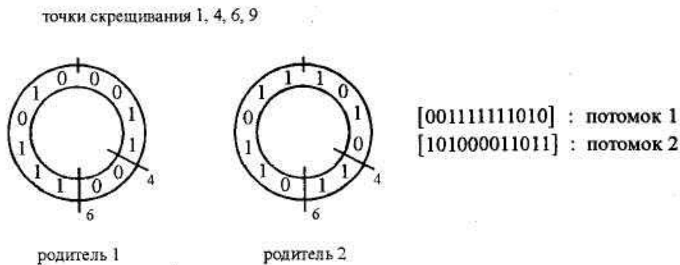


Рис. 3. Двухточечное скрещивание с точками скрещивания 4 и 6.

Многоточечное скрещивание для четырех точек, равных 1,4,6, 9 и 4, 6, 9, 11 для той же пары родителей из предыдущих примеров иллюстрируется на рис.4.

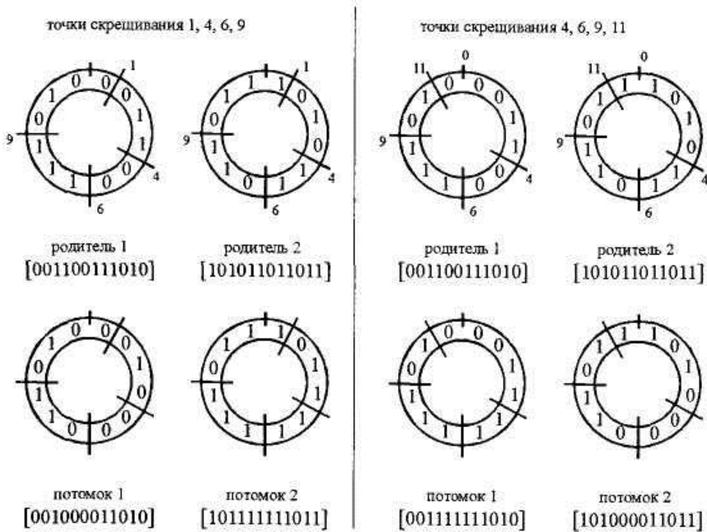


Рис.4. Многоточечное скрещивание с четырьмя точками скрещивания, равными 1, 4, 6, 9 и 4, 6, 9, 11.

Многоточечное скрещивание с большим четным количеством точек скрещивания протекает аналогично показанному на рис.4.

Скрещивание с нечетным количеством точек можно представить таким же образом, если добавить еще одну точку скрещивания в позиции, равной 0. Приведенный выше пример для трех точек можно представить также, как на рис.4, с точками скрещивания 0, 4, 6, 9. При четном количестве точек хромосома рассматривается как замкнутое кольцо (см. рис. 3 и 4), а точки скрещивания выбираются с равной вероятностью по всей его окружности.

Равномерное скрещивание (uniform crossover), иначе называемое *монолитным* или *одностадийным,* выполняется в соответствии со случайно выбранным эталоном, который указывает, какие гены должны наследоваться от первого родителя (остальные гены берутся от

второго родителя). Допустим, что для пары родителей из примеров на рис. 1- 4 выбран эталон 010110111011, в котором 1 означает принятие гена на соответствующей позиции *(locus)* от родителя 1, а 0 - от родителя 2. Таким образом формируется первый потомок. Для второго потомка эталон необходимо считывать аналогично, причем 1 означает принятие гена на соответствующей позиции от родителя 2, а 0 - от родителя 1. В этом случае равномерное скрещивание протекает так, как показано на рис.5.

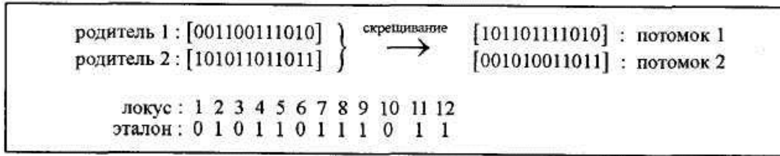


Рис.5. Пример равномерного скрещивания.

Оператор инверсии. Холланд предложил три технологии для получения потомков, отличающихся от родительских хромосом. Это уже известные нам операции скрещивания и мутации, а также операция инверсии. Инверсия выполняется на одиночной хромосоме; при ее осуществлении изменяется последовательность аллелей между двумя случайно выбираемыми позициями *(locus)* в хромосоме. Несмотря на то, что этот оператор был определен по аналогии с биологическим процессом хромосомной инверсии, он не слишком часто применяется в генетических алгоритмах. В качестве примера выполнения инверсии рассмотрим хромосому [001100111010] и допустим, что выбраны позиции 4 и 10. Тогда в результате инверсии получим [001101110010].

3.9.4. Методы кодирования

В классическом генетическом алгоритме применяется двоичное кодирование хромосом. Оно основано на известном способе записи десятичных чисел в двоичной системе, где каждый бит двоичного кода соответствует очередной степени цифры 2. Например, двоичная последовательность [10011] представляет собой код числа 19, поскольку $1*2^4+0*2^3 + 0*2^2 + 1*2^1 + 1*2^0 = 19$. Такой способ кодирования применялся в примере 2. Для кодирования действительных чисел

$x_i \in [a_i, b_i] \in R$ реализуется отображение (3.5) так, как это делалось в примере 3.

В генетических алгоритмах можно, например, использовать код Грея, который характеризуется тем, что двоичные последовательноети, соответствующие двум последовательным целым числам, отличаются только одним битом. Такой способ кодирования хромосом может оказаться оправданным при использовании операции мутации.

Логарифмическое кодирование (logarithmic coding) применяется в генетических алгоритмах для уменьшения длины хромосом. Оно используется, главным образом, в задачах многомерной оптимизации с большими пространствами поиска решений.

При логарифмическом кодировании первый бит (α) кодовой последовательности - это бит знака показательной функции, второй бит *(β)* - бит знака степени этой функции, а остальные биты (bin) представляют значение самой степени:

$[\alpha\beta \text{ bin}] = (-1)^{\beta} e^{(-1)^{\alpha} [bin]_{10}}$

где $[\text{bin}]_{10}$ означает десятичное значение числа, закодированного в виде двоичной последовательности bin. Например,

[10110]

представляет собой кодовую последовательность числа

$x_1 = (-1)^0 e^{(-1)^1 [110]_{10}} = e^{-6} = 0{,}002478752$,

а [01010] представляет собой кодовую последовательность числа

$x_2 = = (-1)^1 e^{(-1)^0 [010]_{10}} = -e^2 = -7{,}389056099$.

Заметим, что таким образом с помощью пяти битов можно закодировать числа из интервала $[-e^7, e^7]$. Это значительно больший интервал, чем [0, 31] из примера 2. Логарифмическое кодирование было реализовано в программе **FlexTool** в качестве дополнительной опции для задач повышенной сложности.

Еще одна модификация классического генетического алгоритма основана на кодировании действительными, а не двоичными числами. Это означает, что гены хромосом принимают действительные значения (аллели являются действительными числами). Такой способ кодирования применяется, в частности, в программе **Evolver**.

3.9.5. Масштабирование функции приспособленности

Масштабирование функции приспособленности выполняется, чаще всего, по двум причинам. Во-первых (об этом уже говорилось при

обсуждении методов селекции), для предотвращения преждевременной сходимости генетического алгоритма. Во-вторых (в конечной фазе выполнения алгоритма), в случае, когда в популяции сохраняется значительная неоднородность, однако среднее значение приспособленности ненамного отличается от максимального значения. Масштабирование функции приспособленности позволяет предупредить возникновение ситуации, в которой средние и наилучшие особи формируют практически одинаковое количество потомков в следующих поколениях, что считается нежелательным явлением. Преждевременная сходимость алгоритма заключается в том, что в популяции начинают доминировать наилучшие, но еще не оптимальные хромосомы. Такая возможность характерна для алгоритмов с селекцией по методу колеса рулетки. Через несколько поколений при селекции, пропорциональной значению функции приспособленности, популяция будет состоять исключительно из копий наилучшей хромосомы исходной популяции. Представляется маловероятным, что именно эта хромосома будет соответствовать оптимальному решению, поскольку исходная популяция - это, как правило, небольшая случайная выборка из всего пространства поиска. Масштабирование функции приспособленности предохраняет популяцию от доминирования неоптимальной хромосомы и тем самым предотвращает преждевременную сходимость генетического алгоритма.

Масштабирование заключается в соответствующем преобразовании функции приспособленности. Различают 3 основных метода масштабирования: линейное, сигма-отсечение и степенное.

Линейное масштабирование (linear scaling) заключается в преобразовании функции приспособленности F к форме F’ через линейную зависимость вида

$$F' = a \cdot F + b,$$

где a и b - константы, которые следует подбирать таким образом, чтобы среднее значение функции приспособленности после масштабирования было равно ее среднему значению до масштабирования, а максимальное значение функции приспособленности после масштабирования было кратным ее среднему значению. Коэффициент кратности чаще всего выбирается в пределах от 1,2 до 2. Необходимо следить за тем, чтобы функция F’ не принимала отрицательные значения.

Сигма-отсечение (sigma truncation) - метод масштабирования, основанный на преобразовании функции приспособленности *F* к форме F' согласно выражению

$F' = F + (\vec{F} - c \cdot \sigma),$

где $\vec{F}$ обозначает среднее значение функции приспособленности по всей популяции, *c* - малое натуральное число (как правило, от 1 до 5), а *σ* - стандартное отклонение по популяции. Если расчетные значения F' отрицательны, то они принимаются равными нулю.

Степенное масштабирование (power law scaling) представляет собой метод масштабирования, при котором функция приспособленности *F* преобразуется к форме F' согласно выражению

$F' = F^k,$

где *k* - число, близкое 1. Значение *k* обычно подбирается эмпирически с учетом специфики решаемой задачи. Например, можно использовать $k = 1{,}005$.

3.9.6. Ниши в генетическом алгоритме

В различных оптимизационных задачах часто приходится иметь дело с функциями, имеющими несколько оптимальных решений. Основной генетический алгоритм в таких случаях находит только глобальный оптимум, но если имеется несколько оптимумов с одним и тем же значением, то он отыскивает только один из них. В некоторых задачах бывает важным найти не только глобальный оптимум, но и локальные оптимумы (не обязательно все). Концепция реализации в генетических алгоритмах подхода, основанного на известных из биологии понятиях ниш и видов, позволяет находить большую часть оптимумов. Практически применяемый в генетическом алгоритме метод образования ниш и видов основан на так называемой *функции соучастия (sharing function).* Эта функция определяет уровень близости и степень соучастия для каждой хромосомы в популяции. Функция соучастия обозначается $s(d_{ij})$, где d_{ij} - мера расстояния между хромосомами ch_i и ch_j. В программе **FlexTool** это расстояние определяется по формуле

$$d_{ij} = \sqrt{\sum_{k=1}^{p} \frac{x_{k,i} - x_{k,j}}{(x_{k\,max} - x_{k,min})^2}}$$

где p означает размерность задачи, $x_{k,min}$ и $x_{k,max}$ определяют соответственно минимальное и максимальное значение k-го параметра, x_{ki} и x_{kj} - обозначают соответственно *k-й* параметр i-й и j-й особей. Очевидно, что расстояние между хромосомами рассчитывается на основе соответствующих им фенотипов.

Функция соучастия $s(d_{ij})$ должна обладать следующими свойствами:

$0 \le s(d_{ij}) \le 1$ для каждого d_{ij},

$s(0) = 1$,

$$\lim_{d_{ij} \to \infty} s(d_{ij}) = 0$$

Одна из функций, для которой эти условия выполняются, имеет вид

$$s(d_{ij}) = \begin{cases} 1 - (d_{ij} / \sigma_s)^{\omega}, & \text{если } d_{ij} > \sigma_s, \\ 0 & \text{в противном случае} \end{cases}$$

где σ_s и ω - константы.

В программе **FlexTool** $\sigma_s = 0{,}5 * q^{-1/p}$, где q обозначает задаваемое пользователем примерное количество пиков оптимизируемой функции. Значение ω принимается равным 1, что означает одинаковую степень соучастия соседних особей. В этом случае новое значение функции приспособленности хромосомы ch, рассчитывается по формуле

$$F_s(ch_i) = \frac{F(ch_i)}{\sum_{i=1}^{N} s(d_{ij})} \quad (3.17)$$

где N обозначает количество хромосом в популяции.

Если хромосома ch_i находится в своей нише в одиночестве, то $F_s(ch_i) = F(ch_i)$. В противном случае значение функции приспособленности уменьшается пропорционально количеству и степени близости соседствующих хромосом. Из выражения (3.17) следует, что увеличение количества похожих друг на друга (т.е. принадлежащих к одной и той же нише) хромосом ограничено, поскольку такое увеличение приводит к уменьшению значения функции приспособленности. В программе **FlexTool** при реализации генетического алгоритма с нишами представляемый метод используется на завершающем этапе обработки каждого поколения.

Имеются также и различные модификации процедуры образования ниш для генетического алгоритма. Например, можно определить меру расстояния между хромосомами не на уровне фенотипа (т.е. параметров задачи), а на уровне генотипа. В этом случае аргументом

функции соучастия будет расстояние Хемминга между кодовыми последовательностями. Известны и другие подходы к модификации функции приспособленности для генетического алгоритма с нишами.

3.9.7. Генетические алгоритмы для многокритериальной оптимизации

Большинство задач, решаемых при помощи генетических алгоритмов, имеют один критерий оптимизации. В свою очередь, многокритериальная оптимизация основана на отыскании решения, одновременно оптимизирующего более чем одну функцию. В этом случае ищется некоторый компромисс, в роли которого выступает решение, оптимальное в смысле Парето. При многокритериальной оптимизации выбирается не единственная хромосома, представляющая собой закодированную форму оптимального решения в обычном смысле, а множество хромосом, оптимальных в смысле Парето. Пользователь имеет возможность выбрать оптимальное решение из этого множества. Рассмотрим определение решения, оптимального в смысле Парето (символами x, y будем обозначать фенотипы).

Определение 3.3

Решение x называется *доминируемым,* если существует решение y, не хуже чем x, т.е. для любой оптимизируемой функции $f_i, j=1,2, \ldots, m$,

$f_i(x) \leq f_i(y)$ при максимизации функции f_i,

$f_i(x) \geq f_i(y)$ при минимизации функции f_i.

Определение 3.4

Если решение не доминируемо никаким другим решением, то оно называется *недоминируемым* или *оптимальным в смысле Парето.*

Существует несколько классических методов, относящихся к многокритериальной оптимизации. Один из них - это метод *взвешенной функции (method of objective weighting),* в соответствии с которым оптимизируемые функции f_i, с весами w_i образуют единую функцию

$$f(x) = \sum_{i=1}^{m} w_i f_i(x),$$

где $w_i \in [0,1]$ и $\sum_{i=1}^{m} w_i = 1$

Различные веса дают различные решения в смысле Парето.

Другой подход известен как *метод функции расстояния (method of distance function).* Идея этого метода заключается в сравнении значений $f_i(x)$ с заданным значением y_i, т.е.

$$f(x) = \left(\sum_{i=1}^{m} / f_i(x) - y_i /^r \right)^{\frac{1}{r}}$$

При этом, как правило, принимается $r = 2$. Это метрика Эвклида.

Еще один подход к многокритериальной оптимизации связан с разделением популяции на подгруппы одинакового размера *(sub-populations),* каждая из которых «отвечает» за одну оптимизируемую функцию. Селекция производится автономно для каждой функции, однако операция скрещивания выполняется без учета границ подгрупп. Алгоритм многокритериальной оптимизации реализован в программе **FlexTool**. Селекция выполняется турнирным методом, при этом «лучшая» особь в каждой подгруппе выбирается на основе функции приспособленности, уникальной для данной подгруппы. Схема такой селекции в случае оптимизации двух функций представлена на рис.1; на этом рисунке F_1 и F_2 обозначают две различные функции приспособленности. Эта схема аналогична схеме, изображенной на рис. 3.26, с той разницей, что на более ранней схеме все подгруппы оценивались по одной и той же функции приспособленности. «Наилучшая» особь из каждой подгруппы смешивается с другими особями, и все генетические операции выполняются так же, как в генетическом алгоритме для оптимизации одной функции. Схему на рис. 1 можно легко обобщить на большее количество оптимизируемых функций. Программа **FlexTool** обеспечивает одновременную оптимизацию четырех функций.

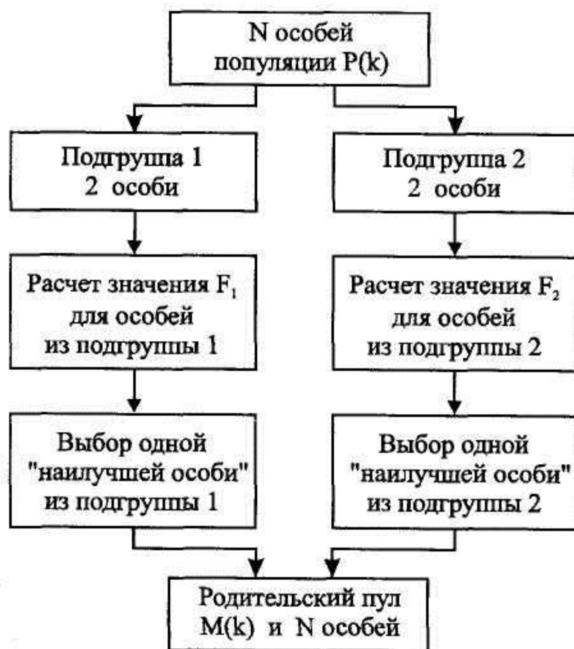


Рис.1. Схема турнирной селекции в случае многокритериальной оптимизации по двум функциям.

3.9.8. Генетические микроалгоритмы

Генетический микроалгоритм - это модификация классического генетического алгоритма, предназначенная для решения задач, не требующих больших популяций и длинных хромосом. Такие алгоритмы применяются при ограниченном времени вычислений в случае, когда решение (не обязательно глобальное) необходимо найти быстро. Речь идет о том, чтобы не производить трудоемких вычислений, связанных с большим количеством итераций. Генетические микроалгоритмы обычно находят несколько худшие решения, однако экономят вычислительные ресурсы компьютера. В качестве примера можно привести генетический микроалгоритм программы **FlexTool.** Он подразделяется на шесть шагов.

1. Сформировать популяцию с числом особей, равным пяти. Можно либо случайным образом выбрать все пять хромосом, либо сохранить одну «хорошую» хромосому, полученную на предыдущих итерациях, и случайным образом «добрать» четыре остальные хромосомы.

2. Рассчитать значения функции приспособленности хромосом в популяции и выбрать лучшую хромосому. Обозначить ее номером 5 и перенести в следующее поколение (элитарная стратегия).
3. Выбрать для репродукции остальные четыре хромосомы на основе детерминированного метода турнирной селекции (наилучшая хромосома также участвует в соревновании за право включения ее копии в родительский пул). В ходе турнирной селекции хромосомы группируются случайным образом, при этом соседствующие пары соперничают за оставшиеся четыре места. Следует обращать внимание на то, чтобы родительская пара не составлялась из двух копий одной и той же хромосомы.
4. Выполнить скрещивание с вероятностью 1; вероятность мутации принять равной 0.
5. Проверить сходимость алгоритма (с использованием соответствующей меры сходимости генотипов или фенотипов). В случае обнаружения сходимости вернуться к шагу 1.
6. Перейти к шагу 2.
Заметим, что в генетическом микроалгоритме размер популяции предполагается небольшим и фиксированным. Применяется элитарная стратегия, которая предотвращает потерю «хороших» хромосом. Поскольку размер популяции невелик, то выполняется детерминированная селекция. Скрещивание проводится с вероятностью 1. Мутация не требуется, так как достаточное разнообразие обеспечивается формированием новой популяции при каждом «рестарте» алгоритма, т.е. в случае перехода к шагу 1 при обнаружении сходимости. Процедура «старта» и «рестарта» алгоритма предназначена для предотвращения преждевременной сходимости; генетический микроалгоритм всегда ищет наилучшие решения. Главная цель его применения заключается в скорейшем нахождении оптимального (или почти оптимального) решения.

4. Модели генетических алгоритмов

4.1. Модели генетических алгоритмов и стратегии отбара и формирования нового поколения

Ниже приведено небольшое описание некоторых моделей генетических алгоритмов. При реализации собственного алгоритма придерживаться этих моделей не обязательно, т.к. очень многое зависит от решаемой задачи, однако, описываемые принципы (например, параллелизм) могут оказаться полезными.

1. Canonical GA (J. Holland)

Данная модель алгоритма является классической. Она была предложена Джоном Холландом в его работе "Адаптация в природных и исусственных средах" (1975). Часто можно встретить описание *простого ГА* (Simple GA, D. Goldberg), он отличается от канонического тем, что использует либо рулеточный, либо турнирный отбор. Модель канонического ГА имеет следующие характеристики:

- Фиксированный размер популяции.
- Фиксированная разрядность генов.
- Пропорциональный отбор.
- Особи для скрещивания выбираются случайным образом.
- Одноточечный кроссовер и одноточечная мутация.
- Следующее поколение формируется из потомков текущего поколения без "элитизма". Потомки занимают места своих родителей.

2. Genitor (D. Whitley)

В данной модели используется специфичная стратегия отбора. Вначале, как и полагается, популяция инициализируется и её особи оцениваются. Затем выбираются случайным образом две особи, скрещиваются, причем получается только один потомок, который оценивается и занимает место наименее приспособленной особи. После этого снова случайным образом выбираются 2 особи, и их

потомок занимает место особи с самой низкой приспособленностью. Таким образом на каждом шаге в популяции обновляется только одна особь. Подводя итоги можно выделить следующие характерные особенности:

- Фиксированный размер популяции.
- Фиксированная разрядность генов.
- Особи для скрещивания выбираются случайным образом.
- Ограничений на тип кроссовера и мутации нет.
- В результате скрещивания особей получается один потомок, который занимает место наименее приспособленной особи.

3. Hybrid algorithm (L. "Dave" Davis)

Использование гибридного алгоритма позволяет объединить преимущества ГА с преимуществами классических методов. Дело в том, что ГА являются робастными алгоритмами, т.е. они позволяют находить хорошее решение, но нахождение оптимального решения зачастую оказывается намного более трудной задачей в силу стохастичности принципов работы алгоритма. Поэтому возникла идея использовать ГА на начальном этапе для эффективного сужения пространства поиска вокруг глобального экстремума, а затем взяв лучшую особь, применить один из "классических" методов оптимизации. Характеристики алгоритма:

- Фиксированный размер популяции.
- Фиксированная разрядность генов.
- Любые комбинации стратегий отбора и формирования следующего поколения
- Ограничений на тип кроссовера и мутации нет.
- ГА применяется на начальном этапе, а затем в работу включается классический метод оптимизации.

4. Island Model GA

Представим себе следующую ситуацию. В некотором океане есть группа близкорасположенных островов, на которых живут популяции особей одного вида. Эти популяции развиваются независимо и только

изредка происходит обмен представителями между популяциями. Островная модель ГА использует описанный принцип для поиска решения. Вариант, безусловно, интересный и является одной из разновидностей параллельных ГА. Данная модель генетического алгоритма обладает следующими свойствами:

- Наличие нескольких популяций, как правило одинакового фиксированного размера.
- Фиксированная разрядность генов.
- Любые комбинации стратегий отбора и формирования следующего поколения в каждой популяции. Можно сделать так, что в разных популяциях будут использоваться разные комбинации стратегий, хотя даже один вариант дает разнообразные решения на различных "островах".
- Ограничений на тип кроссовера и мутации нет.
- Случайный обмен особями между "островами". Если миграция будет слишком активной, то особенности островной модели будут сглажены и она будет не очень сильно отличаться от моделей ГА без параллелизма.

5. CHC (Eshelman)

CHC расшифровывается как **C**ross-population selection, Heterogenous recombination and **C**ataclysmic mutation. Данный алгоритм довольно быстро сходится из-за того, что в нем нет мутаций, используются популяции небольшого размера, и отбор особей в следующее поколение ведется и между родительскими особями, и между их потомками. В силу этого после нахождения некоторого решения алгоритм перезапускается, причем лучшая особь копируется в новую популяцию, а оставшиеся особи являются сильной мутацией (мутирует примерно треть битов в хромосоме) существующих и поиск повторяется. Ещу одной специфичной чертой является стратегия скрещивания: все особи разбиваются на пары, причем скрещиваются только те пары, в которых хромосомы особей существенно различны (хэммингово расстояние больше некоторого порогового плюс возможны ограничения на минимальное расстояние между крайними различающимися битами). При скрещивании используется так называемый HUX-оператор (Half Uniform Crossover) - это разновидность однородного кроссовера, но в нем к каждому потомку

попадает ровно половина битов хромосомы от каждого родителя. Таким образом, модель обладает следующими свойствами:

- Фиксированный размер популяции.
- Фиксированная разрядность генов.
- Перезапуск алгоритма после нахождения решения.
- Небольшая популяция.
- Особи для скрещивания разбиваются на пары и скрещиваются при условии существенных отличий.
- Отбор в следующее поколение проводится между родительскими особями и потомками.
- Используется половинный однородный кроссовер (HUX).
- Макромутация при перезапуске.

Стратегии отбора (selection strategies)

ГА представляет из себя итерационный процесс, в котором особи сначала отбираются для скрещивания, потом скрещиваются, затем из их потомков формируется новое поколение и все начинается сначала. Стратегии отбора являются составной частью ГА и определяют "достойных" для скрещивания особей. Ниже рассматриваются несколько наиболее распространенных стратегий.

Пропорциональный отбор (Proportional selection)

При данном виде отбора сначала подсчитывается присособленность каждой особи f_i. После этого находят среднюю приспособленность в популяции $f_{ср}$ как среднее арифметическое значений приспособленности всех особей. Затем для каждой особи вычисляется отношение $f_i/f_{ср}$. Если это отношение больше 1, то особь считается хорошо приспособленной и допускается к скрещиванию, в противном случае особь, скорее всего, останется "за бортом". Например, если дробь равна 2.36, то данная особь имеет двойной шанс на скрещивание и будет иметь вероятность равную 0.36 третьего скрещивания. Если же приспособленность равна 0.54, то особь примет участие в единственном скрещивании с вероятностью 0.54.

Реализовать это можно, например, следующим образом. Пусть имеется массив двоичных строк (популяция) и дополнительный массив для особей допущенных к скрещиванию. Определим для каждой особи популяции значение описанного выше отношения. Далее, будем записывать строки в промежуточный массив согласно следующему правилу: возьмем целую часть понятно-какого-отношения и ровно столько раз запишем данную строку во вспомогательный массив, после этого с помощью случайной величины (СВ) определим, будем ли мы записывать данную строку ещё раз: если СВ больше дробной части отношения, то "да", если нет, то "нет". В дальнейшем, особи для скрещивания выбираются только из промежуточного массива случайным образом, так что, если строка присутствует в нем в нескольких экземплярах, то у нее больше шансов оставить след в истории. Для уже рассмотренных примеров с числами 2.36 и 0.51 ситуация будет выглядеть следующим образом: первая строка запишется в промежуточный массив два раза и с вероятностью 0.36 запишется в третьй раз. Вторая же строка имеет вероятность равную 0.51 того, что она вообще будет присутствовать во вспомогательном массиве.

Данная стратегия отбора знаменита тем, что именно для неё создана классическая теорема схем (вот откуда там дробь).

Турнирный отбор (Tournament selection)

Турнирный отбор может быть описан следующим образом: из популяции, содержащей N строк, выбирается случайным образом t строк и лучшая строка записывается в промежуточный массив (между выбранными строками проводится турнир). Эта операция повторяется N раз. Строки в полученном промежуточном массиве затем используются для скрещивания (также случайным образом). Размер группы строк, отбираемых для турнира часто равен 2. В этом случае говорят о *двоичном/парном турнире (binary tournament)*. Вообще же t называется *численностью турнира (tournament size).* Чем больше турнир, тем более жесткий вариант селекции, т.е. тем меньше шансов у особей попасть в отбор.

Преимуществом данной стратегии является то, что она не требует дополнительных вычислений и упорядочивания строк в популяции по

возрастанию приспособленности. Также, видимо, такой вариант селекции ближе к реальности, т.к. успешность той или иной особи во многом определяется ее окружением, насколько оно лучше или хуже ее

- ***Отбор усечением (Truncation selection)***

Данная стратегия использует отсортированную по возрастанию популяцию. Число особей для скрещивания выбирается в соответствии с *порогом* T[0; 1]. Порог определяет какая доля особей, начиная с самой первой (=самой приспособленной) будет принимать участие в отборе. В принципе, порог можно задать и числом больше 1, тогда он будет просто равен числу особей из текущей популяции, допущенных к отбору. Среди особей, попавших "под порог" случайным образом N раз выбирается самая везучая и записывается в промежуточный массив, из которого затем выбираются особи непосредственно для скрещивания.

Из-за того, что в этой стратегии используется отсортированная популяция, время её работы может быть большим для популяций большого размера и зависеть также от алгоритма сортировки.

Существуют и другие стратегии отбора, например, с линейным и экспоненциальным ранжированием.

Стратегии формирования нового поколения

После скрещивания особей необходимо решить проблему о том какие из новых особей войдут в следующее поколение, а какие - нет, и что делать с их предками. Есть два способа:

1. Новые особи (потомки) занимают места своих родителей. После чего наступает следующий этап, в котором потомки оцениваются, отбираются, дают потомство и уступают место своим "детям".
2. Создается промежуточная популяция, которая включает в себя как родителей, так и их потомков. Члены этой популяции оцениваются, а затем из них выбираются N самых лучших, которые и войдут в следующее поколение.

4.2. Проверка эффективности ГА с использованием тестовых функций

После того как написан свой собственный ГА вам хочется знать насколько он хорош. Ниже приведены некоторые тестовые функции для ГА. Все тестовые функции могут иметь различное число параметров (n). Поэтому имеет смысл запустить алгоритм для оптимизации некоторой функции сначала с небольшим n (например, 10 или 20), а затем с n=50, 100, 200,... Это даст возможность проверить масштабируемость алгоритма.

Эффективность работы ГА принято оценивать количеством вычислений целевой функции. Чем меньше, тем лучше. После некоторых функций приведены результаты работы QGA генетического олгаритма. Результаты целевой функции меньше 0.001 тоже засчитывались как найденный глобальный минимум. Вот характеристики QGA:

- Фиксированный размер популяции;
- Фиксированная разрядность генов;
- Количество точек разрыва кроссовера равно числу генов (на каждый ген приходится ровно одна точка);
- Для скрещивания отбираются 50% популяции;
- Вероятность мутации 95%;
- Две "элитные" особи;
- Удаление одинаковых особей из популяции (с помощью мутации);
- Точность вычислений: 0.001;
- Результат подсчитан по 50 запускам алгоритма;

Т.к. генетические алгоритмы используют стохастичность, то для того, чтобы определить, насколько эффективен ваш ГА нужно запустить его на одной и той же тестовой функции несколько раз и только после этого анализировать результат. Например, QGA ГА для функции Растригина от 50 переменных находит глобальный минимум в ~40% случаев, используя не более 10000 вычислений функции.

Для некоторых функций есть графики для случая двух переменных. Графики представляют "сечение" поверхности функции плоскостью, проходящей через глобальный минимум, перпендикулярной одной из осей координат.

Ниже приводится набор тестовых функций.

- ***Sphere model.***

Считается легкой функцией для любого метода оптимизации.

$$f(x) = \sum_{i=1}^{n} x_i^2$$

xO(-5,12; 5,12)
Один минимум равный 0 в точке, где x_i=0.0.

Результат QGA:
n=10, количество нахождений глобального минимума (он здесь один) 86%, число вычислений целевой функции не более 1250, максимальное значение 0,008325.
n=30, количество нахождений глобального минимума 34%, число вычислений целевой функции не более 1250, максимальное значение 0,108851.
n=50, количество нахождений глобального минимума 46%, число вычислений целевой функции не более 2500, максимальное значение 0,016291, для скрещивания отбиралось 40% популяции.

- ***Rosenbrock's saddle***

$$f(x) = \sum_{i=1}^{n-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$$

xO(-2,048; 2,048)
Минимум равный 0 в точке, где x_i=1.0. Функция имеет большое медленно убывающее плато.

Результат QGA:
n=2, количество нахождений глобального минимума 92%, число вычислений целевой функции не более 1250, максимальное значение функции 0,00169118.
n=4, количество нахождений глобального минимума 86%, число вычислений целевой функции не более 1250, максимальное значение функции 0,00504982.
n=10, количество нахождений глобального минимума 0%, число вычислений целевой функции не более 10000, максимальное значение функции 0,0186537, минимальное значение функции 0,0019092, для скрещивания отбиралось 50% популяции.
n=10, количество нахождений глобального минимума 8%, число вычислений целевой функции не более 10000, максимальное значение функции 0,0127758, для скрещивания отбиралось 40% популяции.

- ***Step function***

$$f(x) = \sum_{i=1}^{n} |[x_i]|$$

xO(-5,12; 5,12)
$|[x_i]|$ - модуль целой части
Минимум равный 0 в точке, где x_i=0.0.

- ***Gaussian quartic***

$$f(x) = \sum_{i=1}^{n} (x_i^4 + gauss(0,1))$$

xO(-1.28; 1.28)
gauss(0,1) - функция, возвращающая случайную величину с нормальным распределением с математическим ожиданием в 0 и дисперсией равной 1.
Минимум равный 0 в точке, где x_i=0.0.

Rastrigin's function

Функция со сложным рельефом. Считается сложной для оптимизации.

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

x*O*(-5,12; 5,12)
Минимум равный 0 в точке, где x_i=0.0.
Локальный минимум в точке, где одна координата равна 1.0, а остальные равны 0.0

Результат QGA:
n=20, количество нахождений глобального минимума 16%, число вычислений целевой функции не более 5000, максимальное значение функции 19,8992.
n=20, количество нахождений глобального минимума 26%, число вычислений целевой функции не более 10000, максимальное значение функции 17,997.
n=50, количество нахождений глобального минимума 38%, число вычислений целевой функции не более 10000, максимальное значение функции 49,869.
Возможно, такие странные результаты обусловленны рельефом функции Растригина от 20 и 50 переменных, а может быть, и работой ГА. Например, с использованием генетического алгоритма **QGA** никак не удается точно аппроксимировать квадратичной функцией (всего-то три параметра) набор из пяти точек, которые строго принадлежат графику параболы.

Schwefel's (Sine Root) Function

$$f(x) = 418{,}9829\,n + \sum_{i=1}^{n} (-x_i \sin(\sqrt{|x_i|}))$$

x*O*(-500; 500)
Минимум равный 0 в точке, где x_i=420.9687.

Локальный минимум в точке, где одна координата равна -302.5232, а остальные равны 420.9687. Т.к. локальный минимум находится далеко от глобального, то есть опасность, что алгоритм "собъется с пути". Угол под знаком синуса получается в радианах.

 Griewangk's Function

$$f(x) = 1 + \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos \frac{x_i}{\sqrt{i}}$$

x*O*(-600; 600)
Минимум равный 0 в точке, где x_i=0.0.
При n=10 есть ещё 4 локальных минимума равных 0,0074 приблизительно в точке (+-P_i, +-P_i*sqrt(2), 0, ..., 0).

 Ackley's Function

$$f(x) = 20 + e - 20e^{-0,2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)}$$

x*O*(-30; 30)
Минимум равный 0 в точке, где x_i=0.0.

4.3. Примеры оптимизации функции с помощью программы FlexTool

Применение генетического алгоритма для оптимизации различных функций будем иллюстрировать примерами, реализованными на IBM PC совместимом компьютере с использованием программы **FlexTool**.
Программа **FlexTool** позволяет выбирать различные варианты генетического алгоритма - такие, как классический *(regular),* с частичной заменой популяции *(steady-state)* или микроалгоритм *(micro).* Помимо того, предоставляется возможность выбрать метод селекции (рулетка, турнирный или ранговый), а также количество точек скрещивания и способ кодирования (двоичное или логарифмическое). В дан-

ном случае под двоичным кодированием понимается способ, применявшийся в примерах 2 и 3, а логарифмическое кодирование было представлено в п. 3.13.4. Программа **FlexTool** также позволяет одновременно оптимизировать несколько функций и содержит алгоритм создания ниш для нахождения более чем одного оптимума (помимо единственного глобального решения). **FlexTool** взаимодействует с программой **MATLAB** и запускается в ее окне. Для определения оптимизируемых функций используются файлы с расширением *.m* (так называемые *m*-файлы), служащие для записи математических функций в программе **MATLAB.** В рабочем окне **MATLAB'а** также можно считывать значения различных переменных, используемых программой **FlexTool,** что дает, в частности, возможность просмотра популяций хромосом.

Программа **FlexTool** требует от пользователя ввести размерность популяций, вероятности скрещивания и мутации, интервалы изменения параметров задачи (пространства поиска), а также условия останова алгоритма. Предоставляется возможность прервать выполнение алгоритма в произвольный момент времени с последующим возобновлением его работы с точки прерывания (так называемый *warm-start)* либо с начальной точки *(cold-start).*

Рассмотрим примеры оптимизации функции одной, двух и нескольких переменных. В качестве иллюстраций берутся графики большинства оптимизируемых функций (полученные средствами программы **MATLAB)** и сформированные программой **FlexTool** графики функций приспособленности для конкретных итераций генетического алгоритма.

Пример 4.1 демонстрирует способ решения задачи, рассмотренной в примере 2, но при использовании генетического алгоритма, реализованного в программе **FlexTool.** Для такой простой задачи лучше всего подходит генетический микроалгоритм, представленный в п. 3.13.8. В отличие от примера 2, в примере 4.1 ищется минимум функции, заданной формулой $f(x) = 2x^2 + 1$.

Пример 4.1

С помощью генетического микроалгоритма программы **FlexTool** найти минимум функции, заданной формулой

$$f(x) = 2x^2 + 1$$

для целых значений x в интервале от 0 до 31.

Генетический микроалгоритм программы **FlexTool** выполняется на популяции с размерностью, равной пяти. Селекция производится детерминированным турнирным методом с применением элитарной

стратегии, благодаря которой лучшая хромосома текущей популяции всегда переходит в следующее поколение. Производится одноточечное скрещивание. Вероятность скрещивания принимается равной 1, а вероятность мутации - равной 0.

Длина хромосом равна пяти, что очевидно следует из возможности кодирования 32 целых чисел (для указанного интервала изменения переменной x) пятью битами двоичной последовательности. В программе FlexTool применяется двоичное кодирование, аналогичное представленному в примерах 3.1 и 2, однако для удобства программной реализации используется обратный код, т.е. на первой позиции находится наименее значащий бит, а на последней - наиболее значащий. Такой способ записи прямо противоположен повсеместно распространенной форме записи двоичных чисел.

Существенным элементом выполнения генетического микроалгоритма считается процедура «рестарта», т.е. запуска его с начальной точки при обнаружении сходимости (п. 3.13.8). В программе **FlexTool** рестарт производится периодически после выполнения установленного количества итераций. По умолчанию оно равно 7, примеры выполнялись именно при этом значении.

Исходная популяция - на первой итерации генетического микроалгоритма - состоит из следующих хромосом:

[01100] [11000] [01111] [10101] [11001]

со значениями фенотипов

6 3 30 21 19,

которые являются действительными целыми числами из интервала от 0 до 31.

Наименьшее значение функции приспособленности в этой популяции имеет стоящая на втором месте хромосома с фенотипом, равным трем. Оно равно 19. Наибольшее значение функции приспособленности у стоящей на третьем месте хромосомы с фенотипом, равным 30, составляет 1801. Легко подсчитать, что среднее значение функции приспособленности будет равно 699,8.

Популяция, полученная в результате селекции и скрещивания, становится текущей для следующей (второй) итерации. Она характеризуется средним значением функции приспособленности, равным 173,8. Заметно, что это значение меньше, чем рассчитанное для первого поколения. Наибольшее значение функции приспособленности, равное 723, имеет хромосома с фенотипом 19. Это наихудшая особь данной популяции. «Наилучшая к текущему моменту» хромосома с

фенотипом, равным двум, имеет минимальное значение функции приспособленности, которое равно девяти.

В третьем поколении среднее значение функции приспособленности уменьшается до 13. Наибольшее значение для хромосомы с фенотипом, равным трем, составляет 19, а наименьшее значение функции приспособленности для этой популяции, также, как и для предыдущей, равно девяти. «Наилучшей к текущему моменту» продолжает оставаться хромосома с фенотипом, равным двум.

В следующих четырех поколениях (от четвертого до седьмого) среднее значение функции приспособленности по популяции совпадает с наибольшим и наименьшим значениями, равными девяти. Это означает, что популяция состоит из одинаковых хромосом с фенотипами, равными двум. Следовательно, наблюдается сходимость к решению, которое не является оптимальным.

До этого момента алгоритм выполнялся аналогично классическому генетическому алгоритму, причем селекция проводилась детерминированным турнирным методом с сохранением наилучшей хромосомы из популяции (элитарная стратегия).

После семи итераций начинается новый цикл выполнения генетического микроалгоритма. Производится его «рестарт», т.е. повторный запуск алгоритма с начальной точки - случайного выбора четырех новых хромосом для включения в популяцию. Из особей предыдущего поколения сохраняется только одна - «наилучшая к текущему моменту» хромосома с фенотипом, равным двум.

После селекции и скрещивания в очередном (девятом) поколении получаем популяцию со средним значением функции приспособленности, равным 481,4. Наибольшее значение функции приспособленности, равное 1579, имеет хромосома с фенотипом 28, а наименьшее значение, равное девяти, - с фенотипом, равным двум.

На следующих пяти итерациях второго цикла генетического микроалгоритма мы вновь получаем популяцию, состоящую из одинаковых хромосом с фенотипом, равным двум, для которых среднее, наибольшее и наименьшее значения функции приспособленности равны девяти. Следовательно, опять наблюдается сходимость к решению, которое не является оптимальным.

С пятнадцатого поколения начинается очередной (третий) цикл выполнения генетического микроалгоритма. Он также открывается случайным выбором четырех новых хромосом и формированием популяции с сохранением «наилучшей к текущему моменту» особи с фенотипом, равным двум.

На девятнадцатой итерации генетического микроалгоритма, т.е. в пятом поколении третьего цикла (состоящего из семи итераций) возникает сходимость к оптимальному решению, которым оказывается хромосома с фенотипом, равным 0. Среднее, наибольшее и наименьшее значения функции приспособленности по всей популяции с 19 по 21 поколение остается равным 1.

Далее выполняются очередные циклы по семь итераций, начинающиеся «рестартом» алгоритма, и в каждом из них наблюдается сходимость к оптимальному решению, т.е. к хромосоме с фенотипом, равным 0.

Конечно, задачу из примера 4.1 можно решить с применением генетического алгоритма с произвольной размерностью популяций, при задании пользователем значений вероятностей скрещивания и мутации, а также при выборе им одного из методов селекции (рулетки, турнирного или рангового). Таким образом, вместо микроалгоритма можно применить реализованный в программе **FlexTool** обычный генетический алгоритм *(regular).* Однако необходимо отметить, что при его использовании на популяциях малой размерности с вероятностями скрещивания и мутации, соответственно равными 1 и 0, а также при выполнении селекции методом рулетки (так, как в примере 2) несьма часто встает проблема преждевременной сходимости этого алгоритма. Результат значительно улучшается, если вероятность мутации отличается от нуля (например, если она принимается равной 0,1). Несмотря на то, что мутация играет определенно второстепенную роль по отношению к скрещиванию, она оказывается необходимой, поскольку обеспечивает разнообразие хромосом в популяции.

Следует еще раз подчеркнуть, что при выполнении генетического микроалгоритма разнообразие в популяции достигается благодаря процедуре рестарта алгоритма в случае обнаружения его сходимости. В этом случае мутация оказывается излишней.

Следующий пример относится к задаче, похожей на задачу примера 4.1. Минимизируется та же функция, однако переменная x принимает действительные значения из интервала [-5, 5]. По этой причине пространство поиска оказывается большим, чем в примере 4.1. Для решения данной задачи применим обычный генетический алгоритм программы **FlexTool** и селекцию методом рулетки. Заметим, что этот метод, который пригоден и для селекции в случае максимизации

функции, в программе **FlexTool** может использоваться только для задач минимизации. Это обусловлено адаптацией программы к требованиям пользователей, которые чаще решают задачи минимизации (например, погрешностей, затрат и т.п.), чем задачи максимизации.

Пример 4.2

С помощью генетического алгоритма программы **FlexTool** найти минимум функции, заданной формулой $f(x) = 2x^2 + 1$ для $x \in [-5, 5]$ с точностью до 0,1.

Поставленная задача решается путем использования обычного генетического алгоритма (regular) с селекцией методом рулетки. Заметим, что поиск проводится в пространстве, состоящем из 100 возможных решений. Длина хромосом (в соответствии с формулой 3.4)) должна быть равной семи. Пусть размерность популяции составляет 11 особей. Будем применять одноточечное скрещивание с вероятностью 0,9, а вероятность мутации установим равной 0,1.

Исходная популяция состоит из 11 хромосом длиной семь битов, соответствующих следующим фенотипам:

3,4 2,4 2,0 5,0 1,4 0,1 3,0 -2,3 2,3 5,0 -4,8

Наилучшее решение, т.е. хромосома с фенотипом, равным 0, для которой значение функции приспособленности составляет 1, получено на седьмой итерации алгоритма. На первой итерации наибольшее значение функции приспособленности по всей популяции равно 51, среднее - 22,0745, а наименьшее - 1,02. Однако в седьмом поколении наибольшее значение функции приспособленности равно 45,18, среднее по популяции составляет 5,9909, а наименьшее значение соответствует лучшей хромосоме и равно 1. График на рис. 4.1 показывает наименьшее значение функции приспособленности в популяции на последовательных итерациях генетического алгоритма.

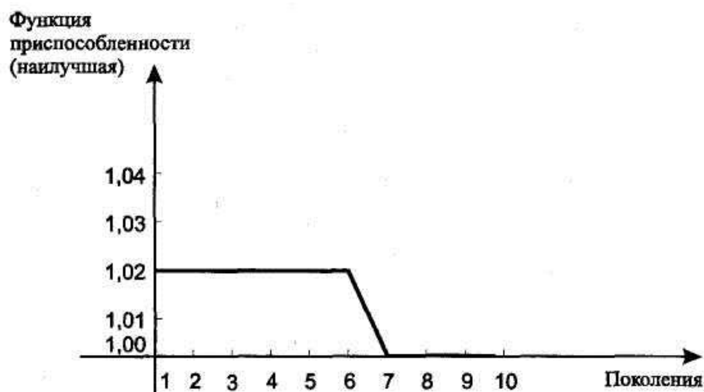


Рис.4.1. Наименьшее значение функции приспособленности в популяции на последовательных итерациях генетического алгоритма для примера 4.2.

Аналогично можно применять генетический алгоритм для нахождения решения с еще большей точностью, например, для двух или трех знаков после запятой. При этом будет соответственно расширяться пространство поиска и увеличиваться длина хромосом.

Следующий пример относится к задаче нахождения максимума функции, имеющей локальные максимумы. Покажем, что генетический алгоритм находит глобальный оптимум. Для поиска максимума будем применять реализованный в программе **FlexTool** турнирный метод, поскольку метод колеса рулетки в ней работает только для задач минимизации.

В программе **FlexTool** по умолчанию приняты следующие (наилучшие для большинства решаемых задач) значения параметров генетического алгоритма:

- размерность популяции = 77,
- вероятность скрещивания = 0,77,
- вероятность мутации = 0,0077.

В приводимых далее примерах используются указанные вероятности скрещивания и мутации, однако размерность популяций будет выбираться в зависимости от размерности решаемых задач.

Пример 4.3
С помощью генетического алгоритма программы **FlexTool** найти минимум функции, заданной формулой

$$f(x)=\frac{1}{(x-0{,}3)^2+0{,}01}+\frac{1}{(x-0{,}9)^2+0{,}04}-6$$

для $x \in [-1, 2]$ с точностью до 0,01.

График оптимизируемой функции представлен на рис. 4.2.

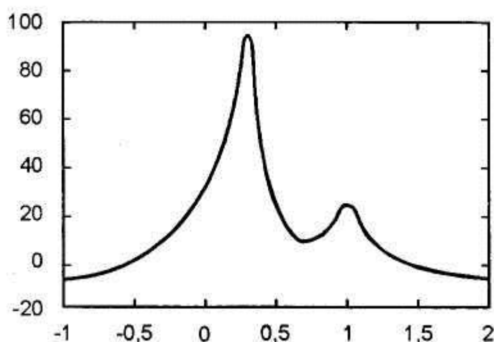


Рис. 4.2. График функции *f(x)* из примера 4.3.

Обратим внимание на то, что функция имеет не только глобальный максимум, но и локальные оптимумы. Для нахождения глобального максимума применяется генетический алгоритм с турнирной селекцией в подгруппах по две особи. Размерность популяции равна 21; вероятности скрещивания и мутации составляют, соответственно 0,77 и 0,0077; используется одна точка скрещивания.

Хромосомы состоят из девяти генов со значениями 0 или 1. На рисунке 4.3 более светлой линией показана динамика изменения «наилучшего», в рассматриваемом случае - максимального значения функции приспособленности в популяции при увеличении количества поколений.

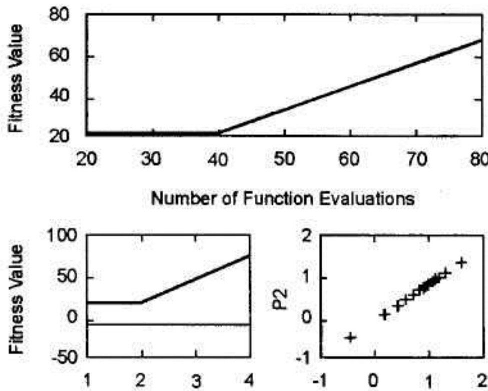


Рис. 4.3. Графики, показывающие значения функции приспособленности для первых четырех поколений в генетическом алгоритме программы **FlexTool** из примера 4.3.

Более темная линия на втором графике иллюстрирует изменение «наихудшего» значения функции приспособленности в популяции при последовательной смене поколений. На оси ординат обоих графиков указаны значения функции приспособленности *(fitness value).* На оси абсцисс второго графика указаны номера поколений, которые соответствуют числам на оси абсцисс первого графика, умноженным на размерность популяции, т.е. на 21.

Из графиков видно, что «наилучшая» хромосома в первом и втором поколениях характеризуется значением функции приспособленности, близким к 20. В то же время значение функции приспособленности «наихудшей» хромосомы примерно равно -5. В четвертом поколении это значение приблизилось к -4, а значение функции приспособленности «наилучшей» хромосомы возросло до 70. В нижней правой части рис. 4.3 показаны значения параметра задачи, т.е. переменной x со значениями в интервале от -1 до 2; другими словами, это фенотипы, соответствующие отдельным хромосомам рассматриваемой популяции. Заметно, что популяция характеризуется значительным разнообразием. На рис. 4.4 популяция более однородна, а на рис. 4.5 зафиксировано только одно значение параметра, равное 0,3; это означает, что все хромосомы в популяции равны между собой.

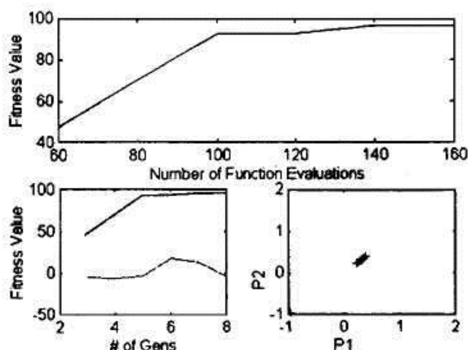


Рис. 4.4. Графики, показывающие значения функции приспособленности с третьего по восьмое поколение в генетическом алгоритме программы **FlexTool** из примера 4.3.

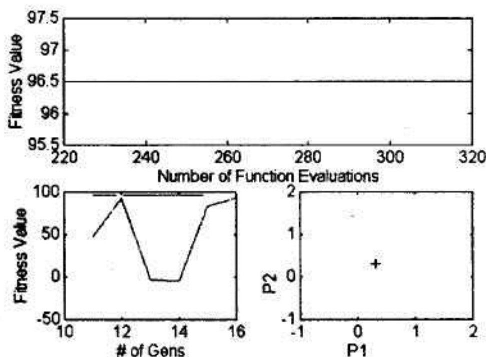


Рис. 4.5. Графики, показывающие значения функции приспособленности для последних четырех поколений в генетическом алгоритме программы **FlexTool** в примере 4.3.

В рассматриваемом примере мы имеем дело только с одним параметром задачи P1, который совпадает с P2, поэтому все точки располагаются вдоль прямой так, как это показано на рис. 4.3 и 4.4.
Графики на рис. 4.4 демонстрируют увеличение «наилучшего» значения функции приспособленности от около 47 в третьем поколении до примерно 70 в четвертом и примерно 93 - в пятом поколении, а в седьмом поколении достигается значение, равное 96,5. Это

максимальная величина, которая остается неизменной в следующих поколениях, что видно на рис.4.5. Таким образом, на седьмой итерации алгоритма найдено «наилучшее» решение - хромосома со значением фенотипа, равным 0,3, для которого значение функции приспособленности составляет 96,5. Нижние графики на рис. 4.4 и 4.5 также показывают, как изменяется «наихудшее» значение функции приспособленности от 3 до 16 поколения. В шестнадцатом поколении это значение равно 92,81. Среднее значение функции приспособленности в популяциях очередных поколений изменяется от 2,24 на первой итерации алгоритма до 96,32 на шестнадцатой итерации, что показывает таблица 4.1.

Таблица 4.1. Среднее значение функции приспособленности в популяциях очередных поколений генетического алгоритма программы **FlexTool** для примера 4.3

Номер поколения	1	2	3	4	5	6	7	8
Среднее значение функции приспособленности	2,24	7,76	14,15	20,08	32,49	50,59	65,47	75,89
Номер поколения	9	10	11	12	13	14	15	16
Среднее значение функции приспособленности	85,66	89,61	93,39	96,15	86,98	91,66	95,86	96,32

Пример 4.4

С помощью генетического алгоритма программы **FlexTool** найти минимум функции, заданной формулой

$f(x)=-x-\sin(10\pi x)+1$

для $x \in [-1, 2]$ с точностью до 0,0001.

График оптимизируемой функции представлен на рис.4.6. Эта функция имеет много локальных оптимумов.

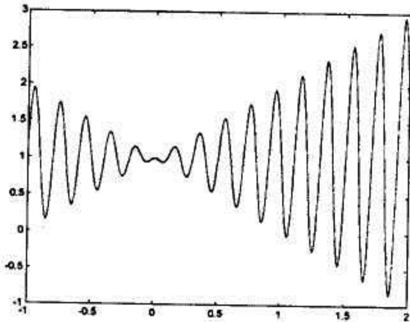


Рис.4.6. График функции *f(x)* из примера 4.4.

Для нахождения глобального минимума на интервале от -1 до 2 применяется (так же, как и в примере 4.3) генетический алгоритм с турнирной селекцией в подгруппах по две особи. Выполняется одноточечное скрещивание с вероятностью 0,77; вероятность мутации равна 0,0077. Размерность популяции увеличена до 55. Длина хромосом в этом случае составляет 15 битов. Графики, демонстрирующие изменения значений функции приспособленности при смене первых четырех поколений, по аналогии с примером 4.3 изображены на рис.4.7, а графики последующих изменений - на рис.4.8.

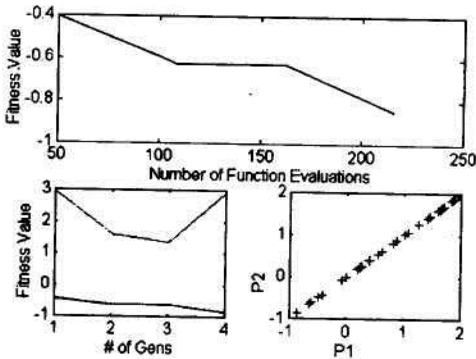


Рис.4.7. Графики, показывающие значения функции приспособленности для первых четырех поколений в генетическом алгоритме программы **FlexTool** из примера 4.4.

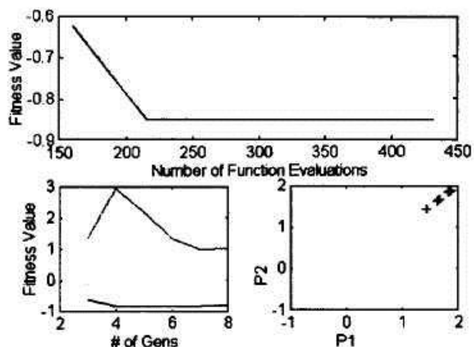


Рис.4.8. Графики, показывающие значения функции приспособленности с третьего по восьмое поколение в генетическом алгоритме программы **FlexTool** из примера 4.4.

«Наилучшее» решение дает хромосома со значением фенотипа 1,85, для которой функция приспособленности равна -0,8503. Это решение получено в десятом поколении. В таблицу 4.2 собраны «наилучшие» значения функции приспособленности на первых десяти итерациях алгоритма.

Таблица 4.2. «Наилучшее» значение функции приспособленности для первых десяти поколений генетического алгоритма программы **FlexTool** для примера 4.4

Номер поколения	1	2	3	4	5
«Наилучшее» значение функции приспособленности	–0,4197	–0,6247	–0,6256	–0,8498	–0,8499
Номер поколения	6	7	8	9	10
«Наилучшее» значение функции приспособленности	–0,8499	–0,8502	–0,8502	–0,8502	–0,8503

Значение фенотипа хромосомы с «наилучшим» значением функции приспособленности для второго поколения равно 1,645, а для четвертого и последующих поколений - 1,85.

Пример 4.5

С помощью генетического алгоритма программы **FlexTool** найти минимум функции двух переменных

$f(x_1,x_2) = x_1^2 + x_2^2$

для $x_1, x_2 \in [-10, 10]$ с точностью до 0,0001.

График оптимизируемой функции представлен на рис. 4.9.

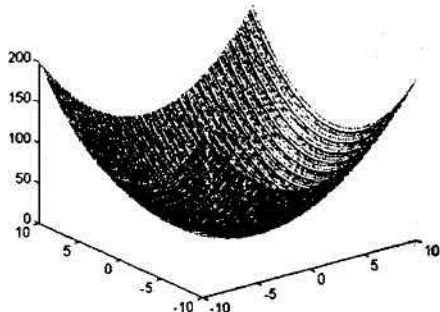


Рис. 4.9. График функции $f(x_1, x_2)$ из примера 4.5.

Эта функция имеет один минимум, равный 0, в точке (0, 0).

Применяется генетический алгоритм с турнирной селекцией в подгруппах по две особи, с одной точкой скрещивания и принятыми по умолчанию значениями вероятностей скрещивания 0,77 и мутации 0,0077, а также с размерностью популяции, равной 77.

Длина хромосом составляет 36 битов, при этом каждую переменную x_1, x_2 представляют 18 генов. Графики, демонстрирующие изменения значений функции приспособленности при смене первых четырех поколений при выполнении генетического алгоритма, изображены на рис.4.10, а графики последующих изменений - на рис. 4.11-4.14.

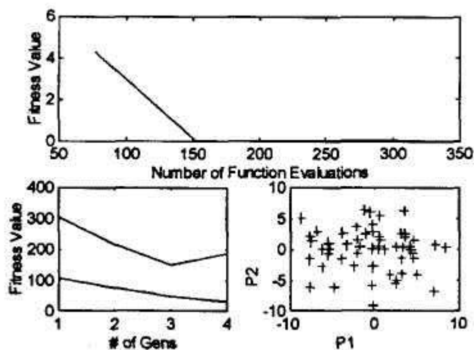


Рис.4.10. Графики, показывающие значения функции приспособленности для первых четырех поколений в генетическом алгоритме программы **FlexTool** из примера 4.5.

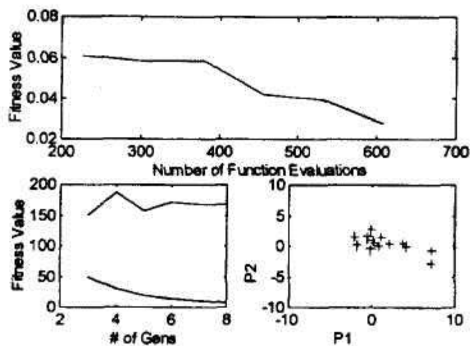


Рис. 4.11. Графики, показывающие значения функции приспособленности со второго по восьмое поколение в генетическом алгоритме программы **FlexTool** из примера 4.5.

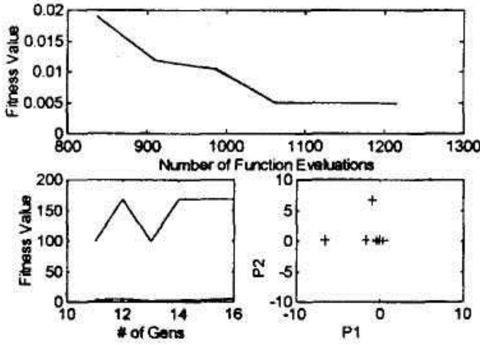


Рис.4.12. Графики, показывающие значения функции приспособленности с десятого по шестнадцатое поколение в генетическом алгоритме программы **FlexTool** из примера 4.5**.**

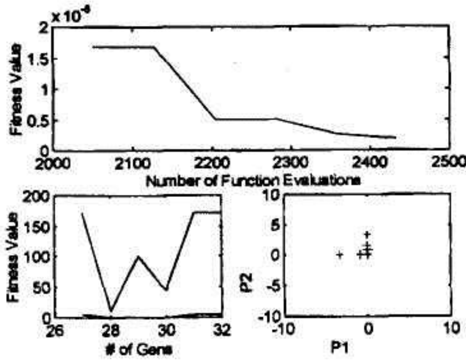


Рис.4.13. Графики, показывающие значения функции приспособленности с двадцать шестого по тридцать второе поколение в генетическом алгоритме программы **FlexTool** из примера 4.5.

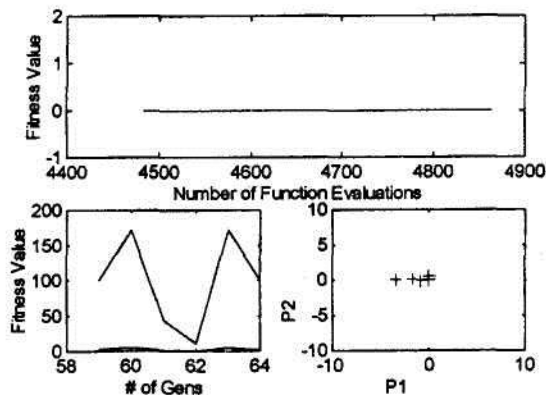


Рис.4.14. Графики, показывающие значения функции приспособленности с пятьдесят девятого по шестьдесят четвертое поколение в генетическом алгоритме программы **FlexTool** из примера 4.5.

По верхнему графику видно, как изменяется «наилучшее» значение функции приспособленности - от значения 4,3543 в первом поколении (по 77 расчетным точкам) до нулевого значения. Во втором и третьем поколениях эта функция имела значение 0,0608, в четвертом и пятом - 0,0586, в шестом - 0,0421; в седьмом - 0,0397, в восьмом - 0,0278, в девятом - 0,0192, в десятом и одиннадцатом - 0,0191, в двенадцатом - 0,0119, в тринадцатом - 0,0105, с четырнадцатого по шестнадцатое - 0,0050, в семнадцатом и восемнадцатом - 0,0011, в девятнадцатом - 0,0001, а в двадцатом и последующих поколениях - значение 0,0000. Весь комплекс изменений показан на рис. 4.16, а перечисленные «наилучшие» значения функции приспособленности выделены на рис.4.15. Таким образом, в результате выполнения генетического алгоритма «наилучшее» решение найдено в двадцатом поколении.

```
mini =
 Columns 1 through 7
  4.3543  0.0608  0.0608  0.0586  0.0586  0.0421  0.0397
 Columns 8 through 14
  0.0278  0.0192  0.0191  0.0191  0.0119  0.0105  0.0050
 Columns 15 through 21
  0.0050  0.0050  0.0011  0.0011  0.0001  0.0000  0.0000
 Columns 22 through 28
  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
 Columns 29 through 35
  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
 Columns 36 through 42
  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
 Columns 43 through 49
  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
 Columns 50 through 56
  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
 Columns 57 through 63
  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000  0.0000
 Column 64
  0.0000
```

Рис. 4.15. Перечень наилучших значений функции приспособленности в популяциях с первого по шестьдесят четвертое поколение в генетическом алгоритме программы **FlexTool** из примера 4.5.

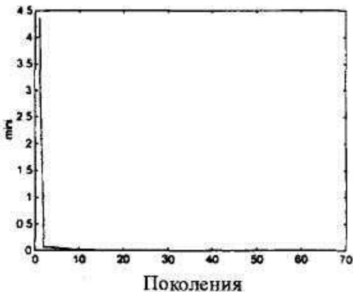


Рис. 4.16. Динамика изменения «наилучших» значений функции приспособленности при последовательной смене поколений в генетическом алгоритме программы **FlexTool** из примера 4.5.

Нижние левые графики на рис. 4.10- 4.14 показывают изменения «наихудшего» (верхняя кривая) и среднего значения функции

приспособленности особей популяции при смене поколений. Изменение «наилучшего» значения функции приспособленности на этих графиках почти незаметно, поскольку соответствующая кривая практически совпадает с горизонтальной осью, так как значения близки к 0. Комплексная динамика средних значений функции приспособленности на протяжении всех поколений демонстрируется на рис.4.17.

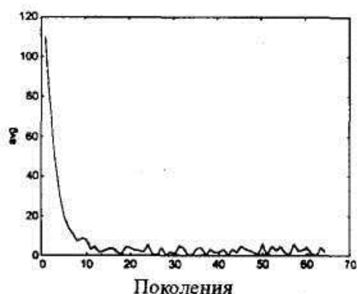


Рис.4.17. Динамика изменения средних значений функции приспособленности при последовательной смене поколений в генетическом алгоритме программы **FlexTool** из примера 4.5.

Обратим внимание на то, что значения функции приспособленности «наихудших» хромосом в отдельных популяциях довольно велики и, очевидно, значительно отличаются от оптимального значения. Динамика изменения «наихудших» значений функции приспособленности при смене поколений показана на рис.4.18.

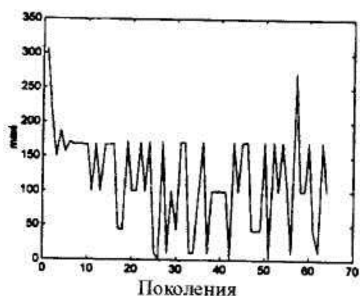


Рис. 4.18. Динамика изменения «наихудших» значений функции приспособленности при последовательной смене поколений в генетическом алгоритме программы **FlexTool** из примера 4.5.

На правых нижних графиках рисунков 4.10- 4.14 показаны значения параметров задачи P1 и P2, соответствующих переменным x_1, x_2, т.е. фенотипы (в интервале от -10 до 10). Наибольшее разнообразие особей наблюдается в начальных поколениях. В последующем в составе популяций появляется все больше одинаковых хромосом.

Напомним, что «наилучшее» решение, равное 0,0000, получено в двадцатом поколении (рис.4.15). Однако следует обратить внимание на динамику изменения «наилучшего» значения функции приспособленности, показанную на рис.4.13. Это значение находилось в интервале от 0 до 0,0002, начиная с 27 и по 32 поколение (т.е. с 27*77 до 32*77 расчетной точки функции приспособленности). Для «наилучшей» хромосомы со значениями фенотипов (после 64 поколений), равными 0,0000 и 0,0001, значение функции $f(x_1, x_2)$ составляет 0,00000001.

Пример 4.6

С помощью генетического алгоритма программы **FlexTool** найти минимум функции двух переменных

$f(x_1,x_2) = x_1^2+(1/2)x_2^2+3$

для $x_1, x_2 \in [-10, 10]$ с точностью до 0,001.

Данная задача аналогична предыдущей (пример 4.5). График оптимизируемой функции представлен на рис.4.19. Эта функция имеет один минимум, равный 3, в точке (0, 0).

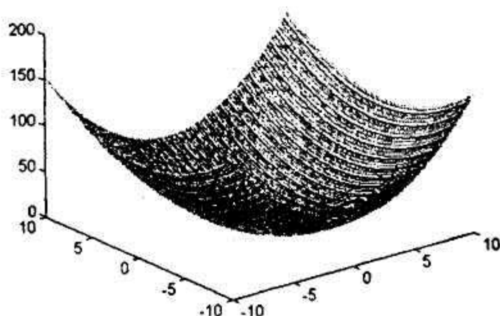


Рис.4.19. График функции $f(x_1, x_2)$ из примера 4.6.

Применяется генетический алгоритм с турнирной селекцией в подгруппах по две особи с одной или двумя точками скрещивания, а также с селекцией по методу рулетки с одной точкой скрещивания. Также как и в предыдущем примере, используются принятые по умол-

чанию значения вероятностей скрещивания 0,77 и мутации 0,0077, а также размерность популяции, равная 77. В этом случае с учетом меньшей требуемой точности длина хромосом составляет 22 бита - по 11 генов на каждую переменную.

Ввиду сходства рассматриваемого и предыдущего примеров графики изменения значений функции приспособленности для конкретных поколений не приводятся. Они аналогичны графикам на рис. 4.10-4.14. Принципиальное различие заключается в том, что в данном случае «наименьшее» значение функции приспособленности стремится к значению, равному 3.

В данном случае представляют интерес графики общей динамики изменения «наилучшего» значения функции приспособленности (подобные графику на рис.4.16) для использованных методов селекции. График для турнирной селекции с одной точкой скрещивания показан на рис.4.20, а с двумя точками скрещивания - на рис.4.21.

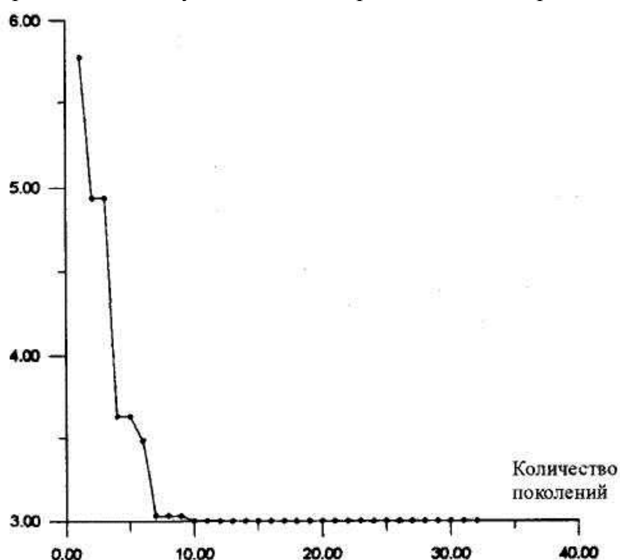


Рис.4.20. Динамика изменения «наилучших» значений функции приспособленности при последовательной смене поколений в генетическом алгоритме с турнирной селекцией и одной точкой скрещивания из примера 4.6.

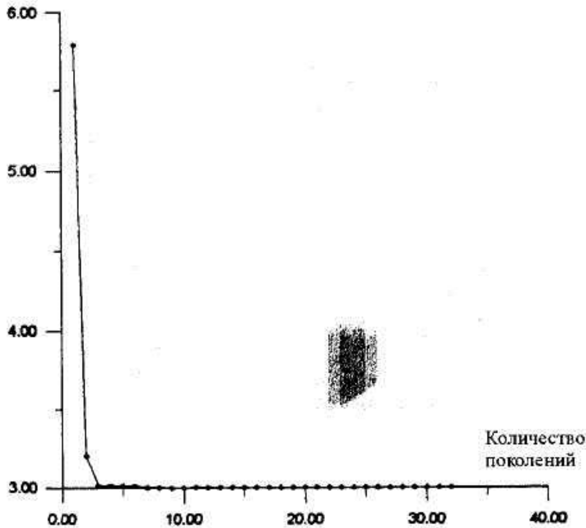


Рис.4.21. Динамика изменения «наилучших» значений функции приспособленности при последовательной смене поколений в генетическом алгоритме с турнирной селекцией и двумя точками скрещивания из примера 4.6.

Заметно, что во втором случае «наилучшее» решение (т.е. значение функции приспособленности, равное 3) было найдено быстрее. Аналогичный график для селекции методом рулетки представлен на рис.4.22.

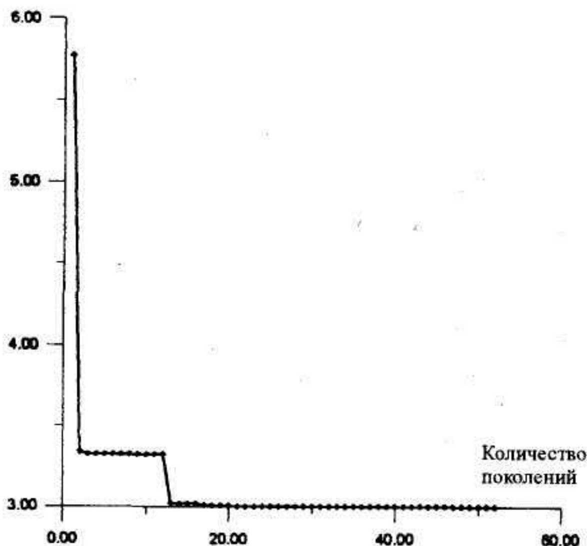


Рис.4.22. Динамика изменения «наилучших» значений функции приспособленности при последовательной смене поколений в генетическом алгоритме с селекцией методом рулетки из примера 4.6.

Можно сделать вывод, что турнирный метод позволяет быстрее находить минимальное значение оптимизируемой функции, чем метод рулетки.

Пример 4.7

С помощью генетического алгоритма программы **FlexTool** найти минимум функции двух переменных

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

для $x_1, x_2 \in [-10, 10]$ с точностью до 0,001.

График оптимизируемой функции представлен на рис.4.23.

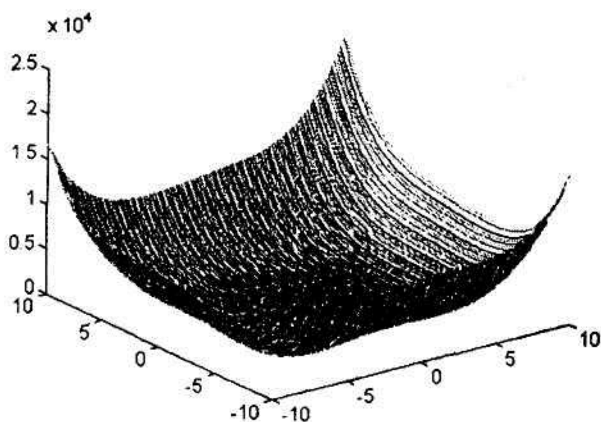


Рис.4.23. График функции $f(x_1, x_2)$ из примера 4.7.

Эта функция имеет минимум, равный 0, в следующих точках: (3, 2), (3,58, -1,85), (-2,80, 3,13), (-3,78, -3,28).

В случае, когда функция имеет минимальное (или максимальное) значение в нескольких различных точках, и это значение является глобальным оптимумом, генетический алгоритм находит, как правило, одну из этих точек. Повторный запуск алгоритма может принести тот же самый результат либо обнаружить координаты другой точки с таким же оптимальным значением функции.

В примере вначале применялся генетический алгоритм с турнирной селекцией в подгруппах по две особи с одной точкой скрещивания и принятые по умолчанию значения вероятностей скрещивания 0,77 и мутации 0,0077, а также размерность популяции, равная 77. Наилучшим решением, найденным в этих условиях, стала точка с координатами (-3,78, -3,28).

Динамику изменения «наилучшего» значения функции приспособленности показывают графики на рис. 4.24-4.26.

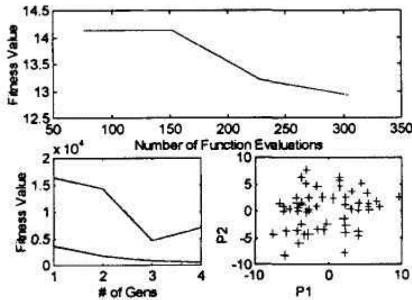


Рис.4.24. Графики, показывающие значения функции приспособленности для первых четырех поколений в генетическом алгоритме с турнирной селекцией из примера 4.7.

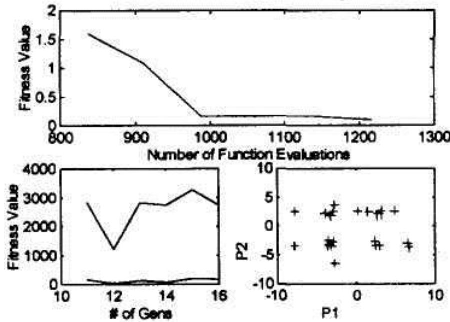


Рис.4.25. Графики, показывающие значения функции приспособленности с десятого по шестнадцатое поколение в генетическом алгоритме с турнирной селекцией из примера 4.7.

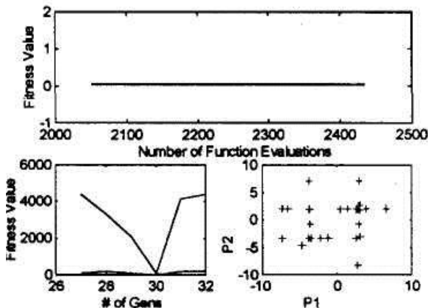


Рис.4.26. Графики, показывающие значения функции приспособленности с двадцать шестого по тридцать второе поколение в генетическом алгоритме с турнирной селекцией из примера 4.7.

Для 32 поколений это значение равно 0,0474, а в 46 поколениях достигается величина 0,0005. На этих же рисунках представлено и изменение «наихудшего» и среднего значений функции приспособленности по популяции при последовательной смене поколений, а также распределение особей в популяции (параметры P1 и P2). В дальнейшем алгоритм был выполнен еще один раз, причем для эксперимента применялась селекция методом рулетки, а вероятности скрещивания и мутации были установлены равными 0,6 и 0,001 соответственно. Графики, иллюстрирующие изменение значений функции приспособленности при последовательной смене поколений, представлены на рис. 4.27 и 3.48.

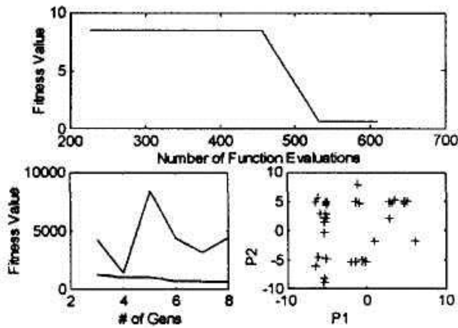


Рис. 4.27. Графики, показывающие значения функции приспособленности с третьего по восьмое поколение в генетическом алгоритме с селекцией методом рулетки из примера 4.7.

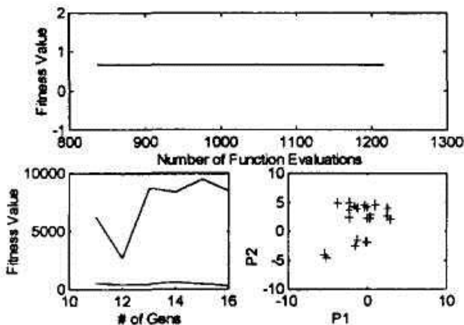


Рис. 4.28. Графики, показывающие значения функции приспособленности с десятого по шестнадцатое поколение в генетическом алгоритме с селекцией методом рулетки из примера 4.7.

«Наилучшее» значение функции приспособленности в 70 поколениях равно 0,0038, а начиная с 83 поколения оно составляет 0,0009. «Наилучшей» особью оказалась хромосома со значениями фенотипов -2,80 и 3,13. Таким образом, в этом эксперименте была найдена другая точка с координатами (-2,80, 3,13), в которой минимизируемая функция принимает нулевое значение.

Попытки поиска других оптимальных точек, соответствующих «наилучшему» решению, можно продолжать с применением того же алгоритма либо с использованием альтернативных методов селекции. Иногда такие попытки приходится повторять по несколько раз, если они приводят к полученным ранее решениям.

Пример 4.8

С помощью генетического алгоритма программы **FlexTool** найти минимум и максимум функции

$$f(x_1,x_2) = 2(1-x_1)^2 e^{-x_1^2-(x_2+1)^2} - 7\left(\frac{x_1}{3} - x_1^3 - x_2^2\right)e^{-x_1^2-x_2^2} - \frac{1}{5}e^{-(x_1+1)^2-x_2^2}$$

для $x_1, x_2 \in [-3, 3]$ с точностью до 0,01.

График оптимизируемой функции представлен на рис.4.29.

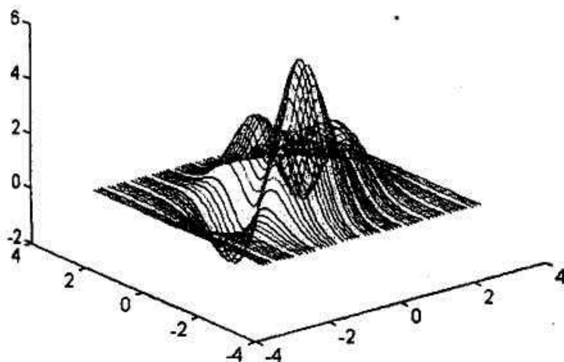


Рис. 4.29. График функции $f(x_1, x_2)$ из примера 4.8.

Эта функция двух переменных имеет несколько так называемых пиков. Ее контурный график изображен на рис.4.30.

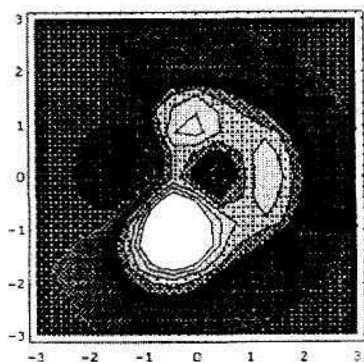


Рис. 4.30. Контурный график функции $f(x_1, x_2)$ из примера 4.8.

Применяется генетический алгоритм с турнирной селекцией в подгруппах по две особи с одной точкой скрещивания; по умолчанию приняты значения вероятностей скрещивания 0,77 и мутации 0,0077, а также размерность популяции, равная 77. В точке (-1,4, 0,17) найден минимум, равный -1,907, а в точке
(-0,39, -0,99) - максимум, равный 5,638. Точнее говоря, при минимизации наилучшим решением оказалась хромосома со значениями фенотипов -1,4 и 0,17, для которой значение функции приспособленности составляет -1,907, а при максимизации - хромосома со значениями фенотипов -0,39 и
-0,99 со значением функции приспособленности 5,638. Динамика изменения значений функции приспособленности при последовательной смене поколений для случая минимизации показана на рис. 4.31-4.35, а для случая максимизации - на рис. 4.36-4.38.

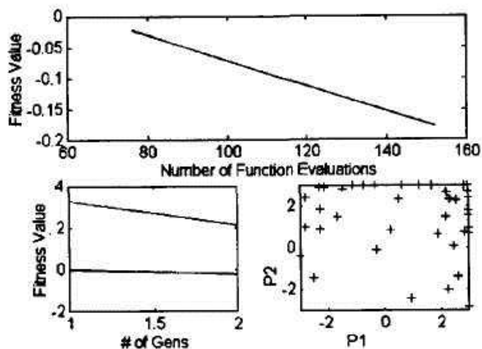


Рис.4.31. Графики, показывающие значения функции приспособленности для первых двух поколений в случае поиска минимума из примера 4.8.

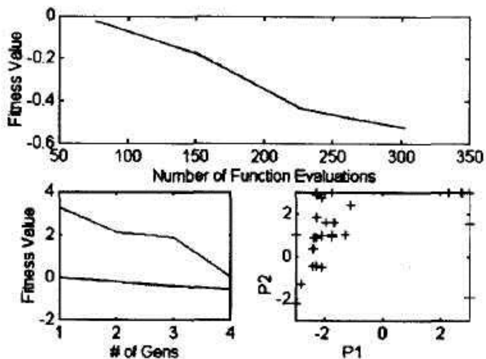


Рис.4.32. Графики, показывающие значения функции приспособленности для первых четырех поколений в случае поиска минимума из примера 4.8.

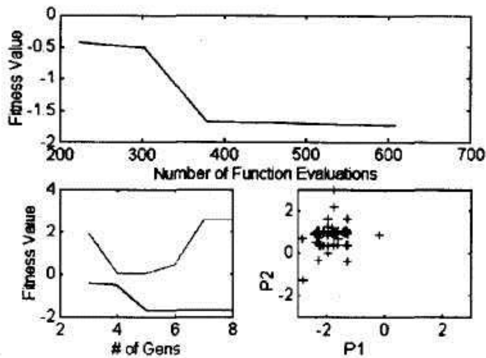


Рис.4.33. Графики, показывающие значения функции приспособленности со второго по восьмое поколение в случае поиска минимума из примера 4.8.

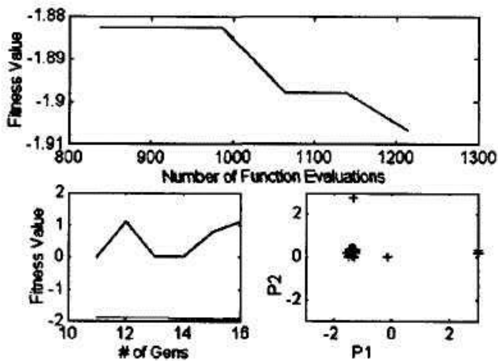


Рис.4.34. Графики, показывающие значения функции приспособленности с десятого по шестнадцатое поколение в случае поиска минимума из примера 4.8.

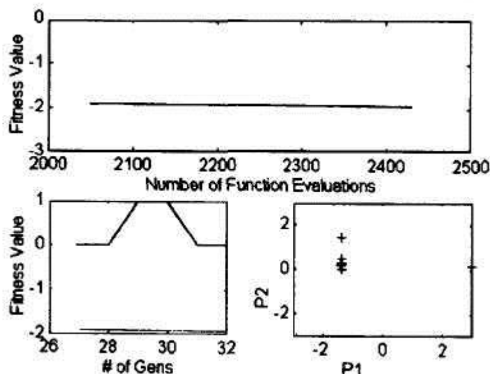


Рис.4.35. Графики, показывающие значения функции приспособленности с двадцать шестого по тридцать второе поколение в случае поиска минимума из примера 4.8.

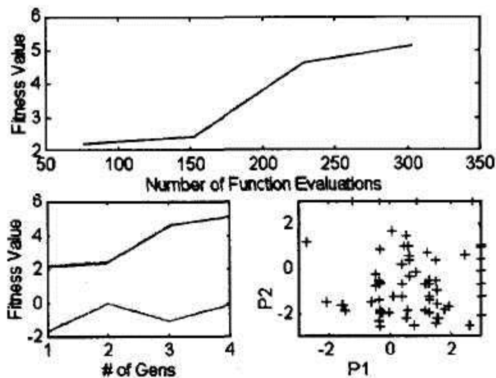


Рис.4.36. Графики, показывающие значения функции приспособленности для первых четырех поколений в случае поиска максимума из примера 4.8.

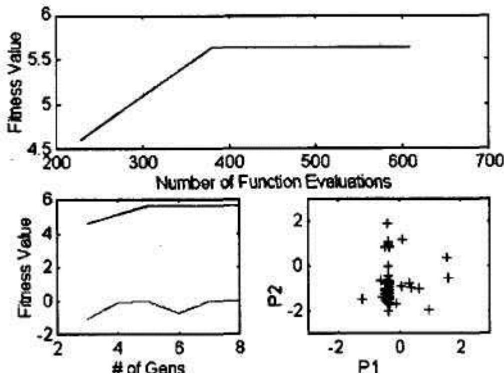


Рис.4.37. Графики, показывающие значения функции приспособленности с третьего по восьмое поколение в случае поиска максимума из примера 4.8.

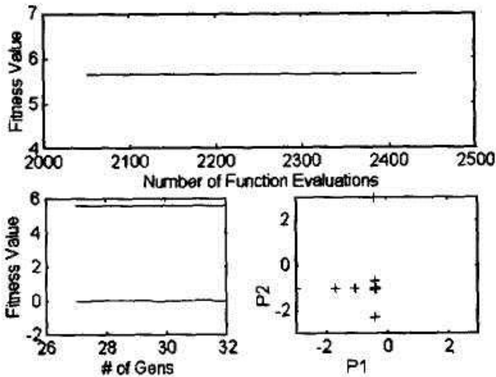


Рис.4.38. Графики, показывающие значения функции приспособленности с двадцать седьмого по тридцать второе поколение в случае поиска максимума из примера 4.8.

Обратим внимание на распределение особей в популяции в зависимости от номера поколения. Вначале (рис.4.31) заметно довольно большое разнообразие и относительно равномерное распределение отдельных точек на плоскости, заданной осями P1 и P2 в интервале от -3 до 3. На рис. 4.32(четвертое поколение) наблюдается отчетливое их смещение в направлении значений P1, близких к -2, а на рис. 4.33 - группирование поблизости значений P2, примерно равных 0,5. На рис.

4.34 большинство точек находится в окрестности оптимального решения с координатами P1 = -1,4 и P2 = 0,17. Из рис. 4 .35 следует, что большинство хромосом в популяции совпадает с «наилучшей» особью, для которой функция приспособленности принимает минимальное значение, равное -1,907. Также необходимо добавить, что этого минимального значения функция приспособленности достигла в шестнадцатом поколении выполнения генетического алгоритма. Верхние графики на рис. 4.31-4.35 иллюстрируют изменение «наилучшего» значения функции приспособленности при последовательной смене поколений (числа на оси абсцисс соответствуют номерам поколений, умноженным на 77). Нижние левые графики на этих рисунках отображают динамику изменения «наилучшего» и «наихудшего» значений функции приспособленности при последовательной смене поколений.

Аналогичные графики приведены на рис. 4.36-4.38; они демонстрируют, как изменяются «наилучшее» и «наихудшее» значения функции приспособленности при последовательной смене поколений в случае поиска максимума. На этих рисунках можно проследить уменьшение разнообразия хромосом в популяции. На рис. 4.38 практически все особи одинаковы и равны «наилучшей» хромосоме со значениями фенотипов -0,39 и -0,99. Функция приспособленности этой хромосомы равна 5,638. Эта точка максимума функции $f(x_1, x_2)$ найдена (также, как и в случае минимизации) в шестнадцатом поколении выполнения генетического алгоритма.

Пример 4.9

С помощью генетического алгоритма программы **FlexTool** найти оптимальный набор весов w_{11}, w_{12}, w_{21}, w_{22}, w_{31}, w_{32}, w_{10}, w_{20}, w_{30} в диапазоне от -10 до 10 для нейронной сети, изображенной на рис. 3.11.

Итак, необходимо решить задачу, поставленную в примере 3.3, т.е. найти значения девяти переменных, обозначающих веса нейронной сети, для которых функция погрешности Q достигает минимального значения, равного 0. Для сети, реализующей логическую систему XOR, функцию погрешности можно определить в виде

$$Q = \frac{1}{4}\sum_{i=1}^{4} \varepsilon_i^{\,2}$$

где

$\varepsilon_i = d_i$ - (1/ (1 + exp((-β)(w_{31}(1/ (1 + $exp((-\beta)(w_{11}u_{1,i} + w_{12}u_{2,i} + w_{10})))) + +$ w_{32}(1 / (1 +exp((-β)($w_{21}u_{1,i} + w_{22}u_{2i} + w_{20}$)))) + w_{30}))))

для i=1,2,3, 4. Значения $u_{1,i}$, $u_{2,i}$, и $d_{1,i}$ - приведены в таблице

u_1	u_2	d = XOR(u_1,u_2)
+1	+1	–1
+1	–1	+1
–1	+1	+1
–1	–1	–1

Предполагается, что $\beta = 1$.

Применяется генетический алгоритм с турнирной селекцией в подгруппах по две особи с одной точкой скрещивания. По умолчанию приняты установленные в программе **FlexTool** значения вероятностей скрещивания 0,77 и мутации 0,0077, а также размерность популяции, равная 77. Длина хромосом для решаемой задачи равна 99 битам - по 11 генов на каждую переменную.

Динамика изменения значений функции приспособленности при последовательной смене поколений иллюстрируется графиками на рис.4.39-4.47.

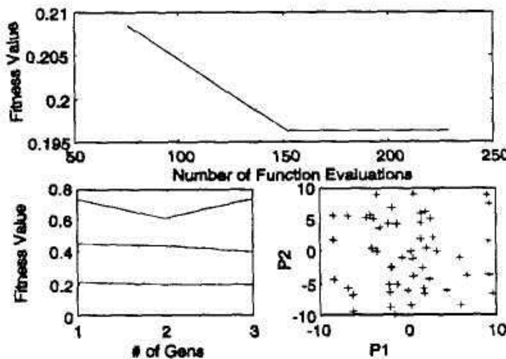


Рис.4.39. Графики, показывающие значения функции приспособленности для первых трех поколений в генетическом алгоритме программы **FlexTool** из примера 4.9.

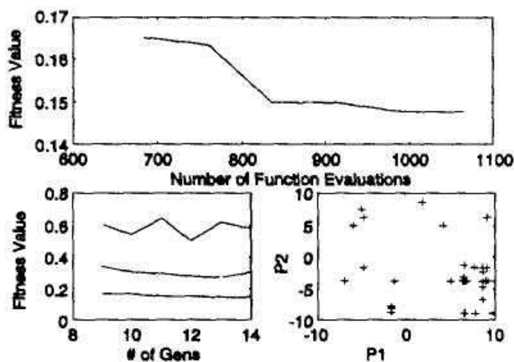


Рис.4.40. Графики, показывающие значения функции приспособленности с девятого по четырнадцатое поколение в генетическом алгоритме программы **FlexTool** из примера 4.9.

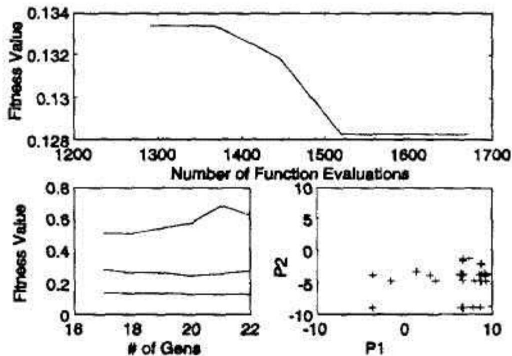


Рис.4.41. Графики, показывающие значения функции приспособленности с семнадцатого по двадцать второе поколение в генетическом алгоритме программы **FlexTool из** примера 4.9.

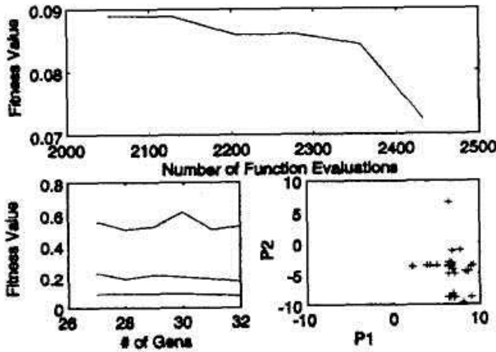


Рис.4.42. Графики, показывающие значения функции приспособленности с двадцать седьмого по тридцать второе поколение в генетическом алгоритме программы **FlexTool** из примера 4.9.

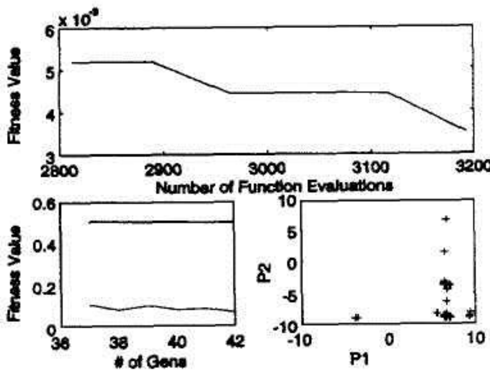


Рис.4.43. Графики, показывающие значения функции приспособленности с тридцать седьмого по сорок второе поколение в генетическом алгоритме программы **FlexTool** из примера 4.9.

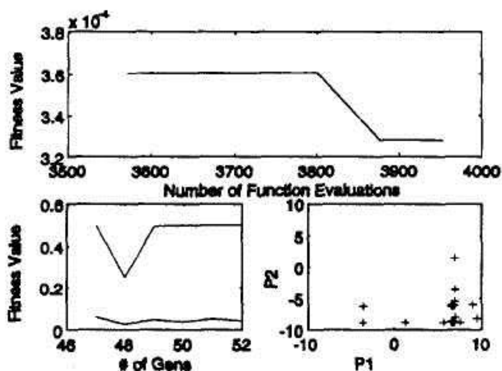


Рис.4.44. Графики, показывающие значения функции приспособленности с сорок седьмого по пятьдесят второе поколение в генетическом алгоритме программы **FlexTool** из примера 4.9.

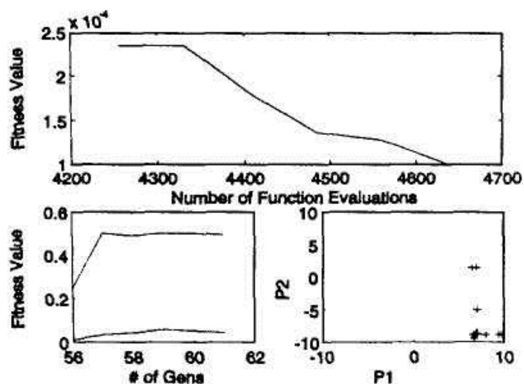


Рис.4.45. Графики, показывающие значения функции приспособленности с пятьдесят шестого по шестьдесят первое поколение в генетическом алгоритме программы **FlexTool** из примера 4.9.

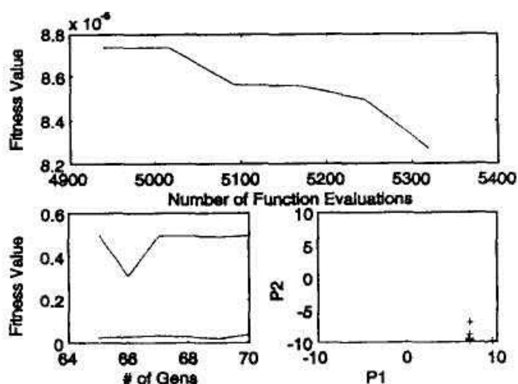


Рис.4.46. Графики, показывающие значения функции приспособленности с шестьдесят пятого по семидесятое поколение в генетическом алгоритме программы **FlexTool** из примера 4.9.

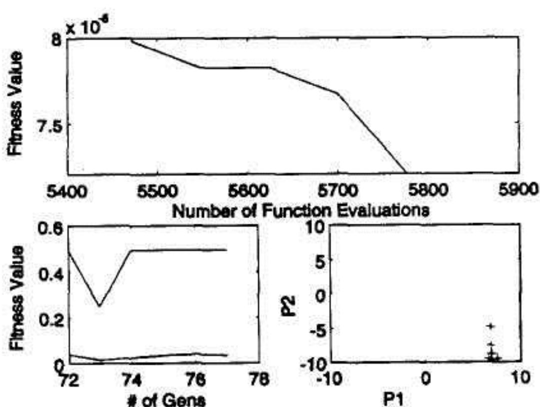


Рис.4.47. Графики, показывающие значения функции приспособленности с семьдесят второго по семьдесят седьмое поколение в генетическом алгоритме программы **FlexTool** из примера **4.9**.

На них также показано распределение особей в зависимости от номера поколения, причем отмечены только параметры Р1 и Р2, соответствующие переменным w_{12} и w_{21} «Наилучшее» значение

функции приспособленности изменяется от значения, примерно равного 0,21, до 0,00007. Напомним, что функция приспособленности соответствует функции погрешности Q, значение которой может изменяться от 1 до 0.

Нижние левые графики иллюстрируют динамику изменения «наилучшего» (нижняя кривая), «наихудшего» (верхняя кривая) и среднего (средняя кривая) значений функции приспособленности в популяции при последовательной смене поколений. На рис. 4.39-4.47 «наилучшее» значение функции приспособленности принимает настолько малое значение, что оно становится незаметным на графиках. В 75-м поколении оно становится равным $7*10^{-5}$, т.е. таким, которое принимается в качестве минимального значения. Таким образом, эта величина может считаться значением функции приспособленности «наилучшей» хромосомы со следующими фенотипами - значениями отдельных переменных, обозначающих веса нейронной сети:

$w_{11}=$ 7,65
w_{12} = - 8,43
w_{21} = -10,00
w_{22} = 9,98
w_{31} = - 4,28
w_{32} = - 3,96
w_{10} = - 4,28
w_{20} - - 3,96
w_{30} = - 4,96

Вычисления были завершены после 77 итераций алгоритма. В случае их продолжения можно было бы ожидать получение результата со значениями для w_{11}, w_{12}, w_{21}, w_{22}, w_{31}, w_{32} оказались бы примерно равны соответственно 10, -10, -10, 10,
-10, -10, а w_{10}, w_{20}, w_{3o} - величине -5. Также очевидно, что уменьшилось бы и значение погрешности Q; скорее всего, оно приблизилось бы к величине, полученной с помощью программы **Evolver**. Рассчитанные значения представляют собой одну из возможных комбинаций весов для нейронной сети, реализующей логическую систему XOR и изображенной на рис. 3.11.

4.4. Генетические алгоритмы и математический аппарат

Как нам известно, генетические алгоритмы предназначены, в основном, для решения задач оптимизации. Примером подобной задачи может служить обучение нейросети, то есть подбора таких значений весов, при которых достигается минимальная ошибка. При этом в основе генетического алгоритма лежит метод случайного поиска. Основным недостатком случайного поиска является то, что нам неизвестно сколько понадобится времени для решения задачи. Для того, чтобы избежать таких расходов времени при решении задачи, применяются методы, проявившиеся в биологии. При этом используются методы открытые при изучении эволюции и происхождения видов. Как известно, в процессе эволюции выживают наиболее приспособленные особи. Это приводит к тому, что приспособленность популяции возрастает, позволяя ей лучше выживать в изменяющихся условиях.

Впервые подобный алгоритм был предложен в 1975 году Джоном Холландом (John Holland) в Мичиганском университете. Он получил название "репродуктивный план Холланда" и лег в основу практически всех вариантов генетических алгоритмов. Однако, для выполнения генетического алгоритма, необходимо закодировать вводимую в него информацию. Рассмотрим вопрос на том, каким образом объекты реального мира могут быть закодированы для использования в генетических алгоритмах. Для цельности изложения материала, опишем некоторые элементарные сведения о генетических алгоритмах, которые уже были нами рассмотрены ранее.

Представление объектов

Из биологии мы знаем, что любой организм может быть представлен своим *фенотипом*, который фактически определяет, чем является объект в реальном мире, и *генотипом*, который содержит всю информацию об объекте на уровне хромосомного набора. При этом каждый ген, то есть элемент информации генотипа, имеет свое отражение в фенотипе. Таким образом, для решения задач нам необходимо представить каждый признак объекта в форме, подходящей для использования в генетическом алгоритме. Все

дальнейшее функционирование механизмов генетического алгоритма производится на уровне генотипа, позволяя обойтись без информации о внутренней структуре объекта, что и обуславливает его широкое применение в самых разных задачах.

В наиболее часто встречающейся разновидности генетического алгоритма для представления генотипа объекта применяются битовые строки. При этом каждому атрибуту объекта в фенотипе соответствует один *ген* в генотипе объекта. Ген представляет собой битовую строку, чаще всего фиксированной длины, которая представляет собой значение этого признака.

Кодирование признаков, представленных целыми числами

Для кодирования таких признаков можно использовать самый простой вариант – битовое значение этого признака. Тогда нам будет весьма просто использовать ген определенной длины, достаточной для представления всех возможных значений такого признака. Но, к сожалению, такое кодирование не лишено недостатков. Основной недостаток заключается в том, что соседние числа отличаются в значениях нескольких битов, так например числа 7 и 8 в битовом представлении различаются в 4-х позициях, что затрудняет функционирование генетического алгоритма и увеличивает время, необходимое для его сходимости. Для того, чтобы избежать эту проблему лучше использовать кодирование, при котором соседние числа отличаются меньшим количеством позиций, в идеале значением одного бита. Таким кодом является код Грея, который целесообразно использовать в реализации генетического алгоритма. Значения кодов Грея рассмотрены в таблице ниже:

Таблица 4.3. Соответствие десятичных кодов и кодов Грея.

Двоичное кодирование			**Кодирование по коду Грея**		
Десятичный код	Двоичное значение	Шестнадцатеричное значение	Десятичный код	Двоичное значение	Шестнадцатеричное значение
0	0000	0h	0	0000	0h
1	0001	1h	1	0001	1h
2	0010	2h	3	0011	3h
3	0011	3h	2	0010	2h

4	0100	4h	6	0110	6h
5	0101	5h	7	0111	7h
6	0110	6h	5	0101	5h
7	0111	7h	4	0100	4h
8	1000	8h	12	1100	Ch
9	1001	9h	13	1101	Dh
10	1010	Ah	15	1111	Fh
11	1011	Bh	14	1110	Eh
12	1100	Ch	10	1010	Ah
13	1101	Dh	11	1011	Bh
14	1110	Eh	9	1001	9h
15	1111	Fh	8	1000	8h

Таким образом, при кодировании целочисленного признака мы разбиваем его на тетрады и каждую тетраду преобразуем по коду Грея.

В практических реализациях генетических алгоритмов обычно не возникает необходимости преобразовывать значения признака в значение гена. На практике имеет место обратная задача, когда по значению гена необходимо определить значение соответствующего ему признака.

Таким образом, задача декодирования значения генов, которым соответствуют целочисленные признаки, тривиальна.

Кодирование признаков, которым соответствуют числа с плавающей точкой

Самый простой способ кодирования, который лежит на поверхности – использовать битовое представление. Хотя такой вариант имеет те же недостатки, что и для целых чисел. Поэтому на практике обычно применяют следующую последовательность действий:

1. Разбивают весь интервал допустимых значений признака на участки с требуемой точностью.
2. Принимают значение гена как целочисленное число, определяющее номер интервала (используя код Грея).

3. В качестве значения параметра принимают число, являющиеся серединой этого интервала.

Рассмотрим вышеописанную последовательность действий на примере:

Допустим, что значения признака лежат в интервале [0,1]. При кодировании использовалось разбиение участка на 256 интервалов. Для кодирования их номера нам потребуется таким образом 8 бит. Допустим значение гена: 00100101bG (заглавная буква G показывает, что используется кодирование по коду Грея). Для начала, используя код Грея, найдем соответствующий ему номер интервала:

25hG->36h->54d. Теперь посмотрим, какой интервал ему соответствует… После несложных подсчетов получаем интервал [0,20703125, 0,2109375]. Значит значение нашего параметра будет (0,20703125+0,2109375)/2=0,208984375.

Кодирование нечисловых данных

При кодировании нечисловых данных необходимо предварительно преобразовать их в числа.

Определение фенотипа объекта по его генотипу

Таким образом, для того, чтобы определить фенотип объекта (то есть значения признаков, описывающих объект) нам необходимо только знать значения генов, соответствующим этим признакам, то есть генотип объекта. При этом совокупность генов, описывающих генотип объекта, представляет собой *хромосому*. В некоторых реализациях ее также называют особью. Таким образом, в реализации генетического алгоритма хромосома представляет собой битовую строку фиксированной длины. При этом каждому участку строки соответствует ген. Длина генов внутри хромосомы может быть одинаковой или различной. Чаще всего применяют гены одинаковой длины. Рассмотрим пример хромосомы и интерпретации ее значения. Допустим, что у объекта имеется 5 признаков, каждый закодирован геном длинной в 4 элемента. Тогда длина хромосомы будет 5*4=20 бит

0010 1010 1001 0100 1101

теперь мы можем определить значения признаков

Признак	Значение гена	Двоичное значение признака	Десятичное значение признака
Признак 1	0010	0011	3
Признак 2	1010	1100	12
Признак 3	1001	1110	14
Признак 4	0100	0111	7
Признак 5	1101	1001	9

Непрерывные генетические алгоритмы - математический аппарат

Фиксированная длина хромосомы и кодирование строк двоичным алфавитом преобладали в теории ГА с момента начала ее развития, когда были получены теоретические результаты о целесообразности использования именно двоичного алфавита. К тому же, реализация такого ГА на ЭВМ была сравнительно легкой. И все же, небольшая группа исследователей шла по пути применения в ГА отличных от двоичных алфавитов для решения частных прикладных задач. Одной из таких задач является нахождение решений, представленных в форме вещественных чисел, что называется не иначе как "поисковая оптимизация в непрерывных пространствах". Возникла следующая идея: решение в хромосоме представлять напрямую в виде набора вещественных чисел. Естественно, что потребовались специальные реализации биологических операторов. Такой тип генетического алгоритма получил название *непрерывного ГА* (real-coded GA), или *генетического алгоритма с вещественным кодированием*.

Первоначально непрерывные гены стали использоваться в специфических приложениях (например, хемометрика, оптимальный подбор параметров операторов стандартных ГА и др.). Позднее они начинают применяться для решения других задач оптимизации в непрерывных пространствах (работы исследователей Wright, Davis,

Michalewicz, Eshelman, Herrera в 1991-1995 гг). Поскольку до 1991 теоретических обоснований работы непрерывных ГА не существовало, использование этого нового подвида было спорным; ученые, знакомые с фундаментальной теорией эволюционных вычислений, в которой было доказано превосходство двоичного алфавита перед другими, критически воспринимали успехи real-coded алгоритмов. После того, как спустя некоторое время теоретическое обоснование появилось, непрерывные ГА полностью вытеснили двоичные хромосомы при поиске в непрерывных пространствах.

Далее в тексте по аналогии с англоязычной терминологией для ГА с двоичным кодированием будет использоваться аббревиатура BGA (Binary coded), для ГА с непрерывными генами – RGA (Real coded).

Преимущества и недостатки двоичного кодирования

Прежде чем излагать особенности математического аппарата непрерывных ГА, остановимся на анализе достоинств и недостатков двоичной схем кодирования.
Как известно, высокая эффективность отыскания глобального минимума или максимума генетическим алгоритмом с двоичным кодированием теоретически обоснована в фундаментальной теореме генетических алгоритмов ("теореме о схеме"), доказанной Холландом. Ее подробное освещение и доказательство было приведено нами ранее. Напомнм, что ее суть заключается в том, что двоичный алфавит позволяет обрабатывать максимальное количество информации по сравнению с другими схемами кодирования.

Однако двоичное представление хромосом влечет за собой определенные трудности при поиске в непрерывных пространствах большой размерности, и когда требуется высокая точность найденного решения. В BGA для преобразования генотипа в фенотип используется специальный прием, основанный на том, что весь интервал допустимых значений признака объекта $[a_i,b_i]$ разбивается на участки с требуемой точностью. Заданная точность *p* определяется выражением

$$p = \frac{b_i - a_i}{2^N - 1},$$

где N – количество разрядов для кодирования битовой строки.

Эта формула показывает, что p сильно зависит от N, т.е. точность представления определяется количеством разрядов, используемых для кодирования одной хромосомы. Поэтому при увеличении N пространство поиска расширяется и становится огромным. Известный пример: пусть для 100 переменных, изменяющихся в интервале [-500; 500], требуется найти экстремум с точностью до шестого знака после запятой. В этом случае при использовании ГА с двоичным кодированием длина строки составит 3000 элементов, а пространство поиска – около 10^{1000} хромосом. Эффективность BGA в этом случае будет невысокой. На первых итерациях алгоритм потратит много усилий на оценку младших разрядов числа, закодированных во фрагменте двоичной хромосомы. Но оптимальное значение на первых итерациях будет зависеть от старших разрядов числа. Следовательно, пока в процессе эволюции алгоритм не выйдет на значение старшего разряда в окрестности оптимума, операции с младшими разрядами окажутся бесполезными. С другой стороны, когда это произойдет, станут не нужны операции со старшими разрядами – необходимо улучшать точность решения поиском в младших разрядах. Такое "идеальное" поведение не обеспечивает семейство алгоритмов BGA, т.к. эти алгоритмы оперируют битовой строкой целиком, и на первых же эпохах младшие разряды чисел "застывают", принимая случайное значение. В классических ГА разработаны специальные приемы по выходу из этой ситуации. Например, последовательный запуск ансамбля генетических алгоритмов с постепенным сужением пространства поиска.

Есть и другая проблема: при увеличении длины битовой строки необходимо увеличивать и численность популяции.

Математический аппарат непрерывных ГА

Как уже отмечалось, при работе с оптимизационными задачами в непрерывных пространствах вполне естественно представлять гены напрямую вещественными числами. В этом случае хромосома есть вектор вещественных чисел. Их точность будет определяться исключительно разрядной сеткой той ЭВМ, на которой реализуется real-coded алгоритм. Длина хромосомы будет совпадать с длиной

вектора-решения оптимизационной задачи, иначе говоря, *каждый ген будет отвечать за одну переменную*. Генотип объекта становится идентичным его фенотипу.

Вышесказанное определяет список основных преимуществ real-coded алгоритмов:

1. Использование непрерывных генов делает возможным поиск в больших пространствах (даже в неизвестных), что трудно делать в случае двоичных генов, когда увеличение пространства поиска сокращает точность решения при неизменной длине хромосомы.
2. Одной из важных черт непрерывных ГА является их способность к локальной настройке решений.
3. Использование RGA для представления решений удобно, поскольку близко к постановке большинства прикладных задач. Кроме того, отсутствие операций кодирования/декодирования, которые необходимы в BGA, повышает скорость работы алгоритма.

Как известно, появление новых особей в популяции канонического ГА обеспечивают несколько биологических операторов: отбор, скрещивание и мутация. В качестве операторов отбора особей в родительскую пару здесь подходят любые известные из BGA: рулетка, турнирный, случайный. Однако операторы скрещивания и мутации не годятся: в классических реализациях они работают с битовыми строками. Нужны собственные реализации, учитывающие специфику real-coded алгоритмов.

Оператор скрещивания непрерывного ГА, или кроссовер, порождает одного или нескольких потомков от двух хромосом. Собственно говоря, требуется из двух векторов вещественных чисел получить новые векторы по каким-либо законам. Большинство real-coded алгоритмов генерируют новые векторы в окрестности родительских пар. Для начала рассмотрим простые и популярные кроссоверы.

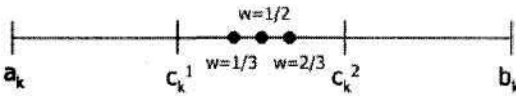
Пусть $C_1=(c_1^1,c_2^1,\dots,c_n^1)$ и $C_2=(c_1^2,c_2^2,\dots,c_n^2)$ – две хромосомы, выбранные оператором селекции для скрещивания. После формулы

для некоторых кроссоверов приводится рисунок – геометрическая интерпретация его работы. Предполагается, что $c_k^1 <= c_k^2$ и $f(C_1) >= f(C_2)$.

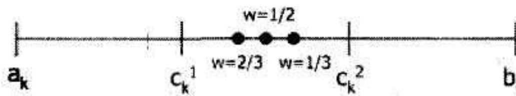
Плоский кроссовер (flat crossover): создается потомок $H=(h_1,\dots,h_k,\dots,h_n)$, h_k, $k=1,\dots, n$ – случайное число из интервала $[c_k^1, c_k^2]$.

Простейший кроссовер (simple crossover): случайным образом выбирается число k из интервала $\{1,2,\dots,n-1\}$ и генерируются два потомка $H1=(c_1^1,c_2^1,\dots,c_k^1,c_{k+1}^2,\dots,c_n^2)$ и $H_2=(c_1^2,c_2^2,\dots,c_k^2,c_{k+1}^1,\dots,c_n^2)$.

Арифметический кроссовер (arithmetical crossover): создаются два потомка $H_1=(h_1^1,\dots,h_n^1)$, $H_2=(h_1^2,\dots,h_n^2)$, где $h_k^1=w*c_k^1+(1-w)*c_k^2$, $h_k^2=w*c_k^2+(1-w)*c_k^1$, $k=1,\dots,n$, w либо константа (равномерный арифметический кроссовер) из интервала [0;1], либо изменяется с увеличением эпох (неравномерный арифметический кроссовер).



Геометрический кроссовер (geometrical crossover): создаются два потомка $H_1=(h_1^1,\dots,h_n^1)$, $H_2=(h_1^2,\dots,h_n^2)$, где $h_k^1= (c_k^1)^w*(c_k^2)^{1-w}$, $(c_k^2)^w*(c_k^1)^{1-w}$, w – случайное число из интервала [0;1].

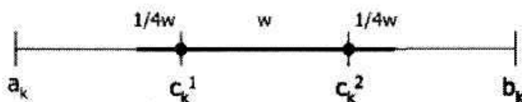


Смешанный кроссовер (blend, BLX-alpha crossover): генерируется один потомок $H=(h_1,\dots,h_k,\dots,h_n)$, где h_k – случайное число из интервала $[c_{min}-I*alpha, c_{max}+I*alpha]$, $c_{min}=min(c_k^1,c_k^2)$, $c_{max}=max(c_k^1,c_k^2)$, $I=c_{max}-c_{min}$. BLX-0.0 кроссовер превращается в плоский.

Линейный кроссовер (linear crossover): создаются три потомка $H_q=(h_1^q,\dots,h_k^q,\dots,h_n^q)$, q=1,2,3, где $h_k^1=0.5*c_k^1+0.5*c_k^2$, $h_k^2=1.5*c_k^1-0.5*c_k^2$, $h_k^3=-0.5*c_k^1+1.5*c_k^2$. На этапе селекции в этом кроссовере отбираются два наиболее сильных потомка.

Дискретный кроссовер (discrete crossover): каждый ген h_k выбирается случайно по равномерному закону из конечного множества $\{c_k^1,c_k^2\}$.

Расширенный линейчатый кроссовер (extended line crossover): ген $h_k=c_k^1+w*(c_k^2-c_k^1)$, w – случайное число из интервала [-0.25;1.25].

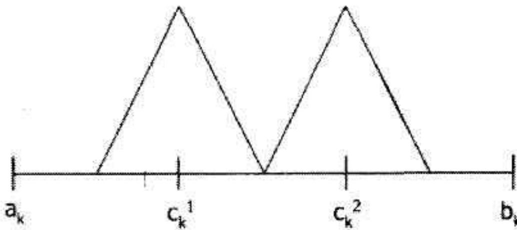


Эвристический кроссовер (Wright's heuristic crossover). Пусть C_1 – один из двух родителей с лучшей приспособленностью. Тогда $h_k=w*(c_k^1-c_k^2)+c_k^1$, w – случайное число из интервала [0;1].

Нечеткий кроссовер (fuzzy recombination, FR-d crossover): создаются два потомка $H_1=(h_1^1,\ldots,h_n^1)$, $H_2=(h_1^1,\ldots,h_n^2)$. Вероятность того, что в i-том гене появится число v_i , задается распределением $p(v_i)\{F(c_k^1),F(c_k^2)\}$, где $F(c_k^1),F(c_k^2)$ – распределения вероятностей треугольной формы (треугольные нечеткие функции принадлежности) со следующими свойствами ($c_k^1=c_k^2$ и $I=|c_k^1–c_k^2|$):

Распределение вероятностей	Минимум	Центр	Максимум
$F(c_k^1)$	$c_k^1–d*I$	c_k^1	c_k^1+d*I
$F(c_k^2)$	$c_k^2–d*I$	c_k^2	c_k^2+d*I

Параметр d определяет степень перекрытия треугольных функций принадлежности, по умолчанию d=0.5.



В качестве оператора мутации наибольшее распространение получили: случайная и неравномерная мутация (random and non-uniform mutation).

При случайной мутации ген, подлежащий изменению, принимает случайное значение из интервала своего изменения. В неравномерной мутации значение гена после оператора мутации рассчитывается по формуле:

$$c_k^* = \begin{cases} c_k + \Delta(t, b_k - c_k), & w = 0 \\ c_k - \Delta(t, b_k - c_k), & w = 1 \end{cases},$$

$$\Delta(t, y) = y\left[1 - r^{\left(1 - \frac{t}{e_max}\right)^b}\right].$$

Сложно сказать, что более эффективно в каждом конкретном случае, но многочисленные исследования доказывают, что непрерывные ГА не менее эффективно, а часто гораздо эффективнее справляются с задачами оптимизации в многомерных пространствах, при этом более просты в реализации из-за отсутствия процедур кодирования и декодирования хромосом.

Рассмотренные кроссоверы исторически были предложены первыми, однако во многих задачах их эффективность оказывается невысокой. Исключение составляет BLX-кроссовер с параметром alpha=0.5 – он превосходит по эффективности большинство простых кроссоверов. Позднее были разработаны улучшенные операторы скрещивания, аналитическая формула которых и эффективность обоснованы теоретически. Рассмотрим подробнее один из таких кроссоверов – SBX.

SBX кроссовер

SBX (англ.: Simulated Binary Crossover) – кроссовер, имитирующий двоичный. Был разработан в 1995 году исследовательской группой под руководством K. Deb’a. Как следует из его названия, этот кроссовер моделирует принципы работы двоичного оператора скрещивания.

SBX кроссовер был получен следующим способом. У двоичного кроссовера было обнаружено важное свойство – среднее значение функции приспособленности оставалось неизменным у родителей и их потомков, полученных путем скрещивания. Затем автором было введено понятие силы поиска кроссовера (search power). Это количественная величина, характеризующая распределение вероятностей появления любого потомка от двух произвольных родителей. Первоначально была рассчитана сила поиска для

одноточечного двоичного кроссовера, а затем был разработан вещественный SBX кроссовер с такой же силой поиска. В нем сила поиска характеризуется распределением вероятностей случайной величины β:

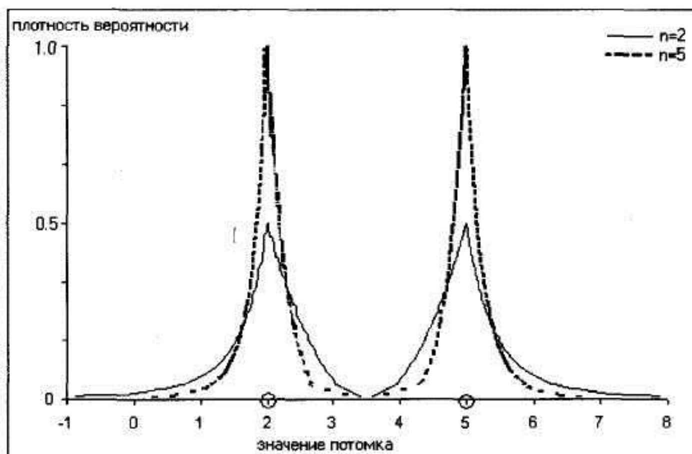
$$P(\beta) = \begin{cases} 0.5(n+1)\beta^{n}, & \beta \le 1 \\ 0.5(n+1)\beta^{-(n+2)}, & \beta > 1. \end{cases}$$

Для генерации потомков используется следующий алгоритм, использующий выражение для P(β). Создаются два потомка $H_k=(h_1^k, \ldots, h_j^k, \ldots, h_n^k)$, k=1,2, где $h_j^1=0.5\cdot[(1-\beta_k)c_j^1+(1-\beta_k)c_j^2]$ - число, полученное по формуле:

$$\beta(u) = \begin{cases} (2u)^{\frac{1}{n+1}}, & u(0,1) \le 0.5 \\ \left(\frac{1}{2(1-u)}\right)^{\frac{1}{n+1}}, & u(0,1) > 0.5. \end{cases}$$

В формуле *u*(0,1) – случайное число, распределенное по равномерному закону, *n* [2,5] – параметр кроссовера.

На рисунке приведена геометрическая интерпретация работы SBX кроссовера при скрещивании двух хромосом, соответствующих вещественным числам 2 и 5.



Из рисунка видно, как параметр n влияет на конечный результат: увеличение n влечет за собой увеличение вероятности появления потомка в окрестности родителя и наоборот.

Эксперименты автора SBX кроссовера показали, что он во многих случаях эффективнее BLX, хотя, очевидно, что не существует ни одного кроссовера, эффективного во всех случаях. Исследования показывают, что использование нескольких различных операторов кроссовера позволяет уменьшить вероятность преждевременной сходимости, т.е. улучшить эффективность алгоритма оптимизации в целом. Для этого могут использоваться специальные стратегии, изменяющие вероятность применения отдельного эволюционного оператора в зависимости от его «успешности», или использование гибридных кроссоверов, которых в настоящее время насчитывается несколько десятков. В любом случае, если перед Вами стоит задача оптимизации в непрерывных пространствах, и Вы планируете применить эволюционные техники, то следует сделать выбор в пользу непрерывного генетического алгоритма.

5. Эволюционное моделирование

5.1. Эволюционные алгоритмы

Генетические алгоритмы *(genetic algorithms)* совместно с эволюционной стратегией и эволюционным программированием представляют три главных направления развития так называемого эволюционного моделирования *(simulated evolution).*
Несмотря на то, что каждый из этих методов возник независимо от других, они характеризуются рядом важных общих свойств. Для любого из них формируется исходная популяция особей, которая в последующем подвергается селекции и воздействию различных генетических операторов (чаще всего скрещиванию и/или мутации), что позволяет находить более хорошие решения.
Эволюционные стратегии - это алгоритмы, созданные в Германии в качестве методов решения оптимизационных задач и основанные на принципах природной эволюции. Эволюционное программирование представляет собой подход, предложенный американскими учеными вначале в рамках теории конечных автоматов и обобщенный позднее для приложений к проблемам оптимизации. Оба направления возникли в шестидесятых годах XX века.
Сконцентрируем внимание на важнейших сходствах и различиях между эволюционными стратегиями и генетическими алгоритмами. Очевидно, что главное сходство заключается в том, что оба метода используют популяции потенциальных решений и реализуют принцип селекции и преобразования наиболее приспособленных особей. Однако обсуждаемые подходы сильно отличаются друг от друга. Первое различие заключается в способе представления особей. Эволюционные стратегии оперируют векторами действительных чисел, тогда как генетические алгоритмы - двоичными векторами.
Второе различие между эволюционными стратегиями и генетическими алгоритмами кроется в организации процесса селекции. При реализации эволюционной стратегии формируется промежуточная популяция, состоящая из всех родителей и некоторого количества потомков, созданных в результате применения генетических операторов. С помощью селекции размер этой промежуточной популяции уменьшается до величины родительской популяции за счет исключения наименее приспособленных особей. Сформированная таким образом популяция образует очередное поколение. Напротив, в генети-

ческих алгоритмах предполагается, что в результате селекции из популяции родителей выбирается количество особей, равное размерности исходной популяции, при этом некоторые (наиболее приспособленные) особи могут выбираться многократно. В то же время, менее приспособленные особи также имеют возможность оказаться в новой популяции. Однако шансы их выбора пропорциональны величине приспособленности особей. Независимо от применяемого в генетическом алгоритме метода селекции (например, рулетки или рангового) более приспособленные особи могут выбираться многократно. При реализации эволюционных стратегий особи выбираются без повторений. В эволюционных стратегиях применяется детерминированная процедура селекции, тогда как в генетических алгоритмах она имеет случайный характер.

Третье различие между эволюционными стратегиями и генетическими алгоритмами касается последовательности выполнения процедур селекции и рекомбинации (т.е. изменения генов в результате применения генетических операторов). При реализации эволюционных стратегий вначале производится рекомбинация, а потом селекция. В случае выполнения генетических алгоритмов эта последовательность инвертируется. При осуществлении применения эволюционных стратегий потомок образуется в результате скрещивания двух родителей и мутации. Формируемая таким образом промежуточная популяция, состоящая из всех родителей и полученных от них потомков, в дальнейшем подвергается селекции, которая уменьшает размер этой популяции до размера исходной популяции. При выполнении генетических алгоритмов вначале производится селекция, приводящая к образованию переходной популяции, после чего генетические операторы скрещивания и мутации применяются к особям (выбираемым с вероятностью скрещивания) и к генам (выбираемым с вероятностью мутации).

Следующее различие между эволюционными стратегиями и генетическими алгоритмами заключается в том, что параметры генетических алгоритмов (такие, как вероятности скрещивания и мутации) остаются постоянными на протяжении всего процесса эволюции, тогда как при реализации эволюционных стратегий эти параметры подвергаются непрерывным изменениям (так называемая самоадаптация параметров).

Еще одно различие между эволюционными стратегиями и генетическими алгоритмами касается трактовки ограничений, налагаемых на решаемые оптимизационные задачи. Если при

реализации эволюционных стратегий на некоторой итерации потомок не удовлетворяет всем ограничениям, то он отвергается и включается в новую популяцию. Если таких потомков оказывается много, то эволюционная стратегия запускает процесс адаптации параметров, например, путем увеличения вероятности скрещивания. В генетических алгоритмах такие параметры не изменяются. В них может применяться штрафная функция для тех особей, которые не удовлетворяют наложенным ограничениям, однако эта технология обладает многими недостатками.

По мере развития эволюционных стратегий и генетических алгоритмов в течение последних лет существенные различия между ними постепенно уменьшаются. Например, в настоящее время при реализации генетических алгоритмов для решения оптимизационных задач все чаще применяется представление хромосом действительными числами и различные модификации «генетических» операторов, что имеет целью повысить эффективность этих алгоритмов. Подобные методы, значительно отличающиеся от классического генетического алгоритма, по традиции сохраняют прежнее название, хотя более корректно было бы называть их «эволюционными алгоритмами». Проблему терминологии мы будем обсуждать несколько позднее.

Эволюционное программирование, также как и эволюционные стратегии, делает основной упор на адаптацию и разнообразие способов передачи свойств от родителя к потомкам в следующих поколениях. Познакомимся со стандартным алгоритмом эволюционного программирования.

Исходная популяция решений выбирается случайным образом. В задачах оптимизации значений действительных чисел (примером которых может служить обучение нейронных сетей) особь (хромосома) представляется цепью значений действительных чисел. Эта популяция оценивается относительно заданной функции (функции приспособленности). Потомки образуются от входящих в эту популяцию родителей в результате случайной мутации. Селекция основана на вероятностном выборе (турнирный метод), при котором каждое решение соперничает с хромосомами, случайным образом выбираемыми из популяции. Решения-победители (оказавшиеся наилучшими) становятся родителями для следующего поколения. Описанная процедура повторяется так долго, пока не будет найдено искомое решение либо не будет исчерпан лимит машинного времени.

Эволюционное программирование применяется для оптимизации функционирования нейронных сетей. Также как и другие эволю-

ционные методы, оно не требует градиентной информации и поэтому может использоваться для решения задач, в которых эта информация недоступна, либо для ее получения требуются значительные объемы вычислений. Одними из первых приложений эволюционного программирования считаются задачи теории искусственного интеллекта, а самые ранние работы касались теории конечных автоматов.

Наблюдается большое сходство между эволюционными стратегиями и эволюционными программированием в их приложениях к задачам оптимизации непрерывных функций с действительными значениями. Некоторые исследователи утверждают, что эти процедуры, в сущности, одинаковы, хотя они и развивались независимо друг от друга. Действительно, оба метода похожи на генетические алгоритмы. Принципиальное различие между ними заключается в том, что эволюционное программирование не связано с конкретной формой представления особей, поскольку оператор мутации не требует применения какого-либо специального способа кодирования.

Первый контакт между научными коллективами, развивавшими эволюционные стратегии и эволюционное программирование, состоялся в начале 1992 г., непосредственно перед первой международной конференцией, посвященной эволюционному программированию. Эти методы развивались независимо на протяжении 30 лет. Несмотря на выделенные различия, они имеют много принципиально сходных свойств.

Все три представленных метода, т.е. генетические алгоритмы, эволюционные стратегии и эволюционное программирование объединяются под общим названием эволюционные алгоритмы *(evolutionary algorithms).* Также применяется термин эволюционные методы *(evolutionary methods).*

Эволюционными алгоритмами называются и другие методы, реализующие эволюционный подход, в частности, генетическое программирование *(genetic programming)*, представляющее собой модификацию генетического алгоритма с учетом возможностей компьютерных программ. При использовании этого метода популяция состоит из закодированных соответствующим образом программ, которые подвергаются воздействию генетических операторов скрещивания и мутации, для нахождения оптимального решения, которым считается программа, наилучшим образом решающая поставленную задачу. Программы оцениваются относительно определенной специальным образом функции приспособленности. Интересной представляется

модификация эволюционных алгоритмов для решения оптимизационных задач методом так называемой мягкой селекции, которая предложена Р. Галаром.

Для обозначения разнообразных алгоритмов, основанных на эволюционном подходе, также применяется понятие эволюционных программ *(evolution programs).* Этот термин объединяет как генетические алгоритмы, эволюционные стратегии и эволюционное программирование, так и генетическое программирование, а также другие аналогичные методы.

Эволюционные программы можно считать обобщением генетических алгоритмов. Классический генетический алгоритм выполняется при фиксированной длине двоичных последовательностей и в нем применяются операторы скрещивания и мутации. Эволюционные программы обрабатывают более сложные структуры (не только двоичные коды) и могут выполнять иные «генетические» операции. Например, эволюционные стратегии могут трактоваться в качестве эволюционных программ, в которых хромосомы представляются вещественными (не двоичными) числами, а мутация используется как единственная генетическая операция.

Структуру эволюционной программы довольно точно отображает блок-схема, приведенная на рис. 3.12. Она совпадает со структурой генетического алгоритма, поскольку идеи эволюционной программы целиком заимствованы из теории генетических алгоритмов. Различия имеют глубинный характер, они касаются способов представления хромосом и реализации генетических операторов. Эволюционные программы допускают большее разнообразие структур данных, поскольку возможно не только двоичное кодирование хромосом, а также предоставляют расширенный набор генетических операторов.

Согласно 3. Михалевичу, эволюционная программа - это вероятностный алгоритм, применяемый на *k-й* итерации к популяции особей

$$P(k) = \{ x^k_1.....x^k_n\}.$$

Каждая особь представляет потенциальное решение задачи, которое в произвольной эволюционной программе может отображаться некоторой (в том числе и достаточно сложной) структурой данных *D*. Любое решение x_i^k оценивается по значению его «приспособленности». Далее в процессе селекции на (k+1)-й итерации из наиболее приспособленных особей формируется очередная популяция. Некоторые особи этой новой популяции трансформируются с помощью «генетических операторов», что позволяет получать новые решения.

Существуют преобразования μ_i (типа мутации), которые изменяют конкретные хромосомы (μ_i: D→D), а также трансформации более высокого порядка γ_i (типа скрещивания), создающие новые особи путем комбинирования фрагментов нескольких (двух или более) хромосом, т.е. γ_i: D × ... × D→D. От эволюционной программы ожидается, что после смены некоторого количества поколений наилучшая особь будет представлять решение, близкое к оптимальному. Структура эволюционной программы может быть представлена в виде псевдокода так, как это изображено на рис. 5.1 (рекомендуется сравнить ее с рис. 3.12).

```
procedure эволюционная_программа
begin
 k = 0
 инициализация популяции P(k)
 оценивание приспособленности особей из P(k)
 while (not условие завершения) do
 begin
 k = k + 1
 селекция особей из P(k – 1) в P(k)
 применение генетических операторов
 оценивание приспособленности особей из P(k)
 end
end
```

Рис. 5.1. Представление эволюционной программы в виде псевдокода.

Рассмотрим обобщенный пример эволюционной программы. Допустим, что ищется граф, который удовлетворяет определенным ограничениям (например, производится поиск топологии коммуникационной сети, оптимальной по конкретным критериям, например, по стоимости передачи и т.п.). Каждая особь в эволюционной программе представляет одно из потенциальных решений, т.е. в данном случае некоторый граф. Исходная популяция графов P(0), формируемая случайным образом либо создаваемая при реализации какого-либо эвристического процесса, считается отправной точкой ($k = 0$) эволюционной программы. Функция приспособленности, которая обычно задается, связана с системой ограничений решаемой задачи. Эта функция определяет «приспособленность» каждого графа путем

выявления «лучших» и «худших» особей. Можно предложить несколько различных операторов мутации, предназначенных для трансформации отдельных графов, и несколько операторов скрещивания, которые будут создавать новый граф в результате рекомбинации структур двух или более графов. Очень часто такие операторы обусловливаются характером решаемой задачи. Например, если ищется граф типа «дерево», то можно предложить оператор мутации, который удаляет ветвь из одного графа и добавляет новую ветвь, объединяющую два отдельных подграфа. Другие возможности заключаются в проектировании мутации независимо от семантики задачи, но с включением в функцию приспособленности дополнительных ограничений - «штрафов» для тех графов, которые не являются деревьями.

Принципиальную разницу между классическим генетическим алгоритмом и эволюционной программой, т.е. эволюционным алгоритмом в широком смысле, иллюстрируют рис. 5.2 и 5.3.

Классический генетический алгоритм, который оперирует двоичными последовательностями, требует представить решаемую задачу в строго определенном виде (соответствие между потенциальными решениями и двоичными кодами, декодирование и т.п.). Сделать это не всегда просто.

Эволюционные программы могут оставить постановку задачи в неизменном виде за счет модификации хромосом, представляющих потенциальные решения (с использованием «естественных» структур данных), и применения соответствующих «генетических» операторов

Другими словами, для решения нетривиальной задачи можно либо преобразовать ее к виду, требуемому для использования генетического алгоритма (рис. 5.2), либо модифицировать генетический алгоритм так, чтобы он удовлетворял задаче (рис. 5.3).

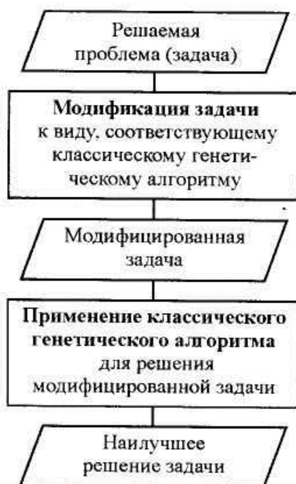


Рис. 5.2. Решение задачи с помощью классического генетического алгоритма.

Рис. 5.3. Решение задачи с помощью эволюционного алгоритма (эволюционной программы)

При реализации первого подхода применяется классический генетический алгоритм, а при реализации второго подхода - эволюционная программа Таким образом, модифицированные генетические алгоритмы можно в общем случае называть эволюционными программами. Однако чаще всего встречается термин эволюционные алгоритмы. Эволюционные программы также могут рассматриваться как эволюционные алгоритмы, подготовленные программистом для выполнения на компьютере. Основная задача программиста заключается при этом в выборе соответствующих структур данных и «генетических» операторов. Именно такая трактовка понятия эволюционная программа представляется наиболее обоснованной.

Все понятия, применяемые в настоящем разделе и относящиеся главным образом к методам, основанным на эволюционном подходе, можно сопоставить главному направлению исследований - компьютерному моделированию эволюционных процессов. Эта область информатики называется *Evolutionary Computation,* что можно перевести как эволюционные вычисления.

К эволюционным алгоритмам также применяется понятие технология эволюционных вычислений. Можно добавить, что название генетические алгоритмы используется как в узком смысле, т.е. для обозначения классических генетических алгоритмов и их несущественных модификаций, так и в широком смысле - подразумевая любые эволюционные алгоритмы, значительно отличающиеся от «классики».

В последнее время появилось множества новых методов, основанных на эволюционном моделировании, но использующих базовые технологии - главным образом классический генетический алгоритм, эволюционные стратегии и эволюционное программирование.

5.2. Приложения эволюционных алгоритмов

Большинство приложений эволюционных алгоритмов, и особенно генетических алгоритмов, касается оптимизационных задач. Простейшими из них можно назвать задачи, представленные в примерах. В каждом из них оптимизируется целевая функция, заданная конкретной формулой, и используется характерное для основного генетического алгоритма двоичное кодирование хромосом.

Как уже упоминалось, последующая модификация классического генетического алгоритма заключалась в представлении хромосом

действительными числами. О таком способе кодирования говорилось и в разд. 3.18. Одной из наиболее известных компьютерных программ, предназначенных для решения задач при помощи генетического алгоритма с кодированием действительными числами *(real coding),* считается программа **Evolver**. В этой программе применяется алгоритм с частичной заменой популяции *(steady-state),* при которой в каждый момент времени заменяется только одна особь. Селекция основана на ранговом методе *(rank-based).* Если говорить о так называемых генетических операторах, то в программе **Evolver** применяются два различных оператора скрещивания и два различных оператора мутации - отдельно для оптимизационных и для комбинаторных задач.
Программа **Evolver** взаимодействует с табличным процессором **Excel,** в котором решаемая задача описывается в соответствующих ячейках таблицы путем задания ее параметров (переменных) и формулы функции приспособленности.

5.2.1. Примеры оптимизации функции с помощью программы Evolver

Прежде чем перейти к примерам, дадим характеристику генетическим операторам, используемым программой **Evolver** в задачах оптимизации функции. .
Скрещивание. Это равномерное скрещивание *(uniform crossover),* определенное аналогично примененному в п. 3.16.3, но для хромосом, состоящих из генов с действительными аллелями.
Скрещивание производится в соответствии с так называемым *показателем скрещивания (crossover rate),* определяющим, какой процент генов потомок унаследует от каждого родителя. В программе **Evolver** показатель скрещивания вводится пользователем и представляет собой число из интервала от 0,01 до 1,0. Например, значение показателя скрещивания 0,8 означает, что потомок получит около 80 % генов со значениями (аллелями) такими же, как у первого родителя, а оставшееся количество (порядка 20 %) - унаследует от второго родителя. Если показатель скрещивания равен 1, то никакого скрещивания практически не происходит, а образуются только так называемые клоны, т.е. хромосомы, идентичные родителям. По умолчанию в программе **Evolver** применяется значение показателя скрещивания, равное 0,5, что означает наследование примерно одинакового количества генов от каждого родителя.

Мутация. Каждый ген в хромосоме представляет один параметр задачи. Следовательно, аллели соответствуют фенотипам, т.е. значениям конкретных переменных. Для каждого гена случайным образом выбирается число из интервала от 0 до 1, которое сравнивается с так называемым *показателем мутации (mutation rate).* Если разыгранное число меньше введенного значения показателя или равно ему, то выполняется мутация данного гена. Эта операция заключается в замене значения гена другим, случайно выбранным числом из области допустимых значений параметра, соответствующего мутирующему гену.

Вводимое пользователем значение показателя мутации представляет собой число из интервала от 0,0 до 1,0. Чем больше это значение, тем большее количество генов подвергается мутации. Если показатель мутации равен 1, то мутации подвергаются 100 % генов, выбираемых случайным образом. С учетом того, что в программе **Evolver** мутация всегда производится после скрещивания, задание показателя мутации равным 1 означает, что в этом случае эффект скрещивания не имеет никакого значения. Очевидно, что если показатель мутации равен 0, то мутация вообще не производится. Как правило, значение показателя мутации задается в пределах от 0,06 до 0,2. Отметим, что определяемый таким образом показатель мутации представляет собой аналог вероятности мутации p_m, описанной в разд. 3.4. В то же время используемый в программе **Evolver** показатель скрещивания имеет смысл, отличающийся от введенной в разд. 3.4 вероятности скрещивания p_c. Значения как показателя мутации, так и показателя скрещивания можно изменять в процессе выполнения программы **Evolver.**

В классическом генетическом алгоритме на каждой итерации обновляется вся популяция, что соответствует формированию очередного поколения. Применительно к программе **Evolver** можно говорить о последовательных «тактах» *(trials)* ее выполнения, причем на каждом такте изменяется только одна особь. Если популяция насчитывает *N* хромосом (особей, которые в описании программы называются организмами), то *N* таких «тактов» составляют одну итерацию классического генетического алгоритма. Поэтому для того, чтобы при заданном количестве «тактов» выполнения программы **Evolver** найти соответствующее ему количество поколений (в терминологии классического генетического алгоритма), необходимо значение *N* разделить на количество особей в популяции. В представляемых далее примерах «такты» будут обозначаться символом *t.* Пример 5.1 демонстрирует

применение программы **Evolver** для оптимизации функции трех переменных.

Пример 5.1

С помощью программы **Evolver** найти минимум функции

$f(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2$

для целочисленных x_1, x_2, x_3 в интервале [-5, 5].

Вычисления производились для популяции размерностью $N = 6$. Использовалось значение показателя скрещивания равное 0,9. Это означает, что в результате скрещивания потомок наследует приблизительно (с округлением до целого) 90 % генов от первого родителя и остальные гены (около 10 %) - от второго родителя. Заметим, что для показателя скрещивания равного 1 будет наблюдаться такой же эффект, что и при отсутствии скрещивания, поскольку потомок будет наследовать все гены только от первого родителя. По этой причине при выбранном значении данного показателя, равном 0,9, скрещивание либо не будет проводиться вообще (точнее говоря, потомок окажется абсолютно идентичным своему первому родителю), либо скрещивание будет выполнено, и потомок унаследует большую часть генов от первого родителя. Для мутации выбрана величина показателя мутации равная 0,1, которая указывает, что значения 10% генов (с округлением до целого) должны подвергнуться случайным изменениям. С помощью программы **Evolver** сформирована следующая исходная популяция:

[-1 3 -3]
[-1 -5 2]
[5 3 -4]
[2 -4 -0]
[5 4 2]
[4 -3 5]

После шести «тактов», т.е. для $t = 6$, что соответствует одной итерации классического генетического алгоритма, получена популяция особей, упорядоченных по убыванию функции приспособленности так, как это показано на рис. 5.4.

№ п/п	$f(x_1, x_2, x_3)$	x_1	x_2	x_3
1	50	5	3	–4
2	50	5	3	–4
3	45	5	4	2
4	30	–1	–5	2
5	20	2	–4	0
6	19	–1	3	–3

Рис. 5.4. Популяция особей для t = 6 (пример 5.1).

Отметим, что в популяцию не входит особь [4 -3 5] со значением функции приспособленности, равным 50. Она была исключена как «наихудшая» особь (с наибольшим значением функции приспособленности), и на ее место введена копия особи [5 3 -4]. Эта особь также оказывается «наихудшей», поэтому она будет исключена на следующем «такте» (t = 7). Популяции для t = 7 и
t = 8 представлены на рис. 5.5 и 5.6.

№ п/п	$f(x_1, x_2, x_3)$	x_1	x_2	x_3
1	50	5	4	2
2	50	5	3	–4
3	45	5	4	2
4	30	–1	–5	2
5	20	2	–4	0
6	19	–1	3	–3

Рис. 5.5. Популяция особей для t = 7 (пример 5.1).

№ п/п	$f(x_1, x_2, x_3)$	x_1	x_2	x_3
1	[50]	0	–4	2
2	50	5	4	2
3	45	5	4	2
4	30	–1	–5	2
5	20	2	–4	0
6	19	–1	3	–3

Рис. 5.6. Популяция особей для t = 8 (пример 5.1).

На последнем месте (№ п/п = 6) размещается «наилучшая к данному моменту» особь, имеющая наименьшее значение функции приспособленности. На первом месте (№ п/п = 1) показана вновь введенная в популяцию особь. На рис. 5.5 это копия особи [5 4 2] из предыдущей популяции (t = 6), а на рис. 5.6 - новая особь [0 -4 2], полученная в результате скрещивания с последующей мутацией от пары особей из популяции для t = 7. Выделенное прямоугольником значение функции приспособленности в первой строке таблицы относится к особи, исключаемой из популяции.
На рис. 5.7 и 5.8 представлены популяции для t =17 и t =18.

№ п/п	$f(x_1, x_2, x_3)$	x_1	x_2	x_3
1	[25]	0	–4	0
2	20	2	–4	0
3	20	0	–4	2
4	20	2	–4	0
5	20	2	–4	0
6	19	–1	3	–3

Рис. 5.7. Популяция особей для t = 17 (пример 5.1).

№ п/п	$f(x_1, x_2, x_3)$	x_1	x_2	x_3
1	20	2	–3	0
2	20	0	–4	2
3	20	2	–4	0
4	20	2	–4	0
5	19	–1	3	–3
6	16	0	–4	0

Рис. 5.8. Популяция особей для *t* = 18 (пример 5.1).

Заметим, что *t* = 18 соответствует трем итерациям классического генетического алгоритма. Новая особь в популяции на рис. 4.87 получена скрещиванием пары особей из предыдущей популяции (для *t*=16), а новая особь в популяции на рис. 3.88 сформирована в результате мутации среднего гена особи [2 -4 0] из предыдущей популяции (для f=17).

На рис. 5.9 показана популяция для *t* = 41. Обратим внимание, что *t*=42 соответствует семи итерациям (поколениям) классического генетического алгоритма. «Наилучшее к данному моменту» решение - это [-1 2 0] со значением функции приспособленности, равным 5.

№ п/п	$f(x_1, x_2, x_3)$	x_1	x_2	x_3
1	10	0	–3	0
2	10	–1	3	0
3	10	–1	3	0
4	10	–1	3	0
5	9	0	–3	0
6	5	–1	2	0

Рис. 5.9. Популяция особей для *t* - 41 (пример 4.1).

Получение «наилучшего» решения, равного [0 0 0], с нулевым значением функции приспособленности возможно в случае, если при

дальнейшем выполнении алгоритма произойдет мутация второго гена (замена -3, 3 или 2 на 0).
Следующий пример представляет собой обобщение предыдущего примера за счет увеличения количества переменных и расширения пространства поиска.

Пример 5.2

С помощью программы **Evolver** найти минимум функции

$f(x_1, x_2, ..., x_6) = x_1^2 + x_2^2 + \ldots + x_6^2$

для целочисленных $x_1, x_2, ..., x_6$ в интервале [-10, 10].

Вычисления производились для популяции размерностью N=50. Использовались принятые по умолчанию в программе **Evolver** значения показателея скрещивания = 0,5 и показателя мутации =0,06. На рис. 5.10 представлена модель решения рассматриваемого примера в табличном процессоре **Excel,** содержащая начальные значения переменных $x_1, x_2, ..., x_6$.

	Минимизация функции F(X1, X2, ..., X6) = X1*X1+ X2*X2+ ... + X6*X6						
	для X1, X2, ..., X6 из интервала [-10,10]						
	X1 =	10					
	X2 =	-8					
	X3 =	9					
	X4 =	7					
	X5 =	-6					
	X6 =	-9					
F(X1, X2, ..., X6) =		411					

Рис. 5.10. Начальные значения переменных и значение функции для примера 5.2, представленные в табличном процессоре **Excel**.

В исходную популяцию включены 50 особей, первая из которых содержит гены со значениями, показаными на рис. 3.90, а остальные хромосомы сгенерированы случайным образом (гены имеют действительные значения из интервала от -10 до 10). Длина каждой хромосомы равна 6 и совпадает с количеством переменных $x_1, x_2, ..., x_6$. Заметно, что приведенные на рис. 5.10 начальные значения переменных $x_1, x_2, ..., x_6$ весьма далеки от оптимального решения, которым для рассматриваемой задачи (также как и в примере 5.1) будет хромосома с нулевыми значениями генов, т.е. [000000]. Поэтому не вызывает удивления то, что хромосома с аллелями, показанными на рис. 5.10,

очень быстро исключается из популяции. На рис. 5.11 изображены графики изменения «наилучшего» (нижняя кривая) и среднего (верхняя кривая) значения функции приспособленности с течением времени (точнее, с увеличением количества «тактов»).

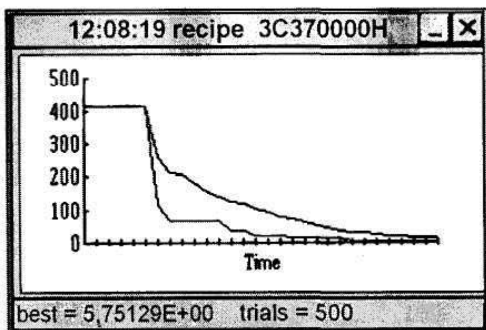


Рис. 5.11. Изменение «наилучшего» (вверху) и среднего (внизу) значения функции приспособленности для примера 5.2.

На этом рисунке один момент времени соответствует 20 «тактам». Для t=500 (через 500 «тактов») наилучшее значение функции приспособленности было равно 5,75129. На рис. 5.12 в виде столбчатой диаграммы показаны значения функции приспособленности всех особей популяции для t = 500.

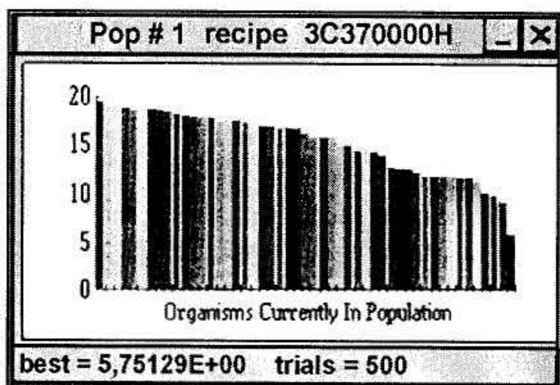


Рис. 5.12. Столбчатая диаграмма значений функции приспособленности особей в популяции при t = 500 для примера 5.2.

Рис. 5.13 представляет значения переменных x_1, x_2, ..., x_6 и соответствующее им значение минимизируемой функции, полученное после 7975 «тактов» выполнения программы **Evolver.**

	Минимизация функции F(X1, X2, ..., X6) = X1*X1+ X2*X2+ ... + X6*X6	
	для X1, X2, ..., X6 из интервала [-10,10]	
	X1 =	-0,021
	X2 =	0,024
	X3 =	-0,096
	X4 =	0,005
	X5 =	0,009
	X6 =	-0,055
F(X1, X2, ..., X6) =		0,013364

Рис. 5.13. Значения переменных и функции для примера 5.2 при $t = 7975$.

На рис. 5.14 и 5.15 приведены аналогичные результаты, но после 10623 и 11953 «тактов».

	Минимизация функции F(X1, X2, ..., X6) = X1*X1+ X2*X2+ ... + X6*X6	
	для X1, X2, ..., X6 из интервала [-10,10]	
	X1 =	-0,021
	X2 =	0,024
	X3 =	-0,011
	X4 =	0,005
	X5 =	0,009
	X6 =	-0,055
F(X1, X2, ..., X6) =		0,004269

Рис. 5.14. Значения переменных и функции для примера 5.2 при $t = 10623$.

Минимизация функции F(X1, X2, ..., X6) = X1*X1+ X2*X2+ ... + X6*X6
для X1, X2, ..., X6 из интервала [-10,10]

X1 =	-0,02
X2 =	0,001
X3 =	-0,011
X4 =	0,002
X5 =	0,009
X6 =	0,001
F(X1, X2, ..., X6) =	0,000608

Рис. 5.15. Значения переменных и функции для примера 5.2 при t = 11953.

Их сопоставление показывает, как получаемые решения приближаются к оптимальному. Рис. 5.16 демонстрирует «наилучшее решение», выработанное после примерно 15000 «тактов» выполнения программы **Evolver.** Последующая работа программы уже не изменяет полученный результат. Итоговая популяция содержит 50 особей со значениями $x_1, x_2, ..., x_6$, показанными на рис. 5.16.

Минимизация функции F(X1, X2, ..., X6) = X1*X1+ X2*X2+ ... + X6*X6
для X1, X2, ..., X6 из интервала [-10,10]

X1 =	-0,02
X2 =	0,001
X3 =	-0,011
X4 =	0
X5 =	0,009
X6 =	0,001
F(X1, X2, ..., X6) =	0,000604

Рис. 5.16. Значения переменных и функции для примера 5.2 при t = 15950.

Повторное возобновление этого алгоритма при тех же начальных данных (рис. 5.10), но других случайных значениях остальных членов исходной популяции дает совершенно иной набор «наилучших»

значений переменных x_1, x_2, ..., x_6, которые тоже близки к оптимальным.

Следующие примеры связаны с минимизацией более сложной функции 9 переменных, а именно - функции погрешности нейронной сети, реализующей логическую систему XOR.

Пример 5.3

С помощью программы **Evolver** найти оптимальный набор весов нейронной сети, изображенной на рис. 3.11 (пример 3.3), если значения весов лежат в интервале от -5 до 5.

Решение этой задачи заключается в подборе такого комплекса значений весов w_{11}, w_{12}, w_{21}, w_{22}, w_{31}, w_{32}, а также w_{10}, w_{20}, w_{30}, который минимизирует значение погрешности Q, заданной в примере 3.3. Это средняя сумма квадратов разностей между ожидаемым и расчетным значением на выходе системы XOR для каждой из четырех пар входов, указанных в табл. 2.1.

u_1	u_2	d = XOR(u_1,u_2)
+1	+1	–1
+1	–1	+1
–1	+1	+1
–1	–1	–1

Задача сводится к минимизации функции

$$Q = \frac{1}{4}\sum_{i=1}^{4} \varepsilon_i^{\,2}$$

относительно определенных выше девяти весов, где $\varepsilon_i = d_i - y_i$, для $i = 1, 2, 3, 4$, а y_i, рассчитывается по формуле

$y_i = 1/(1 + \exp((-\beta)(w_{31}(1/ (1 + exp((-\beta)(w_{11}u_{1,i} + w_{12}u_{2,i} + w_{10})))) +$

$+ w_{32}(1 / (1 + \exp((-\beta)(w_{21}u_{1,i} + w_{22}u_{2i} + w_{20})))) + w_{30})))$.

Примем, *что* $\beta = 1$.

Задача моделировалась в табличном процессоре **Excel**, а оптимизация выполнялась при помощи генетического алгоритма программы **Evolver**. Для популяции с размерностью, равной 50, при значениях показателя скрещивания равного 0,5 и показателя мутации равного 0,06 после примерно 30000 «тактов» получено «наилучшее решение» в виде набора весов, показанного на рис. 5.17. Минимальное значение абсолютной погрешности Q для этих весов составляет 0,015171.

входы		заданное выходное значение	расчетное выходное значение		веса
u1	u2	d	y	w11 =	-4,9955
0	0	0	0,123677	w12 =	-4,9985
0	1	1	0,878417	w21 =	4,994
1	0	1	0,876504	w22 =	-4,988
1	1	0	0,123919	w31 =	4,9975
				w32 =	4,9865
				w10 =	-2,8905
				w20 =	-2,9025
				w30 =	-2,4805
		Минимальная абсолютная погрешность	Q = 0,015171		
		Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений			

Рис. 5.17. «Наилучший» набор весов из интервала [-5, 5] для нейронной сети, реализующей систему XOR (пример 5.3).

Необходимо отметить, что полученное «наилучшее решение» близко набору весов $w_{11} = -5$, $w_{12} = 5$, $w_{21} = 5$, $w_{22} = -5$, $w_{31} = 5$, $w_{32} = 5$, $w_{10} = -3$, $w_{20} = -3$, $w_{30} = -2{,}5$, для которого Q = 0,015167. Нейронная сеть (см. рис. 3.11) с такими весами демонстрирует симметрию, характерную для системы XOR. Очевидно, что найден не единственный оптимальный набор весов для сети этого типа. При повторном запуске программы с той же самой размерностью популяции и такими же показателями скрещивания и мутации, скорее всего, будет найдено «наилучшее решение», содержащее иной «оптимальный» набор весов. На рис. 5.18 приведен пример альтернативного «наилучшего» набора весов, полученного при повторном выполнении генетического алгоритма программы **Evolver** и проведении вычислений на протяжении 6000 «тактов».

входы		заданное выходное значение	расчетное выходное значение		веса
u1	u2	d	y	w11 =	3,333
0	0	0	0,149766	w12 =	3,3495
0	1	1	0,83362	w21 =	-4,979
1	0	1	0,83522	w22 =	-4,977
1	1	0	0,171337	w31 =	-4,998
				w32 =	-4,989
				w10 =	-4,99
				w20 =	1,9165
				w30 =	2,6465
		Минимальная абсолютная погрешность	Q = 0,026655		
		Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений			

Рис. 5.18. Другой «наилучший» набор весов для примера 5.3.

При известных допустимых комбинациях весов для системы XOR, легко предсказать оптимальный набор весов, к которому стремится решение при продолжении выполнения этого алгоритма.

Следующие примеры относятся к той же задаче, однако при расширенном диапазоне изменения весов.

Пример 5.4

С помощью программы Evolver найти оптимальный набор весов нейронной сети, изображенной на рис. 3.11 (пример 3.3), если значения весов лежат в интервале от -10 до 10.

При начальных значениях, совпадающих с приведенными в примере 3.23, было проведено три сеанса вычислений. Через 30000 «тактов» в первом, втором и третьем сеансах получены «наилучшие» решения, показанные на рис. 5.19, 5.20 и 5.22 соответственно.

входы		заданное выходное значение	расчетное выходное значение
u1	u2	d	y
0	0	0	7,67E-03
0	1	1	0,992914
1	0	1	0,99262
1	1	0	7,66E-03

	веса
w11 =	9,997
w12 =	-9,99
w21 =	9,982
w22 =	-9,993
w31 =	-9,983
w32 =	9,972
w10 =	5,139
w20 =	-5,462
w30 =	5,02

Минимальная абсолютная погрешность Q = 5,56E-05

Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений

Рис. 5.19. «Наилучший» набор весов из интервала [-10, 10] для нейронной сети, реализующей систему XOR (пример 5.4).

входы		заданное выходное значение	расчетное выходное значение
u1	u2	d	y
0	0	0	0,007916
0	1	1	0,992505
1	0	1	0,992687
1	1	0	0,007921

	веса
w11 =	-9,939
w12 =	9,972
w21 =	9,975
w22 =	-9,999
w31 =	9,904
w32 =	9,935
w10 =	-5,314
w20 =	-5,379
w30 =	-4,925

Минимальная абсолютная погрешность Q = 5,88E-05

Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений

Рис. 5.20. Другой «наилучший» набор весов для примера 5.4.

На рис. 5.21 изображены графики изменения «наилучшего» (нижняя кривая) и среднего (верхняя кривая) значения функции приспособленности для этого примера после 505 «тактов».

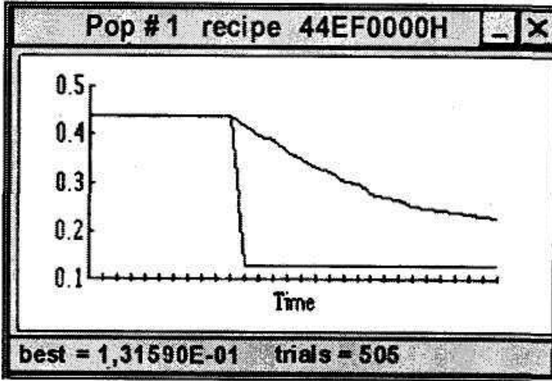


Рис. 5.21. Изменение «наилучшего» и среднего значения функции приспособленности для примера 5.4.

входы		заданное выходное значение	расчетное выходное значение			веса
					w11 =	-6,778
					w12 =	-6,788
u1	u2	d	y		w21 =	-9,86
					w22 =	-9,937
0	0	0	9,65E-03		w31 =	9,995
0	1	1	0,991612		w32 =	-9,989
1	0	1	0,991623		w10 =	9,941
1	1	0	1,08E-02		w20 =	4,226
					w30 =	-4,78
		Минимальная абсолютная погрешность		Q =	8,74E-05	
		Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений				

Рис. 5.22. Еще один «наилучший» набор весов для примера 5.4.

Функция приспособленности для системы XOR задается формулой расчета погрешности Q (см. пример 5.3) и изменяется в пределах от 0 до 1. На рис. 5.21 заметно стремительное уменьшение наилучшего значения функции приспособленности до «наилучшего на данный момент» значения, равного 0,13159. Единица на временной оси этого графика соответствует 20 «тактам». Набор весов, к которому стремится «наилучшее решение», характеризуется тем, что
$w_{11} = w_{12}$ и $w_{21} = w_{22}$, что типично для логической системы XOR. Результаты, представленные на рис. 5.19 и 5.20, можно сравнить с показанными на рис. 5.17 для примера 5.3, а информацию на рис. 5.22 - сопоставить с рис. 5.18.
Заметим, что минимальная погрешность во всех трех случаях (рис. 5.19, 5.20 и 5.22) оказывается значительно меньше, чем для интервала изменения от -5 до 5, рассмотренного в примере 5.3 (рис. 5.17 и 5.18).

Пример 5.5

С помощью программы **Evolver** найти оптимальный набор весов нейронной сети, изображенной на рис. 3.11 (пример 3.3), если значения весов лежат в интервале от -15 до 15.
Применялся тот же генетический алгоритм программы **Evolver**, что и в предыдущих примерах, но для расширенного интервала изменения весов. После 30000 «тактов» получено «наилучшее решение», показанное на рис. 5.23.

входы		заданное выходное значение	расчетное выходное значение
u1	u2	d	y
0	0	0	5,26E-04
0	1	1	0,999369
1	0	1	0,999389
1	1	0	5,25E-04

веса	
w11 =	14,7225
w12 =	-14,985
w21 =	14,934
w22 =	-14,9865
w31 =	14,9775
w32 =	-14,946
w10 =	-7,107
w20 =	7,4685
w30 =	7,3755

Минимальная абсолютная погрешность Q = 3,31E-07

Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений

Рис. 5.23. «Наилучший» набор весов из интервала [-15, 15] для нейронной сети, реализующей систему XOR (пример 5.5).

Повторное выполнение того же алгоритма при тех же начальных условиях через 30000 «тактов» дало «наилучший» набор весов, представленный на рис.5.24.

входы		заданное выходное значение	расчетное выходное значение
u1	u2	d	y
0	0	0	5,23E-04
0	1	1	0,999346
1	0	1	0,999351
1	1	0	5,22E-04

	веса
w11 =	-14,9985
w12 =	14,94
w21 =	-14,7675
w22 =	14,9835
w31 =	14,913
w32 =	-14,9205
w10 =	-7,662
w20 =	7,464
w30 =	7,35

Минимальная абсолютная погрешность Q = 3,49E-07

Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений

Рис. 5.24. Другой «наилучший» набор весов для примера 5.5.

Главное различие между решениями на рис. 5.23 и 5.24 заключается в том, что в первом случае веса w_{11} и w_{21} положительны, а веса w_{12} и w_{22} отрицательны, тогда как во втором случае ситуация оказалась противоположной. В рассматриваемом примере получены две из множества возможных комбинаций весов для нейронной сети, реализующей систему XOR. Еще одно альтернативное решение приведено на рис. 5.25.

входы		заданное выходное значение	расчетное выходное значение
u1	u2	d	y
0	0	0	5,92E-04
0	1	1	0,999441
1	0	1	0,999437
1	1	0	5,85E-04

	веса
w11 =	14,223
w12 =	-14,5155
w21 =	-14,2815
w22 =	13,565
w31 =	14,9595
w32 =	14,9625
w10 =	-7,554
w20 =	-6,6645
w30 =	-7,458

Минимальная абсолютная погрешность Q = 3,30E-07

Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений

Рис.5.25. Еще один «наилучший» набор весов для примера 5.5 (получен после примерно 12000 «тактов»).

Значения входящих в него весов получены после примерно 12000 «тактов». Несложно предугадать, как будут изменяться эти значения при увеличении количества «тактов». Следовательно, это еще одна

комбинация, подобная решениям на рисунках 5.23 и 5.24, причем в данном случае веса w_{11}, w_{22}, w_{31} и w_{32} положительны, а веса w_{12}, w_{21}, w_{10}, w_{20} и w_{30} отрицательны.

Пример 5.6

С помощью программы **Evolver** найти оптимальный набор весов нейронной сети, изображенной на рис. 3.11 (пример 3.3), если значения весов лежат в интервале от -20 до 20.

Это та же задача, что и в предыдущих примерах, однако интервал изменения весов значительно расширен. Задача решалась при той же размерности популяции и таких же значениях показателей скрещивания и мутации, как и в предыдущих примерах. Столбчатая диаграмма на рис. 5.26 демонстрирует значения функции приспособленности особей в популяции после примерно 1500 «тактов».

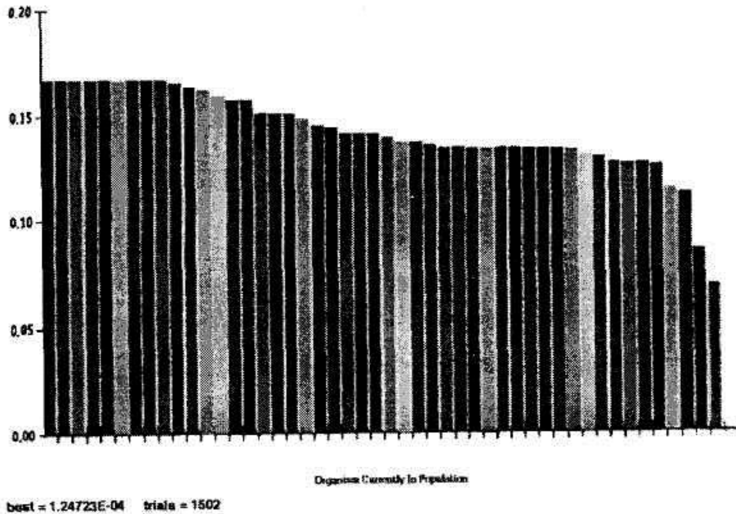


Рис. 5.26. Столбчатая диаграмма значений функции приспособленности особей в популяции при f = 1502 для примера 5.6.

Конечно, это значения в интервале от 0 до 1. На рис. 5.27 изображены графики изменения «наилучшего» (нижняя кривая) и среднего (верхняя кривая) значения функции приспособленности для этого примера после примерно 1500 «тактов».

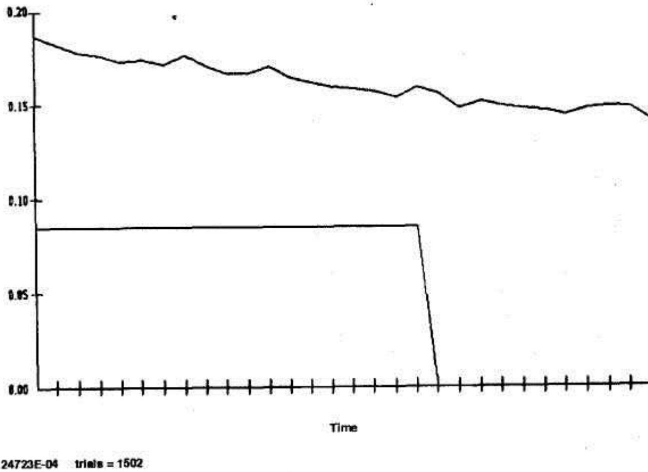


Рис. 5.27. Изменение «наилучшего» (внизу) и среднего (вверху) значения функции приспособленности для примера 5.6.

Единица на временной оси этого графика соответствует 20 «тактам». «Наилучшее на данный момент» (после 1502 «тактов») значение функции приспособленности составляет 1,247 * 10^{-4}. Рис. 5.28 представляет «наилучший» набор весов, полученный после 33000 «тактов», а рис. 5.29 - после 63000 «тактов». Заметно, что значения соединительных весов стремятся к 20 или к -20, а значения w_{10}, w_{20} и w_{30} - соответственно к 10, -10, 10.

входы		заданное выходное значение	расчетное выходное значение
u1	u2	d	y
0	0	0	4,82E-05
0	1	1	0,999952
1	0	1	0,999955
1	1	0	4,82E-05

веса	
w11 =	19,972
w12 =	19,826
w21 =	-19,994
w22 =	19,814
w31 =	-19,958
w32 =	19,89
w10 =	9,822
w20 =	-10,064
w30 =	10,016

Минимальная абсолютная погрешность Q = 2,23E-09

Это 1/4 суммы квадратов погрешностей (d - y)
для каждой пары входных значений

Рис. 5.28. «Наилучший» набор весов из интервала [-20, 20] при t = 33000 для при мера 5.6.

входы		заданное выходное значение	расчетное выходное значение		веса
				w11 =	-19,972
				w12 =	19,924
u1	u2	d	y	w21 =	-19,998
0	0	0	4,82E-05	w22 =	19,992
0	1	1	0,999958	w31 =	-19,958
1	0	1	0,999955	w32 =	19,976
1	1	0	4,82E-05	w10 =	10,412
				w20 =	-10,302
				w30 =	10,016

Минимальная абсолютная погрешность Q = 2,14E-09

Это 1/4 суммы квадратов погрешностей (d - y)
для каждой пары входных значений

Рис. 5.29. «Наилучший» набор весов из интервала [-20, 20] при t = 63000 для примера 5.6.

Если продолжить выполнение алгоритма, то при дальнейшем увеличении количества «тактов» некоторые веса примут значения, равные 20. Очевидно, что наборы весов на рис. 5.28 и 5.29 представляют собой лишь два элемента из множества допустимых комбинаций весов нейронной сети, реализующей логическую систему XOR. На рис. 5.30 показан совершенно другой набор «наилучших» весов, полученных при очередном возобновлении генетического алгоритма программы **Evolver.**

входы		заданное выходное значение	расчетное выходное значение		веса
				w11 =	-19,992
				w12 =	-19,986
u1	u2	d	y	w21 =	13,376
0	0	0	4,65E-05	w22 =	13,356
0	1	1	0,999954	w31 =	-19,988
1	0	1	0,999954	w32 =	-19,992
1	1	0	4,73E-05	w10 =	9,648
				w20 =	-20
				w30 =	10,01

Минимальная абсолютная погрешность Q = 2,17E-09

Это 1/4 суммы квадратов погрешностей (d - y)
для каждой пары входных значений

Рис. 5.30. Другой «наилучший» набор весов для примера 5.26.

Результаты зафиксированы после примерно 15000 тактов при тех же, что и прежде, размерности популяции, значениях показателей скрещивания и мутации и интервале изменения весов.

Отметим, что «наилучшее решение» на рис. 5.30 тоже представляет собой одну из допустимых комбинаций весов системы XOR, поскольку оно стремится к оптимальному решению, в котором $w_{11}=w_{12}$ и $w_{21}= w_{22}$. Из примеров 5.3 – 5.6 следует вывод, что чем больше интервал изменения весов, тем меньше минимальное значение погрешности. Поэтому для получения нейронной сети, которая сможет наиболее точно реализовать систему XOR (т.е. для которой разность между эталонным и расчетным выходными значениями будет минимальной), необходимо подбирать веса из как можно более широкого интервала допустимых значений. Очевидно, что подбор этих весов должен быть оптимальным и минимизировать погрешность так, как это делалось в примерах 5.3 – 5.6. Если бы требовалось выбирать значения весов из интервала, более узкого, чем [-5, 5], то обеспечить минимальную погрешность в пределах, например, 0,1, оказалось бы еще сложнее. На рис. 5.31 показан «наилучший» набор весов, найденный также как в примерах 3.23 - 3.26, но для интервала
[-3, 3].

входы		заданное выходное значение	расчетное выходное значение
u1	u2	d	y
0	0	0	0,323599
0	1	1	0,679728
1	0	1	0,681891
1	1	0	0,323578

	веса
w11 =	2,9946
w12 =	-2,9964
w21 =	-2,9934
w22 =	2,9949
w31 =	2,9997
w32 =	2,9991
w10 =	-2,0394
w20 =	-2,0559
w30 =	1,4229

Минимальная абсолютная погрешность Q = 0,103297

Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений

Рис. 5.31. «Наилучший» набор весов из интервала [-3, 3] для нейронной сети, реализующей систему XOR (пример 5.6).

После 40000 «тактов» минимальное значение погрешности превышало 0,1. Заметим, что для интервала изменения весов от -20 до 20 можно достичь погрешности порядка 10^{-4} уже через 1500 «тактов» (рис. 5.26 и 5.27), а для интервала [-10, 10] минимальная погрешность на уровне 0,1 была достигнута уже через 500 «тактов» (рис. 5.21). В примерах 5.3 – 5.6 начальные значения весов, которые учитываются программой **Evolver** в качестве генов одной из особей исходной популяции, в общем случае принимаются равными 1. Для нейронной сети,

изображенной на рис. 3.11, в этом случае величина погрешности будет составлять Q = 0,438, что сильно отличается от минимального значения. Конечно, можно принять и другие начальные значения, далекие от оптимальных.

Пример 5.7

С помощью программы **Evolver** найти оптимальный набор весов нейронной сети, изображенной на рис. 3.11 (пример 3.3), если значения весов лежат в интервале от -7 до 8.

Начальные значения весов, составляющие генотип одной из особей исходной популяции, представлены на рис. 5.32.

входы		заданное выходное значение	расчетное выходное значение			веса
					w11 =	3,1592
					w12 =	4,5006
					w21 =	1,715
u1	u2	d	y		w22 =	-2,5636
0	0	0	0,105235		w31 =	-2,5994
0	1	1	0,290778		w32 =	-2,0924
1	0	1	0,016872		w10 =	-0,321
1	1	0	0,204858		w20 =	-2,2872
					w30 =	-0,8546
		Минимальная абсолютная погрешность	Q =	0,380644		
		Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений				

Рис. 5.32. Начальные значения весов для примера 5.7.

Видно, что значения y для конкретных пар входов совершенно не соответствуют функции XOR. Значение погрешности Q = 0,38 считается абсолютно неудовлетворительным.

Для решения задачи применялся тот же генетический алгоритм при той же размерности популяции и тех же показателях скрещивания и мутации, что и в предыдущих примерах. Всего лишь через 96 «тактов» получены наилучшие значения функции приспособленности конкретных особей в популяции, показанные на рис. 5.33 (столбчатая диаграмма).

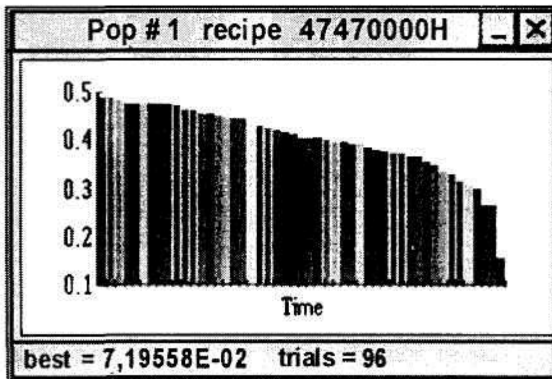


Рис. 5.33. Столбчатая диаграмма значений функции приспособленности особей в популяции **при** t = 96 для примера 5.7.

В свою очередь, на рис. 5.34 представлена таблица, содержащая генотипы всех 50 особей этой популяции (состоящие из девяти действительных чисел, соответствующих конкретным весам) и значения их функции приспособленности, размещенные в первой колонке.

Pop # 1 recipe 47470000H

	Result	Var 1	Var 2	Var 3	Var 4	Var 5	Var 6	Var 7	Var 8	Var 9
Org 1	4,730e-01	1,495e+00	-3,729e+00	4,569e+00	1,369e+00	1,797e+00	3,471e+00	-6,598e+00	5,928e-01	2,658e+00
Org 2	4,729e-01	-1,947e+00	-7,634e+00	1,390e+00	6,614e+00	6,276e+00	-2,782e+00	3,845e+00	4,912e-01	-2,026e+00
Org 3	4,723e-01	3,145e+00	6,210e+00	-6,679e+00	3,765e+00	-3,720e+00	7,513e+00	9,528e-01	7,973e+00	-1,066e+00
Org 4	4,718e-01	1,495e+00	2,304e+00	4,569e+00	1,369e+00	-7,639e+00	3,471e+00	-5,995e+00	-2,605e+00	2,658e+00
Org 5	4,703e-01	1,495e+00	2,304e+00	1,782e+00	-4,678e+00	1,406e+00	3,471e+00	-5,995e+00	-2,605e+00	3,256e+00
Org 6	4,689e-01	-1,947e+00	7,910e+00	3,923e+00	3,328e-01	6,276e+00	-2,782e+00	3,845e+00	2,116e+00	-2,096e-01
Org 7	4,629e-01	-2,364e+00	4,739e+00	6,166e+00	3,466e+00	5,334e+00	-1,915e+00	-5,150e+00	6,773e+00	4,973e+00
Org 8	4,594e-01	1,334e+00	1,287e+00	-7,035e+00	-2,564e+00	-2,599e+00	5,108e+00	4,480e-01	-2,589e+00	-8,546e-01
Org 9	4,548e-01	-7,533e+00	-3,729e+00	5,872e-01	-3,490e+00	2,453e+00	7,688e+00	3,758e+00	5,928e-01	1,553e+00
Org 10	4,521e-01	6,658e+00	3,717e+00	-6,678e+00	4,170e+00	4,403e+00	3,144e-01	-2,526e+00	6,720e+00	1,570e+00
Org 11	4,485e-01	1,806e+00	-4,501e+00	4,265e+00	-2,564e+00	-2,599e+00	-3,688e-01	-3,466e+00	-2,287e+00	-2,523e+00
Org 12	4,435e-01	6,112e+00	5,577e+00	5,987e+00	-5,248e+00	-2,518e+00	-7,077e+00	1,742e+00	-6,743e+00	3,150e+00
Org 13	4,429e-01	-2,574e+00	-5,763e+00	-3,503e+00	-1,814e+00	1,094e+00	4,170e+00	-7,954e+00	7,736e-01	1,957e+00
Org 14	4,409e-01	-7,533e+00	-3,729e+00	4,265e+00	-3,490e+00	1,797e+00	-3,698e-01	-3,466e+00	5,928e-01	-2,523e+00
Org 15	4,378e-01	2,824e-01	-1,276e+00	-4,874e+00	1,115e+00	-6,130e+00	-2,594e+00	6,620e+00	3,699e+00	4,680e+00
Org 16	4,334e-01	4,467e+00	-7,147e+00	-5,989e+00	1,319e+00	4,403e+00	-6,583e+00	1,824e+00	-6,600e+00	-1,247e+00
Org 17	4,323e-01	-7,920e-01	7,984e+00	4,569e+00	3,794e+00	-7,639e+00	3,471e+00	-5,395e+00	6,706e+00	2,658e+00
Org 18	4,277e-01	-7,533e+00	-3,729e+00	5,872e-01	-3,490e+00	-2,142e+00	7,688e+00	3,758e+00	1,156e+00	-1,713e+00
Org 19	4,111e-01	1,679e+00	-1,690e+00	-3,872e+00	-3,364e+00	-2,654e+00	7,179e+00	-5,344e-01	3,536e+00	2,199e+00
Org 20	4,077e-01	-5,066e+00	5,777e+00	-5,989e+00	1,319e+00	4,403e+00	-4,075e+00	4,274e+00	4,362e+00	-1,247e+00
Org 21	4,036e-01	-5,066e+00	-6,873e+00	2,008e+00	5,871e+00	-2,574e+00	-4,075e+00	4,274e+00	4,362e+00	7,990e+00
Org 22	3,990e-01	1,334e+00	1,938e+00	-7,035e+00	7,767e+00	2,291e+00	-6,376e+00	-1,358e+00	-2,589e+00	-8,546e-01
Org 23	3,906e-01	3,159e+00	-4,501e+00	-7,035e+00	-2,564e+00	-2,599e+00	-6,376e+00	4,480e-01	-3,432e-01	-8,546e-01
Org 24	3,838e-01	-6,824e+00	-6,480e-01	-6,678e+00	8,424e-01	3,886e+00	-3,650e+00	-2,526e+00	-4,260e+00	-1,862e+00
Org 25	3,806e-01	3,159e+00	-4,501e+00	1,715e+00	-2,564e+00	-2,599e+00	-2,092e+00	-3,210e-01	-2,287e+00	-8,546e-01
Org 26	3,806e-01	7,302e+00	-1,332e+00	4,944e+00	1,107e+00	-4,405e+00	-4,642e+00	-7,054e+00	-5,908e+00	2,564e+00
Org 27	3,788e-01	1,431e+00	7,098e+00	4,265e+00	2,654e+00	1,797e+00	-3,688e-01	-3,466e+00	3,280e+00	-2,523e+00
Org 28	3,713e-01	7,593e+00	3,940e+00	-3,729e-01	5,424e+00	9,824e-01	-5,086e+00	3,689e+00	-1,642e+00	3,546e+00
Org 29	3,711e-01	4,467e+00	-4,987e+00	-3,894e+00	2,440e-01	-1,488e-01	-6,583e+00	1,243e+00	1,288e+00	1,099e+00
Org 30	3,667e-01	4,467e+00	3,717e+00	-3,894e+00	4,170e+00	4,403e+00	-6,583e+00	-2,526e+00	6,720e+00	1,099e+00
Org 31	3,643e-01	-3,692e+00	-7,147e+00	-5,989e+00	1,319e+00	1,442e+00	-7,009e+00	1,824e+00	-6,600e+00	-1,247e+00
Org 32	3,566e-01	1,061e+00	1,508e+00	4,250e+00	1,014e+00	6,583e+00	-2,174e+00	-2,698e+00	-7,648e-01	-2,432e-01
Org 33	3,497e-01	6,658e+00	3,717e+00	-5,350e+00	9,408e-01	-6,238e+00	4,889e+00	6,626e+00	1,363e+00	1,570e+00
Org 34	3,442e-01	1,495e+00	-1,280e-01	4,250e+00	1,014e+00	6,583e+00	-2,174e+00	-7,766e+00	-2,605e+00	-2,432e-01
Org 35	3,423e-01	4,467e+00	5,777e+00	-5,989e+00	8,320e-01	4,403e+00	-4,075e+00	1,249e+00	1,288e+00	-1,247e+00
Org 36	3,402e-01	-2,802e+00	-8,872e-01	-5,430e+00	-5,268e+00	1,878e+00	7,349e+00	-5,782e+00	-2,847e+00	-1,260e+00
Org 37	3,386e-01	4,758e+00	-4,501e+00	7,191e+00	-2,564e+00	1,486e+00	9,120e-02	3,460e+00	1,950e+00	-8,546e-01
Org 38	3,349e-01	1,334e+00	1,338e+00	-6,678e+00	4,170e+00	2,291e+00	-6,376e+00	-1,358e+00	6,720e+00	1,570e+00
Org 39	3,348e-01	-5,190e+00	8,080e-02	3,923e+00	3,328e-01	2,421e+00	-2,498e+00	3,845e+00	2,116e+00	-2,096e-01
Org 40	3,286e-01	4,467e+00	2,304e+00	1,782e+00	4,170e+00	4,403e+00	-6,583e+00	-2,526e+00	6,720e+00	3,256e+00
Org 41	3,223e-01	1,334e+00	1,938e+00	1,364e+00	7,767e+00	2,291e+00	5,108e+00	-1,358e+00	-2,588e+00	-7,888e-01
Org 42	3,147e-01	3,300e+00	-2,718e+00	-4,326e+00	1,847e+00	-3,518e+00	-3,644e+00	7,678e+00	3,561e+00	4,307e+00
Org 43	2,925e-01	1,334e+00	-1,332e+00	-7,035e+00	1,107e+00	2,291e+00	-4,642e+00	-7,054e+00	-5,908e+00	-8,546e-01
Org 44	2,893e-01	6,112e+00	-4,501e+00	6,536e+00	-4,044e+00	-2,518e+00	6,296e-01	1,742e+00	-2,287e+00	3,150e+00
Org 45	2,712e-01	-3,882e+00	-3,528e-01	6,533e+00	-7,691e+00	-3,912e+00	-6,151e+00	6,254e+00	5,270e+00	5,142e+00
Org 46	2,628e-01	7,386e+00	7,810e+00	-3,099e+00	3,328e-01	5,682e+00	-1,783e+00	-5,232e+00	1,136e-01	-2,096e-01
Org 47	2,508e-01	3,153e+00	-4,501e+00	1,715e+00	-3,387e+00	-2,699e+00	3,144e-01	6,162e+00	6,720e+00	1,570e+00
Org 48	2,091e-01	-2,726e+00	-7,593e+00	5,533e+00	-4,528e+00	-4,086e+00	1,626e+00	6,840e-01	-3,542e+00	-8,868e-01
Org 49	2,089e-01	-7,870e+00	-7,277e+00	6,848e+00	2,642e+00	-6,345e+00	-2,023e+00	3,656e+00	-2,590e+00	1,225e+00
Org 50	7,196e-02	5,940e+00	-6,462e+00	-7,697e+00	5,495e+00	6,822e+00	7,410e+00	-6,834e+00	-4,494e+00	-5,816e-01

☐ Live Update [Update]

Рис.5.34. Популяция особей при t - 96 для примера 5.7.

Обратим внимание на то, что начальная хромосома с аллелями, равными значениям весов из рис. 5.32, расположена в средней части этой таблицы и обозначена «Организм 25». Следует помнить, что особи в популяции упорядочены от «наихудшего» к «наилучшему». Последняя особь («Организм 50») - это хромосома, «наилучшая на данный момент», т.е. после 96 «тактов». Значения весов, входящих в состав этой хромосомы, показаны на рис. 5.35 вместе с выходными значениями для конкретных пар входов и значением минимизируемой погрешности. Итак, после 96 «тактов» получен исключительно хороший результат.

входы		заданное выходное значение	расчетное выходное значение
u1	u2	d	y
0	0	0	0,356101
0	1	1	0,991321
1	0	1	0,785702
1	1	0	0,33914

	веса
w11 =	5,94
w12 =	-6,4616
w21 =	-7,597
w22 =	5,4952
w31 =	6,8216
w32 =	7,4104
w10 =	-6,8336
w20 =	-4,4936
w30 =	-0,6816

Минимальная абсолютная погрешность Q = 0,071956

Это 1/4 суммы квадратов погрешностей (d - y)
для каждой пары входных значений

Рис.5.35. Набор весов, полученный при t = 96 для примера 5.7.

При повторном выполнении этой же программы с самого начала при том же исходном наборе весов (рис. 5.32) через примерно 10000 «тактов» получена погрешность Q = 0,0962, через 20000 «тактов» - Q = 0,07956, а через 21000 «тактов» - Q = 0,0687. После 24000 тактов эта погрешность уменьшилась до значения Q=0,0077 (рис. 5.36), а после 40000 «тактов» получен «наилучший» набор весов, показанный на рис. 5.37.

входы		заданное выходное значение	расчетное выходное значение
u1	u2	d	y
0	0	0	0,021682
0	1	1	0,864866
1	0	1	0,952198
1	1	0	0,096875

веса	
w11 =	7,176
w12 =	-7,953
w21 =	5,8144
w22 =	-3,4576
w31 =	-6,0736
w32 =	7,9752
w10 =	3,936
w20 =	-3,7112
w30 =	1,9576

Минимальная абсолютная погрешность Q = 7,60E-03

Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений

Рис. 5.36. Результат, полученный после 24000 «тактов», при повторном запуске программы **Evolver** для примера 5.7.

входы		заданное выходное значение	расчетное выходное значение
u1	u2	d	y
0	0	0	0,020804
0	1	1	0,975645
1	0	1	0,97539
1	1	0	0,020713

веса	
w11 =	7,9696
w12 =	-7,953
w21 =	7,9816
w22 =	-7,9976
w31 =	-7,964
w32 =	7,9752
w10 =	3,936
w20 =	-4,1104
w30 =	3,8312

Минимальная абсолютная погрешность Q = 5,16E-04

Это 1/4 суммы квадратов погрешностей (d - y) для каждой пары входных значений

Рис. 5.37. «Наилучший» набор весов, полученный для случая рис. 5.36, после 40000 «тактов».

Таким образом, процессы выполнения одного и того же алгоритма при одинаковой размерности популяции и повторяющихся значениях показателей скрещивания и мутации оказываются совершенно различными. Это, конечно, определяется случайным набором особей исходной популяции и случайнымм проведением скрещивания и мутации.

Для случая на рис. 5.35 получен результат, подобный представленному на рис. 5.37, причем веса w_{21}, w_{22}, w_{31}, а также w_{10} и w_{30} имеют противоположные знаки.

При использовании генетического алгоритма программы **Evolver** для решения задач, аналогичных рассмотренным в примерах 5.3 – 5.7, генетические операторы изменяют значения отдельных весов

(мутация) и осуществляют обмен значений весов, выбранных случайным образом, между двумя хромосомами (скрещивание). В то же время, в программе **FlexTool** (пример 4.9) генетические операторы рекомбинируют гены внутри хромосом, которые представляют собой двоичные последовательности, соответствующие закодированным значениям конкретных весов. В программе **Evolver** хромосомы состоят из генов, имеющих действительные значения, совпадающие со значениями весов. Программа **Evolver** - это прекрасный инструмент для оптимизации функции нескольких переменных, при этом она может применяться для поиска максимума или минимума функции двух и даже одной переменной. Заметим, что в случае только одной переменной скрещивание практически не производится, и выполняется только операция мутации, что характерно для так называемого эволюционного программирования. Все примеры, решенные в при помощи программы **FlexTool,** можно решать также и средствами программы **Evolver.** Например, на рис. 5.38 показано изменение «наилучшего» и среднего значения функции приспособленности в случае поиска максимума функции, представленной на рис.4.6.

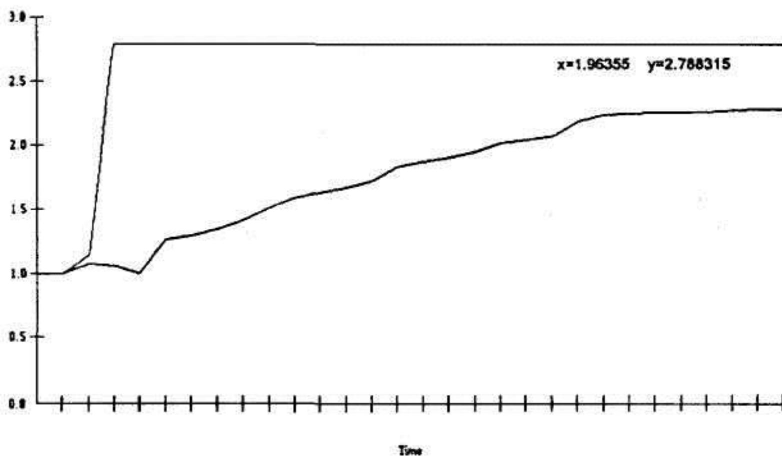


Рис. 5.38. Изменение «наилучшего» и среднего значения функции приспособленности в случае поиска максимума функции, изображенной на рис. 4.6, с помощью программы **Evolver.**

Одно деление на временной оси соответствует 20 «тактам». Вычисления проводились с принятыми по умолчанию значениями

показателей скрещивания (0,5) и мутации (0,06), а также размерности популяции (50). Графики и «наилучшие» решения регистрировались после 700 «тактов» (t = 700). Пример 5.8 посвящен оптимизации функции двух переменных, график которой изображен на рис.4.23.

Пример 5.8

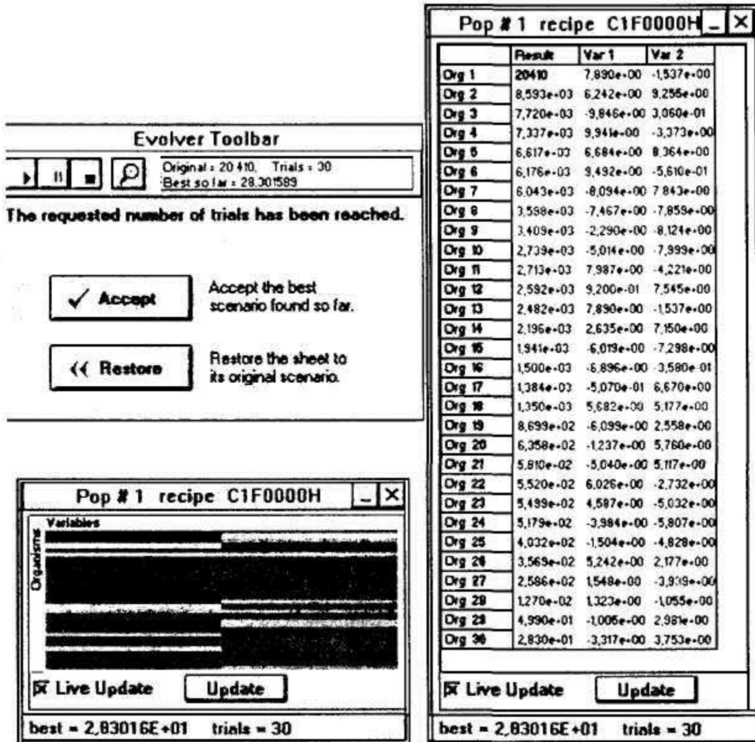
С помощью программы **Evolver** найти минимум функции из примера 4.7.

График этой функции изображен на рис.4.23. В примере 4.7 приведены координаты четырех точек, в которых эта функция имеет минимальное значение, равное 0. Генетический алгоритм должен найти одну из этих точек. Применяется программа **Evolver** с принятыми по умолчанию значениями показателей скрещивания (0,5) и мутации (0,06). Размерность популяции выбрана равной 30. На рис. 5.39 представлены начальные значения переменных x_1 и x_2, которые введены в исходную популяцию в качестве генов одной из хромосом.

	A	B	C	D	E	F	G	H	I	J	K
1											
2			ФУНКЦИЯ	HIMMELBLAU				Вычисление минимума			
3											
4			F(X1,X2) =	(X1^2 +X2 -11)^2 + (X1 +X2^2 -7)^2				-10 <=	X1,X2	<= 10	
5											
6											
7	X1 =	10									
8	X2 =	10									
9	F(X1,X2)=	20410									

Рис. 5.39. Начальные значения для примера 5.8.

Значение функции приспособленности для этой хромосомы очень велико - оно составляет 20410. Понятно, что данная хромосома будет очень скоро исключена из популяции, что подтверждается рис. 5.40.

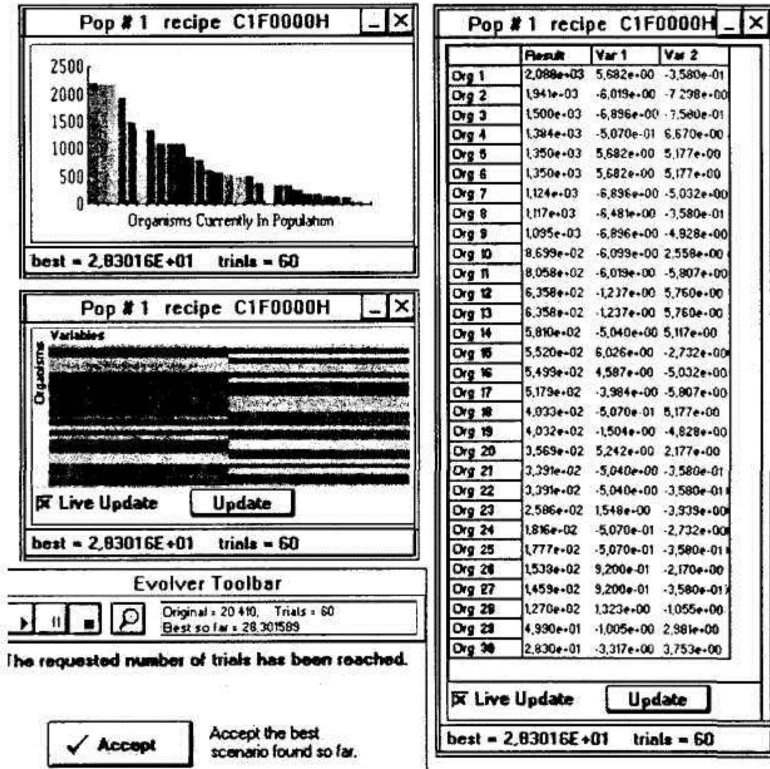


	Result	Var 1	Var 2
Org 1	20410	7,890e+00	-1,537e+00
Org 2	8,593e+03	6,242e+00	9,256e+00
Org 3	7,720e+03	-9,846e+00	3,060e-01
Org 4	7,337e+03	9,941e+00	-3,373e+00
Org 5	6,617e+03	6,684e+00	8,364e+00
Org 6	6,176e+03	9,492e+00	-5,610e-01
Org 7	6,043e+03	-8,094e+00	7,843e+00
Org 8	3,598e+03	-7,467e+00	-7,859e+00
Org 9	3,409e+03	-2,290e+00	-8,124e+00
Org 10	2,739e+03	-5,014e+00	-7,999e+00
Org 11	2,713e+03	7,987e+00	-4,221e+00
Org 12	2,592e+03	9,200e-01	7,545e+00
Org 13	2,482e+03	7,890e+00	-1,537e+00
Org 14	2,196e+03	2,635e+00	7,150e+00
Org 15	1,941e+03	-6,019e+00	-7,298e+00
Org 16	1,500e+03	-6,896e+00	-3,580e-01
Org 17	1,384e+03	-5,070e-01	6,670e+00
Org 18	1,350e+03	5,682e+00	5,177e+00
Org 19	8,699e+02	-6,099e+00	2,558e+00
Org 20	6,358e+02	-1,237e+00	5,760e+00
Org 21	5,810e+02	-5,040e+00	5,117e+00
Org 22	5,520e+02	6,026e+00	-2,732e+00
Org 23	5,499e+02	4,587e+00	-5,032e+00
Org 24	5,179e+02	-3,984e+00	-5,807e+00
Org 25	4,032e+02	-1,504e+00	-4,828e+00
Org 26	3,569e+02	5,242e+00	2,177e+00
Org 27	2,586e+02	1,548e+00	-3,939e+00
Org 28	1,270e+02	1,323e+00	-1,055e+00
Org 29	4,990e+01	-1,006e+00	2,981e+00
Org 30	2,830e+01	-3,317e+00	3,753e+00

Рис.5.40. Популяция особей при f=30 для примера 5.8.

На этом рисунке показаны хромосомы популяции для t=30 (после 30 «тактов»), что соответствует первой итерации (первому поколению) классического генетического алгоритма. Переменные *var*1 и *var2* обозначают соответственно x_1, и x_2, первый столбец *(result)* содержит значения функции приспособленности конкретных хромосом. Первое значение (20410) принадлежит особи, исключенной из популяции. На ее место вводится новая хромосома с аллелями, равными 7,89 и -1,537, для которой значение функции приспособленности еще только предстоит рассчитать. В левом нижнем углу рисунка демонстрируется разнородность особей этой популяции. В данном случае она также довольно велика. На рис. 5.41 приведены аналогичные графики для

популяции после 60 «тактов» (*t*=60), а также столбчатая диаграмма, иллюстрирующая конкретные особи.



	Result	Var 1	Var 2
Org 1	2,088e+03	5,682e+00	-3,580e-01
Org 2	1,941e+03	-6,019e+00	-7,298e+00
Org 3	1,500e+03	-6,896e+00	7,580e-01
Org 4	1,384e+03	-5,070e-01	6,670e+00
Org 5	1,350e+03	5,682e+00	5,177e+00
Org 6	1,350e+03	5,682e+00	5,177e+00
Org 7	1,124e+03	-6,896e+00	-5,032e+00
Org 8	1,117e+03	-6,481e+00	-3,580e-01
Org 9	1,095e+03	-6,896e+00	-4,928e+00
Org 10	8,699e+02	-6,099e+00	2,558e+00
Org 11	8,058e+02	-6,019e+00	-5,807e+00
Org 12	6,358e+02	-1,237e+00	5,760e+00
Org 13	6,358e+02	-1,237e+00	5,760e+00
Org 14	5,810e+02	-5,040e+00	5,117e+00
Org 15	5,520e+02	6,026e+00	-2,732e+00
Org 16	5,499e+02	4,587e+00	-5,032e+00
Org 17	5,179e+02	-3,984e+00	-5,807e+00
Org 18	4,033e+02	-5,070e-01	5,177e+00
Org 19	4,032e+02	-1,504e+00	-4,828e+00
Org 20	3,569e+02	5,242e+00	2,177e+00
Org 21	3,391e+02	-5,040e+00	-3,580e-01
Org 22	3,391e+02	-5,040e+00	-3,580e-01
Org 23	2,586e+02	1,548e+00	-3,939e+00
Org 24	1,816e+02	-5,070e-01	-2,732e+00
Org 25	1,777e+02	-5,070e-01	-3,580e-01
Org 26	1,533e+02	9,200e-01	-2,170e+00
Org 27	1,459e+02	9,200e-01	-3,580e-01
Org 28	1,270e+02	1,323e+00	-1,055e+00
Org 29	4,990e+01	-1,005e+00	2,981e+00
Org 30	2,830e+01	-3,317e+00	3,753e+00

Рис. 5.41. Популяция особей при *t*=60 для примера 5.8.

«Наилучшее на данный момент» значение функции приспособленности все еще слишком велико и составляет 28,3.
На рис. 5.42 представлены те же графики после 150 «тактов»
(*t*=150), дополненные (в левом нижнем углу) графиком изменения «наилучшего» значения функции приспособленности, которое стремительно уменьшается.

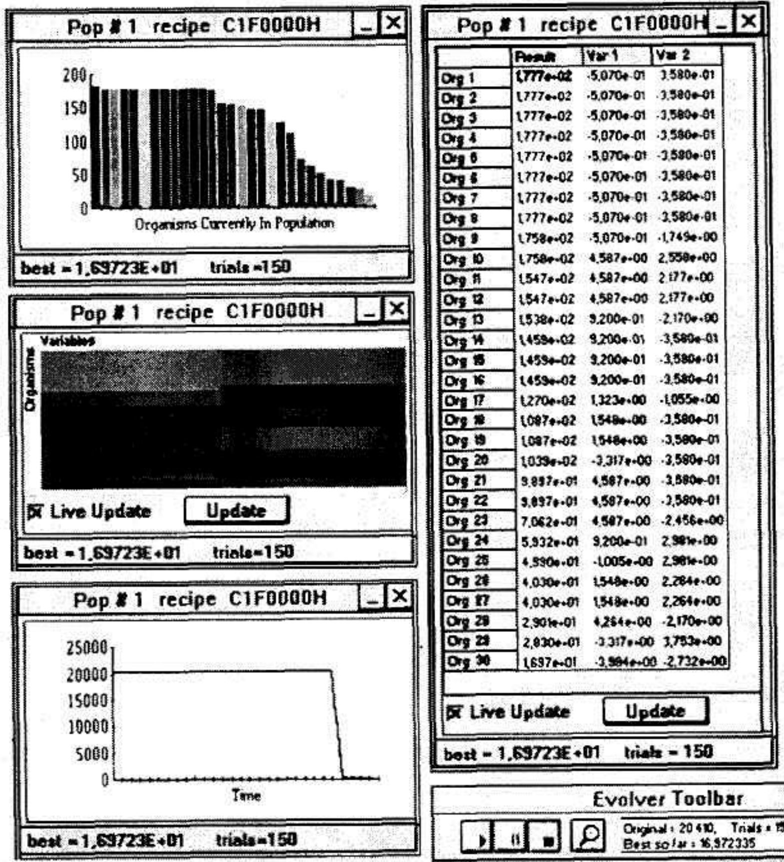
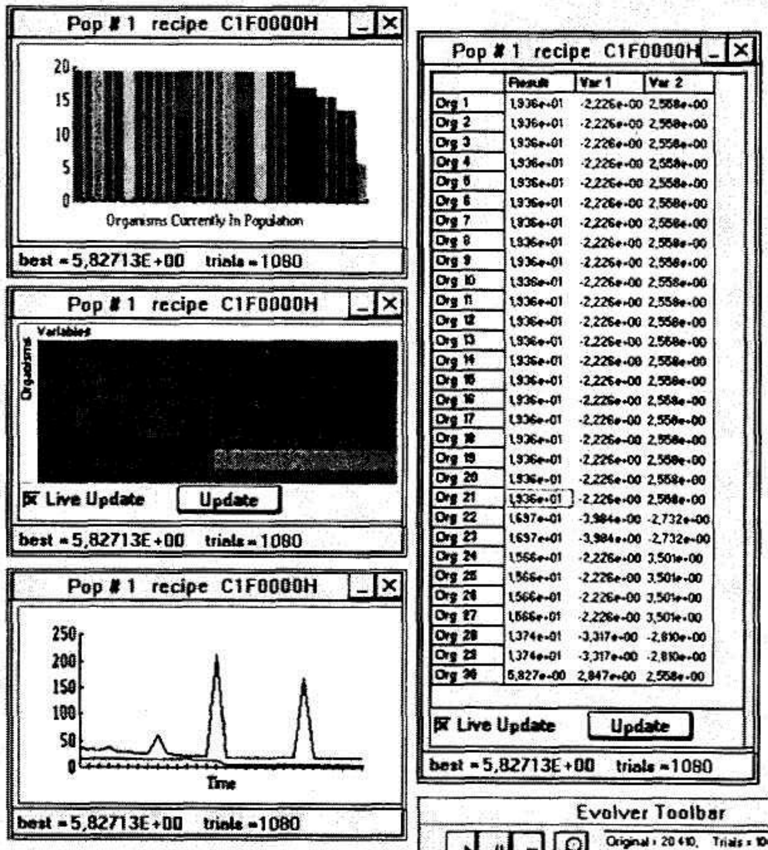


Рис. 5.42. Популяция особей и график функции приспособленности при t=150 для примера 5.8.

В средней части слева показана разнородность популяции, которая также значительно снизилась.

Еще меньшая разнородность популяции наблюдается на рис. 5.43, который содержит также столбчатую диаграмму и значения генов конкретных хромосом популяции.



	Result	Var 1	Var 2
Org 1	1,936e+01	-2,226e+00	2,558e+00
Org 2	1,936e+01	-2,226e+00	2,558e+00
Org 3	1,936e+01	-2,226e+00	2,558e+00
Org 4	1,936e+01	-2,226e+00	2,558e+00
Org 5	1,936e+01	-2,226e+00	2,558e+00
Org 6	1,936e+01	-2,226e+00	2,558e+00
Org 7	1,936e+01	-2,226e+00	2,558e+00
Org 8	1,936e+01	-2,226e+00	2,558e+00
Org 9	1,936e+01	-2,226e+00	2,558e+00
Org 10	1,936e+01	-2,226e+00	2,558e+00
Org 11	1,936e+01	-2,226e+00	2,558e+00
Org 12	1,936e+01	-2,226e+00	2,558e+00
Org 13	1,936e+01	-2,226e+00	2,558e+00
Org 14	1,936e+01	-2,226e+00	2,558e+00
Org 15	1,936e+01	-2,226e+00	2,558e+00
Org 16	1,936e+01	-2,226e+00	2,558e+00
Org 17	1,936e+01	-2,226e+00	2,558e+00
Org 18	1,936e+01	-2,226e+00	2,558e+00
Org 19	1,936e+01	-2,226e+00	2,558e+00
Org 20	1,936e+01	-2,226e+00	2,558e+00
Org 21	1,936e+01	-2,226e+00	2,558e+00
Org 22	1,697e+01	-3,984e+00	-2,732e+00
Org 23	1,697e+01	-3,984e+00	-2,732e+00
Org 24	1,566e+01	-2,226e+00	3,501e+00
Org 25	1,566e+01	-2,226e+00	3,501e+00
Org 26	1,566e+01	-2,226e+00	3,501e+00
Org 27	1,566e+01	-2,226e+00	3,501e+00
Org 28	1,374e+01	-3,317e+00	-2,810e+00
Org 29	1,374e+01	-3,317e+00	-2,810e+00
Org 30	5,827e+00	2,847e+00	2,558e+00

Рис. 5.43. Популяция особей и график функции приспособленности при t=1080 для примера 5.8.

«Наименьшее» значение функции приспособленности здесь составляет 5,83 для t=1080, т.е. после 1080 «тактов». Это значение все еще намного больше минимального. Графики в левой нижней части рисунка показывают изменение среднего (верхняя кривая) и «наилучшего» (нижняя кривая) значения функции приспособленности.
На рис. 5.44 представлены те же графики после 8000 «тактов» (t=8000). Заметно, что как «наилучшее», так и среднее значение функции приспособленности в популяции принимают значения, близкие к 0. На

всех графиках одно деление на временной оси соответствует 20 «тактам».

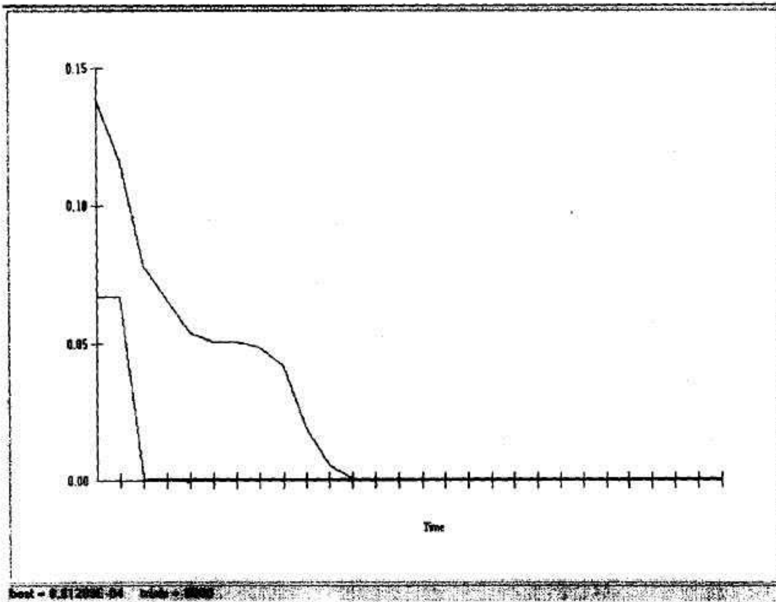


Рис. 5.44. График функции приспособленности при t = 8000 для примера 5.8.

На рис. 5.45 изображена столбчатая диаграмма особей популяции для t=8000. Заметноы, что все хромосомы в популяции стали одинаковыми.

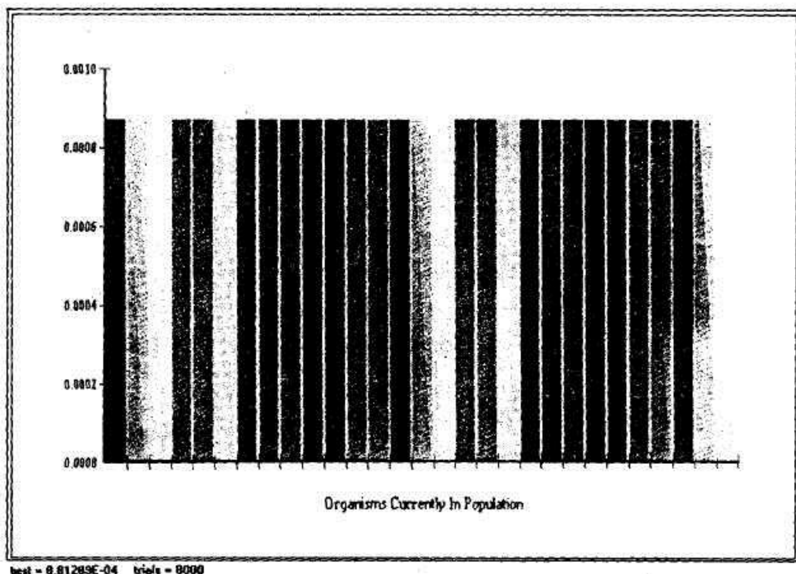


Рис. 5.45. Столбчатая диаграмма значений функции приспособленности особей в популяции при t = 8000 для примера 5.8.

Значение их функции приспособленности, очевидно, такое же, как и для «наилучшего» решения, т.е. равно 0,000881.

	A	B	C	D	E	F	G	H	I	J	K
1											
2			ФУНКЦИЯ	HIMMELBLAU			Вычисление минимума				
3											
4			F(X1,X2) = (X1^2 +X2 -11)^2 + (X1 +X2^2 -7)^2					-10 <=	X1,X2 <=	10	
5											
6											
7	X1 =	-3,78									
8	X2 =	-3,279									
9	F(X1,X2)=	0,000881									

Рис. 5.46. «Наилучшее» решение для примера 5.8 (полученное при t = 8000).

«Наилучшее» решение показано на рис. 5.46. Это хромосома со значениями переменных x_1 = -3,78 и x_2 = -3,279.

5.2.2. Решение комбинаторных задач с помощью программы Evolver

При решении комбинаторных задач проблема заключается в поиске наилучшего решения среди возможных перестановок параметров задачи. В качестве примера можно назвать сортировку списка имен (пример 5.9) или задачу коммивояжера. В программе **Evolver** для решения комбинаторных задач применяются генетические операторы, определение которых несколько отличается от аналогичных операторов, ориентированных на оптимизационные задачи. В частности:

Скрещивание разбивается на следующие шаги:

1) случайным образом выбираются позиции у первого родителя; их количество зависит от показателя скрещивания;

2) находятся позиции с такими же значениями генов (аллелями) у второго родителя;

3) значения оставшихся позиций первого родителя копируются на оставшиеся позиции второго родителя в последовательности, в которой они записаны у первого родителя.

Описанный способ скрещивания иллюстрируется на рис. 5.47.

Рис. 5.47. Скрещивание с сохранением порядка в генетическом алгоритме программы **Evolver**.

На этом рисунке показаны две хромосомы родителей, состоящие из семи генов со значениями из интервала целых чисел от 1 до 7. Каждый ген в хромосоме характеризуется уникальным значением. Каждая хромосома представляет собой перестановку натуральных чисел от 1 до 7. Под каждым геном указан номер его позиции (locus). Допустим, что показатель скрещивания равен 0,5, и у первого родителя

случайным образом выбраны позиции 1, 4, 5, 6, на которых находятся значения 3, 7, 6, 2 соответственно. У второго родителя эти значения находятся на позициях 1, 5, 6, 7. В результате копирования значений оставшихся позиций первого родителя (т.е. чисел 5, 1, 4 с позиций 2, 3, 7 соответственно) на оставшиеся позиции второго родителя (т.е. на позиции 2, 3, 4) в последовательности, в которой они записаны у первого родителя, образуется потомок со значениями генов 3, 5, 1,4,7,2,6.
Представленный метод скрещивания применяется в программе **Evolver** для решения задач, которые сводятся к поиску хромосом с наилучшим упорядочением генов. Описываемый способ подобен упорядоченному скрещиванию *(order crossover)*, показанному на рис. 5.48.

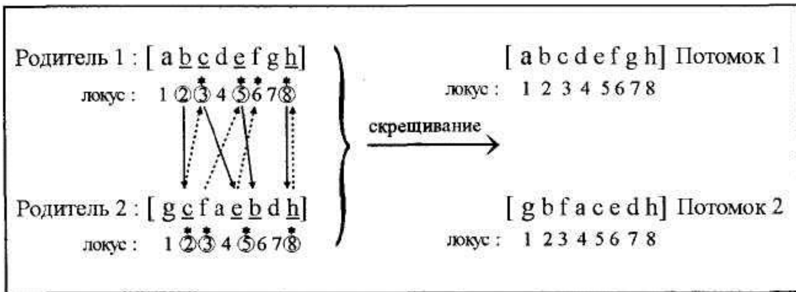


Рис. 5.48. Упорядоченное скрещивание *(order crossover).*

Номера позиций выбираются случайным образом. Далее значения генов с выбранных позиций одного из родителей переносятся на соответствующие позиции второго родителя. В результате скрещивания образуются два потомка.
Мутация. Оператор мутации реализует так называемую мутацию, основанную на упорядочении *(order-based mutation).* Определенная таким образом мутация заключается в случайном выборе двух позиций в хромосоме и обмене значений генов на этих позициях. Например, после мутации хромосомы [3 5 1 4 7 2 6] на выбранных позициях 2 и 5 будет получена хромосома [3714526]. Количество обменов возрастает или снижается пропорционально увеличению или уменьшению показателя мутации.
Этот способ мутации отличается тем, что в результате ее выполнения формируется новая хромосома с измененной последовательностью генов. Такая мутация применяется для поиска наилучшей перестановки параметров задачи.

Пример 5.9

Этот пример демонстрирует применение генетического алгоритма для сортировки списка имен в алфавитном порядке.

Для упрощения будем рассматривать очень короткий список из семи имен, начинающихся буквами J, M, B, R, S, H, F. Таким образом, наилучшее решение ищется в пространстве решений, состоящем из 7!=5040 возможных перестановок семи элементов. Наилучшее решение очевидно - это B, F, H, J, M, R, S. На этом простейшем примере познакомимся с тем, как генетический алгоритм решает задачи этого типа. Припишем каждому имени, включенному в исходный (несортированный) список, порядковый номер так, как это сделано в первом столбце на рис. 5.49.

Исходный список (несортированный) №	Имена	Одно из возможных решений			
1	J	P_1	3	B	$g_{21}=0$
2	M	P_2	7	F	$g_{31}=0,\ g_{32}=0$
3	B	P_3	4	R	$g_{41}=0,\ g_{42}=0,\ g_{43}=1$
4	R	P_4	6	H	$g_{51}=0,\ g_{52}=0,\ g_{53}=1,\ g_{54}=0$
5	S	P_5	1	J	$g_{61}=0,\ g_{62}=0,\ g_{63}=1,\ g_{64}=0,\ g_{65}=0$
6	H	P_6	2	M	$g_{72}=0,\ g_{72}=0,\ g_{73}=0,\ g_{74}=0,\ g_{75}=0,\ g_{76}=0$
7	F	P_7	5	S	

$G=[000001001000100000000]$

Значение функции приспособленности = 3

Рис. 5.49. Одно из допустимых решений задачи из примера 5.9.

Допустимое решение, представленное на рисунке - это B, F, R, H, J, M, S. Приведенным первым буквам имен предшествуют соответствующие им порядковые номера из первого столбца. Последовательность этих номеров (на рисунке они вписаны в клетки) идентифицирует данное решение. На рис. 5.50 таким же образом представлено наилучшее решение, определяемое последовательностью 3, 7, 6, 1,2,4,5.

Исходный список (несортированный)		Наилучшее решение	
№	Имена		
1	J	P_1 [3] B	$g_{21}=0$
2	M	P_2 [7] F	$g_{31}=0,\ g_{32}=0$
3	B	P_3 [6] H	$g_{41}=0,\ g_{42}=0,\ g_{43}=0$
4	R	P_4 [1] J	$g_{51}=0,\ g_{52}=0,\ g_{53}=0,\ g_{54}=0$
5	S	P_5 [2] M	$g_{61}=0,\ g_{62}=0,\ g_{63}=0,\ g_{64}=0,\ g_{65}=0$
6	H	P_6 [4] R	$g_{71}=0,\ g_{72}=0,\ g_{73}=0,\ g_{74}=0,\ g_{75}=0,\ g_{76}=0$
7	F	P_7 [5] S	

G = [000000000000000000000]

Значение функции приспособленности = 0

Рис.5.50. Наилучшее решение задачи из примера 5.9.

Рассматриваемая задача имеет семь переменных (параметров задачи). Обозначим их P_1, P_2, ..., P_7. Каждая из этих переменных может принимать целые значения от 1 до 7. На рис. 5.49 показана одна из особей популяции, для которой значения конкретных переменных (параметров задачи) равны 3, 7, 4, 6, 1, 2, 5, а наилучшее решение на рис. 5.50 характеризуется значениями этих переменных 3, 7, 6, 1, 2, 4, 5.

Приведенные последовательности чисел рассматриваются как аллели хромосом, представляющих соответствующие особи. Для каждой особи, входящей в популяцию, при выполнении генетического алгоритма рассчитывается значение функции приспособленности и на этой основе выбираются наилучшие и наихудшие особи.

Прежде чем определить функцию приспособленности для рассматриваемой задачи, введем ряд обозначений. Пусть $n(P_i)$ соответствует имени, определенному порядковым номером P_i, i=1,2.....7, и пусть $n(P_i)<n(P_j)$ означает, что имя с порядковым номером P_i - должно предшествовать имени с порядковым номером P_j, т.е. они упорядочиваются по алфавиту. Пусть G обозначает последовательность, составленную из элементов g_{km}, $k = 2.....7$, $m = 1.....k-1$, где

$$g_{km}=\begin{cases}1, & \text{если } n(P_k)<n(P_m)\\ 0 & \text{в противном случае}\end{cases}$$

Очевидно, что если последовательность G состоит из одних нулей, то последовательность имен будет корректной. Поэтому наилучшая особь

должна характеризоваться последовательностью G, все элементы которой равны нулю. Следовательно, функцию приспособленности можно определить как сумму элементов последовательности G, и нас, конечно, будет интересовать минимизация этой функции (которая может принимать целые значения в интервале от 0 до 21). Заметим, что если бы элемент g_{km} принимал нулевое значение при $n(P_k)<n(P_m)$ и значение 1 в противном случае, то следовало бы максимизировать количество единиц в последовательности G. Теперь перейдем к интерпретации функции приспособленности из примера 3.4.

Задача, сформулированная в примере 5.9, решается с помощью программы **Evolver**, размерность популяции принята равной 10, показатель скрещивания равен 0,5, показатель мутации равен 0. При $t = 10$, что соответствует первой популяции классического генетического алгоритма, получена популяция, представленная на рис. 5.51.

1	15	3	5	1	4	7	2	6
2	13	5	3	1	4	2	7	6
3	13	2	6	1	5	4	3	7
4	11	3	4	5	1	7	2	6
5	11	7	2	6	5	4	1	3
6	11	1	7	7	4	5	6	3
7	11	7	4	1	5	6	3	2
8	9	6	4	1	3	2	7	5
9	7	3	5	1	7	6	2	4
10	5	3	6	1	4	2	7	5

Рис.5.51. Популяция особей в генетическом алгоритме программы **Evolver** для задачи из примера 5.9.

Первый столбец определяет последовательность хромосом в популяции (от «наихудшей» к «наилучшей»). Во втором приведены значения функции принадлежности каждой хромосомы. В третьем столбце записаны сами хромосомы, состоящие из семи генов. Каждая из этих хромосом представляет собой перестановку натуральных чисел от 1 до 7. Первое значение функции приспособленности, очерченное прямоугольником и равное 15, относится к хромосоме, исключенной из предыдущей популяции. Вместо нее вводится хромосома [3514726],

полученная в результате скрещивания хромосом [3 5 1 7 6 2 4] и [3 4 5 1 7 2 6], присутствующих на рис. 3.131.
Последняя хромосома, имеющая функцию приспособленности, равную пяти, после 10 «тактов» является «наилучшей на данный момент». При продолжении выполнения алгоритма можно получить наилучшую хромосому [3 7 6 1 2 4 5] с функцией приспособленности, равной 0. Эта оптимальная хромосома представлена на рис. 5.50. Заметим, что в рассматриваемом примере функция приспособленности интерпретируется как погрешность, которую, конечно же, следует минимизировать. Для наилучшего решения эта погрешность обязана быть равной 0.
Пример 5.9 легко обобщить на любые перечни имен или названий, подлежащих сортировке в алфавитном порядке (либо в соответствии с другим ключом). Задачи такого типа можно решать с помощью эволюционного алгоритма программы **Evolver.** Эта программа также позволяет решить известную из литературы задачу о коммивояжере, имеющую гораздо более сложную структуру, чем рассмотренная задача из примера 5.9.

5.3. Эволюционные алгоритмы в нейронных сетях

Объединение генетических алгоритмов и нейронных сетей известно в литературе под аббревиатурой COGANN *(Combinations of Genetic Algorithms and Neural Networks).* Это объединение может быть вспомогательным *(supportive)* либо равноправным *(collaborative).* Вспомогательное объединение двух методов означает, что они применяются последовательно один за другим, причем один из них служит для подготовки данных, используемых при реализации второго метода. При равноправном объединении оба метода применяются одновременно. Классификация этих типов объединений генетических алгоритмов и нейронных сетей представлена в табл. 5.1.

Таблица 5.1. Объединение генетических алгоритмов и нейронных сетей

Вид объединения	Характеристика объединения	Примеры использования
	Генетические алгоритмы и нейронные сети независимо применяются для решения одной и той же задачи	Однонаправленные нейронные сети, сети Кохонена с самоорганизацией и генетические алгоритмы в задачах классификации
Вспомогательное	Нейронные сети для обеспечения генетических алгоритмов	Формирование исходной популяции для генетического алгоритма
	Генетические алгоритмы для обеспечения нейронных сетей	Анализ нейронных сетей
		Подбор параметров либо преобразование пространства параметров
		Подбор параметров либо правила обучения (эволюция правил обучения)
Равноправное	Генетические алгоритмы для обучения нейронных сетей	Эволюционное обучение сети (эволюция весов связей)
	Генетические алгоритмы для выбора топологии нейронной сети	Эволюционный подбор топологии сети (эволюция сетевой архитектуры)
	Системы, объединяющие адаптивные стратегии генетических алгоритмов и нейронных сетей	Нейронные сети для решения оптимизационных задач с применением генетического алгоритма для подбора весов сети
		Реализация генетического алгоритма с помощью нейронной сети
		Применение нейронной сети для реализации оператора скрещивания в генетическом алгоритме

В последующей части настоящего раздела будут обсуждаться конкретные комбинации, отраженные в приводимой таблице. Рисунки 5.52 – 5.54 иллюстрируют различные подходы к решению задач, рассматриваемые как вспомогательные объединения генетических алгоритмов и нейронных сетей.

Необходимо отметить, что термин «генетические алгоритмы» применяется здесь в более широком смысле, чем классический генетический алгоритм.

5.3.1. Независимое применение генетических алгоритмов и нейронных сетей

Генетические алгоритмы и нейронные сети могут независимо применяться для решения одной и той же задачи. Этот подход иллюстрируется на рис. 5.52.

Рис.5.52. Генетический алгоритм и нейронная сеть независимо применяются для решения одной и той же задачи.

Например, описаны независимые применения нейронных сетей, генетических алгоритмов и алгоритма KNN «ближайший сосед» *(К - means nearest neighbour)* для решения задач классификации. В литературе проведится сравнение трехслойной однонаправленной нейронной сети с обучением по методу обратного распространения ошибки (обучение с учителем), сети Кохонена с самоорганизацией (обучение без учителя), системы классификации, основанной на генетическом алгоритме, а также алгоритма KNN «ближайший сосед». Авторы ряда работ считают независимое применение этих методов для решения задачи автоматической классификации результатов ЭМГ

(электромиография - регистрация электрической активности мышц) вспомогательным объединением.
Известны и другие работы, в которых сравниваются возможности применения различных методов (в частности, генетических алгоритмов и нейронных сетей) для решения одних и тех же задач. Примером задачи, которую можно решить с помощью как нейронной сети, так и генетического алгоритма, может служить задача о коммивояжере.

5.3.2. Нейронные сети для поддержки генетических алгоритмов

Большинство исследователей изучали возможности применения генетических алгоритмов для обеспечения работы нейронных сетей. К немногочисленным обратным случаям относится гибридная система, предназначенная для решения задачи трассировки, которая классифицируется как пример вспомогательного объединения нейронных сетей и генетических алгоритмов. В этой системе генетический алгоритм используется в качестве оптимизационной процедуры, предназначенной для нахождения кратчайшего пути. Нейронная сеть применяется при формировании исходной популяции для генетического алгоритма. Этот подход схематичиески иллюстрируется на рис. 5.53.

Рис. 5.53. Вспомогательное объединение нейронной сети с генетическим алгоритмом.

5.3.3. Генетические алгоритмы для поддержки нейронных сетей

Подход, основанный на использовании генетического алгоритма для обеспечения работы нейронной сети, схематически представлен на рис. 5.54.

Рис. 5.54. Вспомогательное объединение генетического алгоритма с нейронной сетью.

Известно множество работ, посвященных подобному объединению рассматриваемых методов. Можно выделить три области проблем:
- применение генетического алгоритма для подбора параметров либо преобразования пространства параметров, используемых нейронной сетью для классификации;
- применение генетического алгоритма для подбора правила обучения либо параметров, управляющих обучением нейронной сети;
- применение генетического алгоритма для анализа нейронной сети.

Две первые области приложения генетических алгоритмов в нейронных сетях, вообще говоря, позволяют улучшать функционирование сетей (т.е. решают проблему синтеза), тогда как третья служит для анализа их функционирования. Начнем обсуждение с последней позиции.

Анализ нейронных сетей. Некоторые исследователи применяли генетические алгоритмы в качестве вспомогательного инструмента для выяснения закономерностей функционирования нейронных сетей либо анализа эффективности их работы. Генетический алгоритм

использовался для построения «инструментальной системы», облегчающей понимание функционирования сети - попросту говоря, для выяснения, что и почему делает сеть. Такое понимание необходимо для того, чтобы нейросетевой классификатор не воспринимался в качестве «черного ящика», который формирует ответ неким таинственным образом, и чтобы решения по классификации объектов были объяснимыми. Подобный «инструментарий» *(explanation facilities)* используется в большинстве экспертных систем. Построение этих инструментов для их применения в нейронных сетях считается более масштабной проблемой, относящейся к анализу сетей. Генетический алгоритм применялся для построения так называемых кодовых векторов *(codebook vectors),* представляющих собой входные сигналы, при которых функция активации конкретного выходного нейрона сети принимает максимальное или близкое к нему значение. Входные векторы представлялись в хромосомах множеством вещественных чисел от 0,0 до 1,0. Анализировалась нейронная сеть, предназначенная для решения задачи классификации. Аналогичный подход применялся для сети ART1 (частного случая ART с двоичными входными сигналами). С помощью генетического алгоритма также проводился анализ нейронной сети, используемой в качестве модели ассоциативного запоминающего устройства. Приведенные примеры характеризуют вспомогательное объединение генетических алгоритмов и нейронных сетей, хотя и не могут считаться типичными по отношению к схеме, представленной на рис. 5.54.

Подбор параметров либо преобразование пространства параметров. Генетический алгоритм используется при подготовке данных для нейронной сети, играющей роль классификатора. Эта подготовка может выполняться путем преобразования пространства параметров либо выделением некоторого подпространства, содержащего необходимые параметры.

Первый из этих методов, так называемое преобразование пространства параметров, применяется чаще всего в алгоритмах типа «ближайший сосед», хотя известны также его приложения в нейросетевых классификаторах. Второй подход заключается в выделении подмножества учитываемых параметров. Оказывается, что ограничение множества параметров часто улучшает функционирование нейронной сети в качестве классификатора и, к тому же, сокращает объемы вычислений. Подобное ограничение множества учитываемых нейронной сетью параметров применялось, в частности, для контроля сценариев происшествий на ядерных объектах, а также для

распознавания китайских иероглифов. Известны и другие примеры подготовки данных для нейронных сетей при помощи генетических алгоритмов.
Подбор параметров и правил обучения. Генетический алгоритм также применяется для подбора параметров обучения - чаще всего скорости обучения (learning rate) и так называемого момента для алгоритма обратного распространения ошибки . Такое адаптивное уточнение параметров алгоритма обратного распространения (они кодируются в хромосомах) в результате эволюции может рассматриваться как первая попытка эволюционной модификации правил обучения. Вместо непосредственного применения генетического алгоритма для подбора параметров обучения развивается эволюционный подход, направленный на построение оптимального правила (алгоритма) обучения.
Эволюция правил обучения будет представлена дальше.
Заметим, что эволюционная концепция уже может рассматриваться как переход от вспомогательного к равноправному объединению генетического алгоритма и нейронных сетей.

5.3.4. Применение генетических алгоритмов для обучения нейронных сетей

Мысль о том, что нейронные сети могут обучаться с помощью генетического алгоритма, высказывалась различными исследователями. Первые работы на эту тему касались применения генетического алгоритма в качестве метода обучения небольших однонаправленных нейронных сетей, но в последующем было реализовано применение этого алгоритма для сетей с большей размерностью.
Как правило, задача заключается в оптимизации весов нейронной сети, имеющей априори заданную топологию. Веса кодируются в виде двоичных последовательностей (хромосом). Каждая особь популяции характеризуется полным множеством весов нейронной сети. Оценка приспособленности особей определяется функцией приспособленности, задаваемой в виде суммы квадратов погрешностей, т.е. разностей между ожидаемыми (эталонными) и фактически получаемыми значениями на выходе сети для различных входных данных.
Приведем два важнейших аргумента в пользу применения генетических алгоритмов для оптимизации весов нейронной сети. Прежде всего, генетические алгоритмы обеспечивают глобальный

просмотр пространства весов и позволяют избегать локальные минимумы. Кроме того, они могут использоваться в задачах, для которых информацию о градиентах получить очень сложно либо она оказывается слишком дорогостоящей.
Эволюционному обучению нейронных сетей, т.е. эволюции весов связей, посвящен п.5.3.7.1.

5.3.5. Генетические алгоритмы для выбора топологии нейронных сетей

В качестве наиболее очевидного способа объединения генетического алгоритма с нейронной сетью интуитивно воспринимается попытка закодировать в генотипе топологию объекта (в рассматриваемом случае - нейронной сети) с указанием количества нейронов и связей между ними при последующем определении весов сети с помощью любого известного метода.
Проектирование оптимальной топологии нейронной сети может быть представлено в виде поиска такой архитектуры, которая обеспечивает наилучшее (относительно выбранного критерия) решение конкретной задачи. Такой подход предполагает перебор пространства архитектур, составленного из всех возможных вариантов, и выбор точки этого пространства, наилучшей относительно заданного критерия оптимальности.
С учетом достоинств эволюционного проектирования архитектуры в последние годы было выполнено большое количество исследований, в которых основное внимание уделялось эволюции соединений нейронной сети, т.е. количества нейронов и топологии связей между ними. Лишь в некоторых работах рассматривалась эволюция функций переходов, хотя эти функции считаются важным элементом архитектуры и оказывают существенное влияние на функционирование нейронной сети.
Также, как и в случае эволюционного обучения, первый шаг эволюционного проектирования архитектуры заключается в формировании исходного множества рассматриваемых вариантов. Типовой цикл эволюции архитектур представлен в п. 5.2.7.2.

5.3.6. Адаптивные взаимодействующие системы

К равноправному объединению генетических алгоритмов и нейронных сетей следует отнести комбинацию адаптивных стратегий обоих методов, составляющую единую адаптивную систему. Можно привести три примера систем такого типа. Первый из них - это нейронная сеть для оптимизационной задачи с генетическим алгоритмом для определения весов сети. Второй пример относится к реализации генетического алгоритма с помощью нейронной сети. В этом случае нейронные подсистемы применяются для выполнения генетических операций репродукции и скрещивания. В третьем примере, несколько похожем на предыдущий, нейронная сеть также применяется в качестве оператора скрещивания в генетическом алгоритме, предназначенном для решения оптимизационных задач. Представленные примеры касаются такого равноправного объединения генетических алгоритмов и нейронных сетей, которое в результате позволяет получить более эффективный алгоритм, объединяющий лучшие качества обоих методов.

5.3.7. Типовой цикл эволюции

Как только определенный вид эволюции вводится в искусственную нейронную сеть, сразу возникает потребность в соответствующей ему схеме хромосомного представления данных, т.е. должен быть создан способ генетического кодирования особей популяции. Разработка способа кодирования считается первым этапом такого эволюционного подхода, наряду с которым типовой процесс эволюции включает следующие шаги:

- декодирование;
- обучение;
- оценивание приспособленности;
- репродукция;
- формирование нового поколения.

Приведенная на рис. 3.12 блок-схема сохраняет свою актуальность, поскольку (как уже упоминалось в разд. 3.18) она отображает и классический генетический алгоритм, и так называемые эволюционные программы, которые основаны на генетическом подходе и обобщают его принципы. Следовательно, этой универсальной блок-схеме соответствуют различные эволюционные алгоритмы, и в каждом из

них в первую очередь должна быть сгенерирована исходная популяция хромосом. По аналогии с классическим генетическим алгоритмом инициализация (т.е. формирование этой исходной популяции) заключается в случайном выборе требуемого количества включаемых в нее хромосом, что предполагает соответствующее генетическое кодирование каждой особи. В классическом генетическом алгоритме хромосомы представляются только двоичными последовательностями. При эволюционном подходе выбор способа кодирования представляет собой важную и актуальную задачу.

Далее в соответствии с типовым циклом эволюции следует декодировать каждую особь (хромосому) исходной или текущей популяции для того, чтобы получить множество решений (фенотипов) данной задачи. В случае эволюции весов, архитектур и/или правил обучения фенотипы представляют соответственно множества весов, архитектур и правил обучения.

Впоследствии согласно генетическому алгоритму рассчитываются значения функции приспособленности особей исходной (или текущей) популяции. При нейросетевом подходе после декодирования хромосом получается множество нейронных сетей, для которых степень приспособленности определяется по результатам обучения этих сетей.

При реализации типового цикла эволюции необходимо сконструировать множество соответствующих нейронных сетей (фенотипов):

- сети с фиксированной архитектурой и множеством закодированных хромосомами весов - в случае эволюции весов;
- сети с закодированной хромосомами архитектурой - в случае эволюции архитектуры;
- сети со случайно сгенерированными архитектурами и начальными весами - в случае эволюции правил обучения.

После обучения оценивается приспособленность каждой особи, входящей в текущую популяцию. Заметим, что также как и в примере максимизации функции, для оценивания приспособленности хромосом необходимо их вначале декодировать и лишь затем рассчитать значения функции приспособленности особей по их фенотипам.

Следующий шаг генетического алгоритма - это селекция хромосом. Выбираются хромосомы, подлежащие репродукции, т.е. формируется родительский пул, особи которого в результате применения генетических операторов сформируют популяцию потомков. Селекция может быть основана на методе рулетки или любом другом, например, по алгоритму Уитли (Whitley). Согласно этим методам селекция

производится с вероятностью, пропорциональной приспособленности хромосом, либо согласно их рангу (при использовании рангового метода). Под репродукцией в данном случае понимается процесс отбора (селекции) и копирования (размножения) хромосом для формирования из них переходной популяции (родительского пула), особи которой будут подвергаться воздействию генетических операторов скрещивания, мутации и, возможно, инверсии.

Применение генетических операторов с выбранным методом селекции хромосом происходит аналогично классическому генетическому алгоритму, причем эти операторы могут отличаться от скрещивания и мутации базового алгоритма. Как отмечалось ранее, для конкретной задачи генетические операторы могут определяться в индивидуальном порядке.

Также как и в классическом генетическом алгоритме, в результате применения генетических операторов с выбранным методом селекции хромосом формируется новая популяция особей (потомков). Последующие шаги алгоритма повторяются для очередной популяции вплоть до выполнения условия завершения генетического алгоритма. На каждой итерации формируется новое поколение потомков.

Наилучшая особь из последнего поколения считается искомым решением данной задачи. Таким образом получается наилучшее множество весов, наилучшая архитектура либо наилучшее правило обучения.

5.3.7.1. Эволюция весов связей

Эволюционный подход к обучению нейронных сетей состоит из двух основных этапов. Как указывалось ранее, первый из них - это выбор соответствующей схемы представления весов связей. Он заключается в принятии решения - можно ли кодировать эти веса двоичными последовательностями или требуется какая-то другая форма. На втором этапе уже осуществляется сам процесс эволюции, основанный на генетическом алгоритме.

После выбора схемы хромосомного представления генетический алгоритм применяется к популяции особей (хромосом, содержащих закодированное множество весов нейронной сети) с реализацией типового цикла эволюции, состоящего из четырех шагов.

1) Декодирование каждой особи (хромосомы) текущего поколения для восстановления множества весов и конструирование соответствующей

этому множеству нейронной сети с априорно заданной архитектурой и правилом обучения.
2) Расчет общей среднеквадратичной погрешности между фактическими и заданными значениями на всех выходах сети при подаче на ее входы обучающих образов. Эта погрешность определяет приспособленность особи (сконструированной сети); в зависимости от вида сети функция приспособленности может быть задана и другим образом.
3) Репродукция особей с вероятностью, соответствующей их приспособленности, либо согласно их рангу (в зависимости от способа селекции - например, по методу рулетки или ранговому методу).
4) Применение генетических операторов - таких как скрещивание, мутация и/или инверсия для получения нового поколения.
Блок-схема, иллюстрирующая эволюцию весов, представлена
на рис. 5.55.

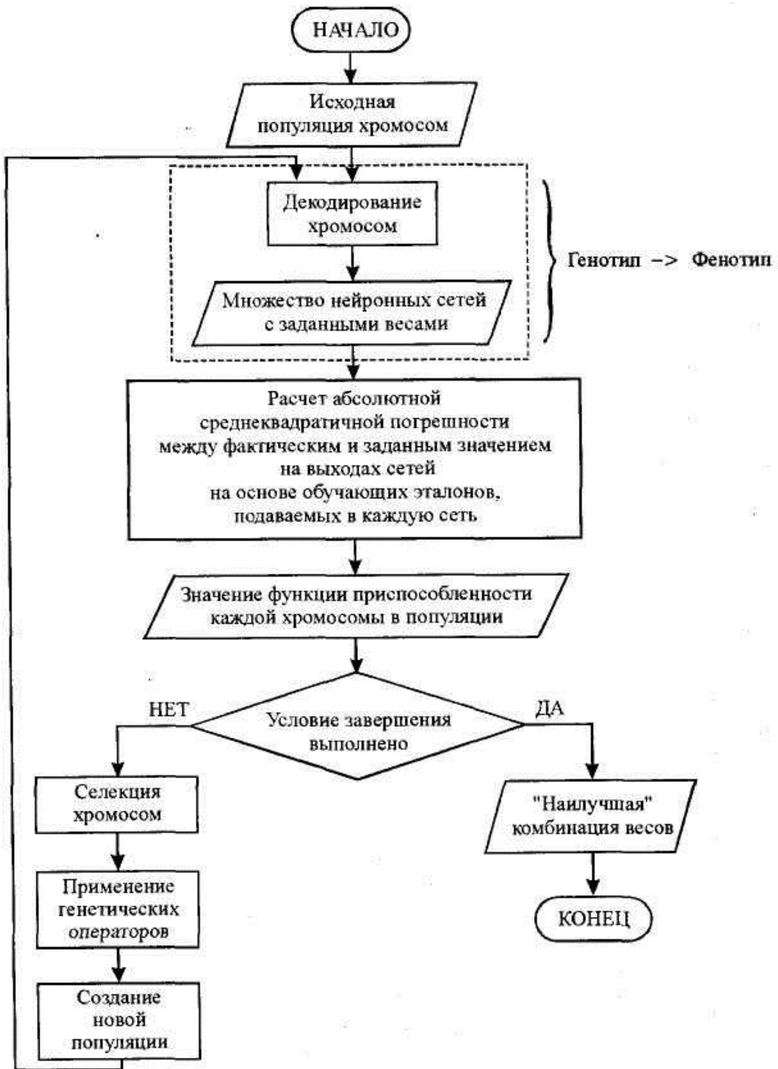


Рис. 5.55. Блок-схема генетического алгоритма поиска наилучшего набора весов нейронной сети (случай эволюции весов).

В соответствии с этой схемой были рассчитаны веса для нейронной сети, реализующей систему XOR с помощью программы **Evolver** и с

помощью программы **FlexTool**. В соответствии с первым этапом типового цикла эволюции априорно задаются и остаются неизменными архитектура сети, определяющая количество слоев, число нейронов в каждом слое и топологию межнейронных связей, а также правило обучения сети. Приспособленность каждой особи (генотипа) оценивается значением среднеквадратичной погрешности, рассчитанной по соответствующей этой особи нейронной сети (фенотипу).

В представленном процессе эволюционного обучения реализуется режим так называемого пакетного обучения *(batch training mode),* при котором значения весов изменяются только после предъявления сети всех обучающих образов. Такой прием отличается от применяемого в большинстве последовательных алгоритмов обучения - например, в методе обратного распространения ошибки веса уточняются после предъявления сети каждой обучающей выборки.

Рассмотрим более подробно первый этап эволюционного подхода к обучению, связанный с фиксацией схемы представления весов. Как уже отмечалось, необходимо выбрать между бинарным представлением и кодированием весов действительными числами. Помимо традиционного двоичного кода, может применяться код Грея, логарифмическое кодирование либо другие более сложные формы записи данных. В роли ограничителя выступает требуемая точность представления значений весов. Если для записи каждого веса используется слишком мало битов, то обучение может продолжаться слишком долго и не принести никакого эффекта, поскольку точность аппроксимации отдельных комбинаций действительных значений весов дискретными значениями часто оказывается недостаточной. С другой стороны, если используется слишком много битов, то двоичные последовательности, представляющие нейронные сети большой размерности, оказываются очень длинными, что сильно удлиняет процесс эволюции и делает эволюционный подход к обучению нерациональным с практической точки зрения. Вопрос оптимизации количества битов для представления конкретных весов все еще остается открытым.

Для устранения недостатков схемы двоичного представления данных было предложено задавать значения весов действительными числами, точнее - каждый вес описывать отдельным действительным числом. Такой способ кодирования реализован, в частности, в программе **Evolver**. Он использовался при решении примеров.

Стандартные генетические операторы, разработанные для схемы двоичного представления данных, могут применяться и в случае задания весов двоичными числами, однако для большей эффективности эволюционного алгоритма и ускорения его выполнения созданы специальные генетические операторы.

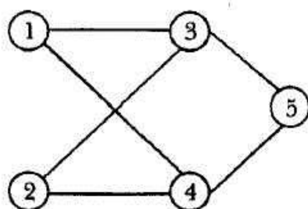
5.3.7.2. Эволюция архитектуры сети

В п.5.3.7.1 при рассмотрении эволюционного обучения нейронных сетей предполагалось, что архитектура сети задается априорно и не изменяется в процессе эволюции весов. Однако сохраняет актуальность вопрос - как выбрать архитектуру сети? Известно, что архитектура оказывает решающее влияние на весь процесс обработки информации нейронной сетью. К сожалению, чаще всего она подбирается экспертами методом проб и ошибок. В таких условиях способ оптимального (или почти оптимального) проектирования архитектуры нейронной сети для конкретной задачи оказался бы очень полезным. Один из возможных подходов заключается в эволюционном формировании архитектуры с применением генетического алгоритма.

Также как и в случае эволюционного обучения, на первом этапе эволюционного проектирования архитектуры принимается решение относительно соответствующей формы ее описания. Однако в данной ситуации проблема не связана с выбором между двоичным и вещественным представлением (т.е. действительными числами), поскольку речь может идти только о дискретных значениях. Необходимо выбрать более общую концептуальную структуру представления данных, например, в форме матриц, графов и т.п. Ключевой вопрос состоит в принятии решения о количестве информации об архитектуре сети, которая должна кодироваться соответствующей схемой. С одной стороны, полная информация об архитектуре может непосредственно кодироваться в виде двоичных последовательностей, т.е. каждая связь и каждый узел (нейрон) прямо специфицируется определенным количеством битов. Такой способ представления называется *схемой непосредственного кодирования.* С другой стороны, могут представляться только важнейшие параметры или свойства архитектуры - такие как количество узлов (нейронов), количество связей и вид переходной функции нейрона. Этот способ представления называется *схемой косвенного кодирования.* Существуют и другие названия указанных способов представления данных, например, вместо

«непосредственного кодирования» встречается термин *сильная схема спецификации,* а вместо «косвенного кодирования» - *слабая схема классификации*.

Схема непосредственного кодирования означает, что каждая связь нейронной сети непосредственно задается его двоичным представлением. Матрица C размерностью n×n, $C = [c_{ij}]_{n\times n}$ может представлять связи нейронной сети, состоящей из n узлов (нейронов), причем значение c_{ij} определяет наличие или отсутствие связи между i-м и j-м нейронами. Если $c_{ij} = 1$, то связь существует, если $c_{ij} = 0$, то связь отсутствует. При таком подходе двоичная последовательность (хромосома), представляющая связи нейронной сети, имеет вид комбинации строк (или столбцов) матрицы C. Пример рассматриваемого способа кодирования для $n = 5$ приведен на рис. 5.56.



Нейронная сеть

Матрица связей :

$$C = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Хромосома :

[0011000110000010000100000]

Рис. 5.56. Пример непосредственного кодирования матрицы связей для нейронной сети.

Если значение n обозначает количество нейронов в сети, то связи между этими нейронами будут представляться двоичной последовательностью, имеющей длину n^2. Очевидный недостаток такого способа кодирования заключается в стремительном увеличении длины генотипа при расширении нейронной сети. Однако в обсуждаемую схему представления можно легко внести ограничения, которые сократят длину хромосом. В частности, могут приниматься во внимание только однонаправленные связи, что позволит учитывать только те элементы матрицы C, которые задают связи данного узла (нейрона) со следующим узлом. В этом случае хромосома для примера на рис. 5.56 будет иметь вид 0110110011.

Схема непосредственного кодирования может одновременно применяться для определения как связей, так и их весов. Этот способ кодирования применяется, главным образом, для небольших нейронных сетей.

Схема косвенного кодирования - это способ сокращения длины описания связей, который заключается в кодировании только наиболее важных свойств, но не каждой связи нейронной сети. По этой причине заметным достоинством схемы косвенного кодирования становится компактное представление связей. Такая схема выглядит более обоснованной с биологической точки зрения. Согласно современной нейрологии, невозможно прямо и независимо описать закодированной в хромосомах генетической информацией всю нервную систему. Такой вывод следует, например, из факта, что генотип человека состоит из гораздо меньшего количества генов, чем число нейронов в его мозге.

Известны различные методы косвенного кодирования.

Второй этап эволюционного проектирования архитектуры нейронной сети состоит (в соответствии с типовым циклом эволюции) из следующих шагов:

1) Декодирование каждой особи текущей популяции для описания архитектуры нейронной сети.

2) Обучение каждой нейронной сети с архитектурой, полученной на первом шаге, с помощью заранее заданного правила (некоторые его параметры могут адаптивно уточняться в процессе обучения). Обучение должно начинаться при различных случайно выбираемых начальных значениях весов и (при необходимости) параметров правила обучения.

3) Оценивание приспособленности каждой особи (закодированной архитектуры) по достигнутым результатам обучения, т.е. по наименьшей целой среднеквадратичной погрешности обучения либо на основе тестирования, если наибольший интерес вызывает способность к обобщению, наименьшая длительность обучения или упрощение архитектуры (например, минимизация количества нейронов и связей между ними).

4) Репродукция особей с вероятностью, соответствующей их приспособленности или рангу в зависимости от используемого метода селекции.

5) Формирование нового поколения в результате применения таких генетических операторов, как скрещивание, мутация и/или инверсия.

Блок-схема, иллюстрирующая эволюцию архитектур, представлена на рис. 5.57.

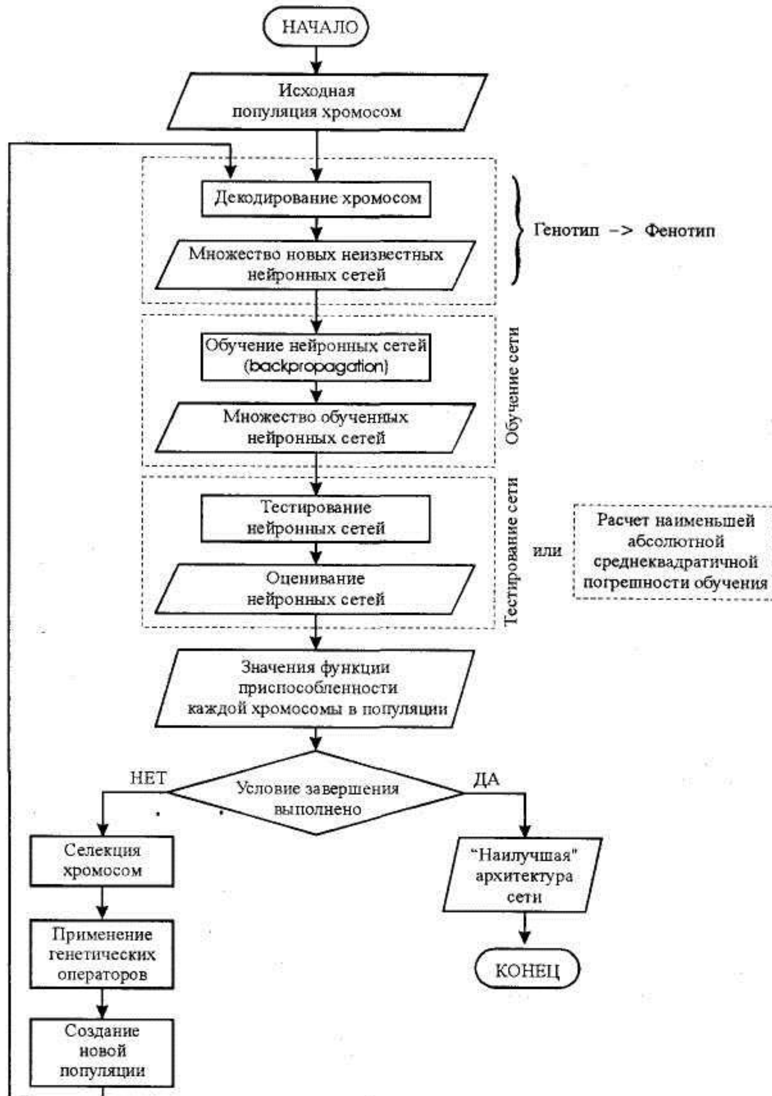


Рис. 5.57. Блок-схема генетического алгоритма для поиска наилучшей архитектуры нейронной сети (случай эволюции архитектур).

Если говорить об обучении сети (шаг 2), то наиболее часто встречается развитие топологии однонаправленных сетей с применением алгоритма обратного распространения ошибки с целью локального обучения. Известны работы, в которых описывается применение генетического алгоритма для одновременной адаптации и весов и топологии. В других исследованиях допускались соединения в пределах одного слоя, обратные связи, а также обучение на основе конкуренции *(competetive learning)* и по Хеббу.

Достоинством эволюционного подхода считается тот факт, что функцию приспособленности можно легко определить специально для эволюции сети со строго определенными свойствами. Например, если для оценивания приспособленности использовать результаты тестирования вместо результатов обучения, то будет получена сеть с лучшей способностью к обобщению.

5.3.7.3. Эволюция правил обучения

Известно, что для различных архитектур и задач обучения требуются различные алгоритмы обучения. Поиск оптимального (или почти оптимального) правила обучения, как правило, происходит с учетом экспертных знаний и часто - методом проб и ошибок. Поэтому весьма перспективным считается развитие автоматических методов оптимизации правил обучения нейронных сетей. Развитие человеческих способностей к обучению от относительно слабых до весьма сильных свидетельствует о потенциальной возможности применения эволюционного подхода в процессе обучения искусственных нейронных сетей.

Схема хромосомного представления в случае эволюции правил обучения должна отражать динамические характеристики. Статические параметры (такие как архитектура или значения весов сети) кодировать значительно проще. Попытка создания универсальной схемы представления, которая позволила бы описывать произвольные виды динамических характеристик нейронной сети, заведомо обречена на неудачу, поскольку предполагает неоправданно большой объем вычислений, требуемых для просмотра всего пространства правил обучения. По этой причине на тип динамических характеристик обычно налагаются определенные ограничения, что позволяет выбрать общую структуру правила обучения. Чаще всего устанавливается, что

для всех связей нейронной сети должно применяться одно и то же правило обучения, которое может быть задано функцией вида

$$\Delta w(t) = \sum_{k=1}^{n} \sum_{i_1, i_2 \cdots i_k = 1}^{n} [\theta_{i_1, i_2 \cdots i_k} \prod_{j=1}^{k} x_{i_j}(t-1)], \ (*)$$

где t - время, Δ_w - приращение веса, x_{i_j} - так называемые локальные переменные, $\theta_{i_1, i_2 \cdots i_k}$ - вещественные коэффициенты.

Главная цель эволюции правил обучения заключается в подборе соответствующих значений коэффициентов $\theta_{i_1, i_2 \cdots i_k}$.

С учетом большого количества компонентов уравнения (*), что может сделать эволюцию слишком медленной и практически неэффективной, часто вводятся дополнительные ограничения, основанные на эвристических посылках.

Представим типовой цикл эволюции правил обучения.

1) Декодирование каждой особи текущей популяции для описания правила обучения, которое будет использоваться в качестве алгоритма обучения нейронных сетей.

2) Формирование множества нейронных сетей со случайно сгенерированными архитектурами и начальными значениями весов, а также оценивание этих сетей с учетом их обучения по правилу, полученному на шаге 1, в категориях точности обучения или тестирования, длительности обучения, сложности архитектуры и т.п.

3) Расчет значения приспособленности каждой особи (закодированного правила обучения) на основе полученной на шаге 2 оценки каждой нейронной сети, что представляет собой своеобразный вид взвешенного усреднения.

4) Репродукция особей с вероятностью, соответствующей их приспособленности или рангу в зависимости от используемого метода селекции.

5) Формирование нового поколения в результате применения таких генетических операторов, как скрещивание, мутация и/или инверсия.

Блок-схема, иллюстрирующая эволюцию архитектур, представлена на рис. 5.58.

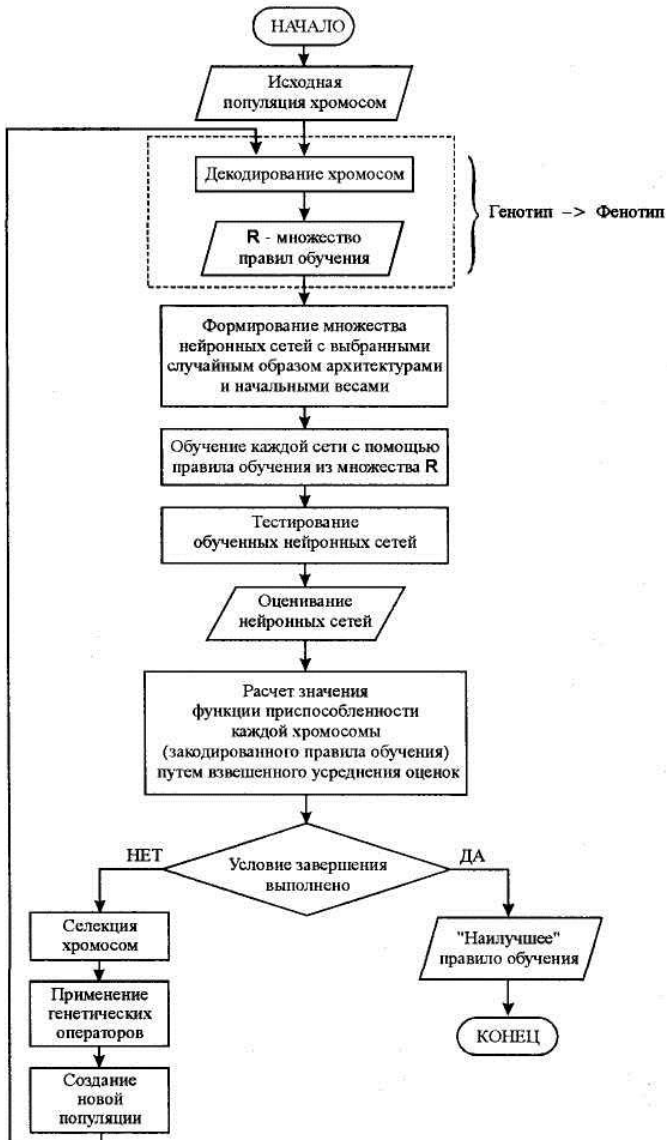


Рис. 5.58. Блок-схема генетического алгоритма для поиска наилучшего правила обучения (случай эволюции правил обучения).

5.4. Примеры моделирования эволюционных алгоритмов в приложении к нейронным сетям

Представленные примеры можно рассматривать как иллюстрацию возможности применения генетического (в частности, эволюционного) алгоритма для подбора весов нейронной сети. Этот алгоритм будет применяться в перечисленных примерах вместо традиционного метода обучения, например, вместо алгоритма обратного распространения ошибки *(backpropagation).* Очень часто применяется так называемый гибридный подход, состоящий в объединении обоих методов. Как правило, вначале при помощи генетического алгоритма находится решение, достаточно близкое к оптимальному, и затем оно рассматривается как отправная точка для традиционного поиска оптимальной точки, например, по методу обратного распространения ошибки.

5.4.1. Программы Evolver и BrainMaker

Метод обратного распространения ошибки применяется для обучения нейронных сетей в программе **BrainMaker**. В примере 3.30 с использованием этой программы тестируется нейронная сеть, реализующая логическую систему XOR со значениями весов, рассчитанными в п. 3.19.1 с помощью программы **Evolver.** При решении примеров 3.31 и 3.32 демонстрируется гибридный подход, т.е. обучение этой же нейронной сети программой **BrainMaker,** но при начальных значениях весов, рассчитанных генетическим алгоритмом программы **Evolver.** В свою очередь, пример 3.33 иллюстрирует обучение нейронной сети, реализующей логическую систему XOR, при использовании только программы **BrainMaker.**

Пример 5.1

Протестировать с помощью программы **BrainMaker** нейронную сеть, реализующую логическую систему XOR (рис. 3.11) с весами, рассчитанными в примере 3.27 генетическим алгоритмом программы **Evolver** и представленными на рис. 5.37.

На рис. 5.59 показаны значения весов, введенные в программу **BrainMaker.** Они сгруппированы в два блока, разделенные пустой строкой. Первый блок содержит веса связей между входным и скрытым слоем так, что для каждого из двух нейронов скрытого слоя при-

ведены веса связей со всеми входами, а последние элементы строк - это w_{10} и w_{20}. Второй блок содержит веса связей между скрытым и выходным слоем, т.е. три веса выходного нейрона, причем последним указан вес w_{30}. В примере использовался интервал значений весов [-7, 8].

Результаты тестирования сети приведены на рис. 5.60-5.63. Можно сделать вывод о хорошей обученности сети. Значения на выходах для четырех пар входных значений практически совпадают с показанными на рис. 5.37. Абсолютная разность между заданным значением d и выходным значением у для каждой пары входов u_1 и u_2 (рис. 5.60-5.63) оказалась меньше 0,025, поэтому сеть может считаться хорошо обученной с толерантностью 0,025 и тем более - с толерантностью 0,1. Понижение порога толерантности до 0,02 означало бы, что эта сеть не считается хорошо обученной и что требуемый уровень обучения не может быть достигнут. Дальнейшее обучение при толерантности, равной или меньшей 0,02, не изменяет значений весов, показанных на рис. 5.59.

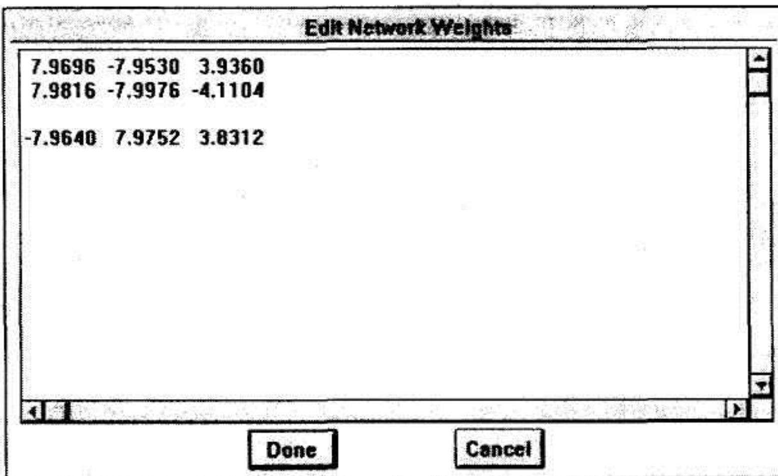


Рис. 5.59. Показанные на рис. 5.37 веса нейронной сети, реализующей систему XOR (рис. 3.11) и подготовленной к тестированию программой **BrainMaker.**

Однако, как будет видно из примера 5.2, уменьшение толерантности до 0,022 принесло бы эффект в виде «дообучения» сети.

Пример 5.2

Обучить с помощью программы **BrainMaker** нейронную сеть, реализующую логическую систему XOR (рис. 3.11) с весами, рассчитанными в примере 3.27 генетическим алгоритмом программы **Evolver** и представленными на рис. 5.36.

Вначале с помощью программы **BrainMaker** была протестирована нейронная сеть, показанная на рис. 3.11. В качестве начальных значений для алгоритма обучения программы **BrainMaker** использовались веса, рассчитанные генетическим алгоритмом программы **Evolver.** Следовательно, это типичный пример гибридного подхода, поскольку генетический алгоритм используется для нахождения начальных значений весов для градиентного алгоритма обратного распространения ошибки *(backpropagation).* Эти начальные значения в формате программы **BrainMaker** представлены на рис. 5.64, а соответствующие результаты тестирования сети - на рис.5.65 – 5.68.

Анализ результатов тестирования свидетельствует о том, что они оказываются хуже, чем показанные на рис. 5.60-5.63, поскольку в предыдущем примере не было ни одной ошибки, а в текущем примере в пяти случаях из 22 сеть дала неверный ответ.

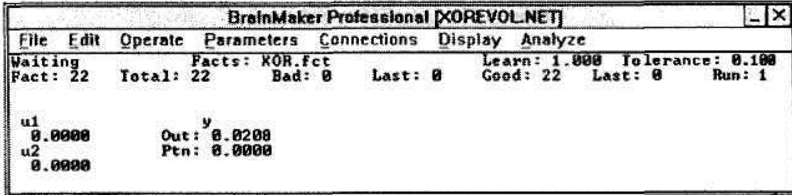


Рис. 5.60. Результат тестирования программой **BrainMaker** нейронной сети из рис. 3.11 с начальными значениями весов, представленными на рис. 5.59, для $u_1 = 0$, $u_2 = 0$ и d = 0.

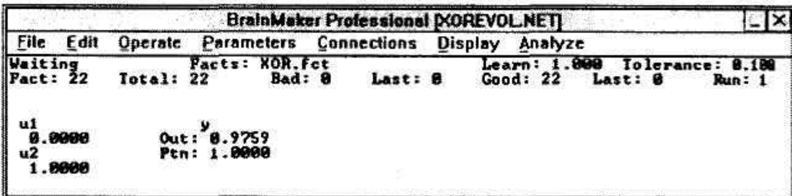


Рис.5.61. Результат тестирования программой **BrainMaker** нейронной сети из рис. 3.11 с начальными значениями весов, представленными на рис. 5.59, для $u_1 = 0$, $u_2 = 1$ и d = 1.

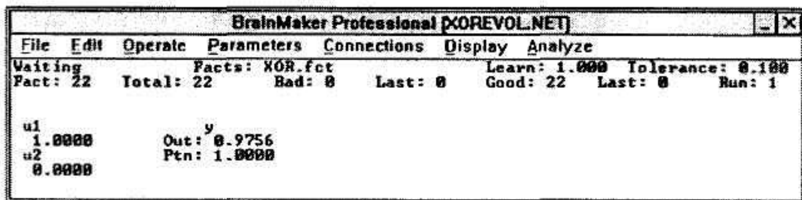


Рис.5.62. Результат тестирования программой **BrainMaker** нейронной сети из рис. 3.11 с начальными значениями весов, представленными на рис. 5.59, для $u_1 = 1$, $u_2 = 0$ и d = 1.

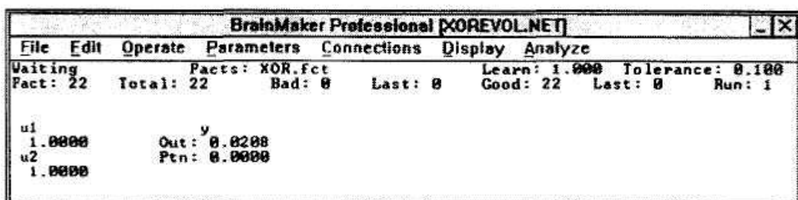


Рис. 5.63. Результат тестирования программой **BrainMaker** нейронной сети из рис. 3.11 с начальными значениями весов, представленными на рис. 5.59, для $u_1 = 1$, $u_2 = 1$ и d = 0.

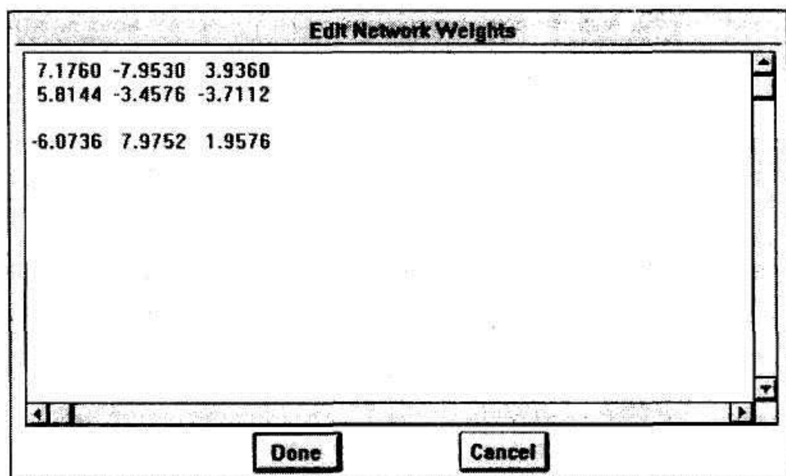


Рис. 5.64. Показанные на рис. 5.593.116 веса нейронной сети, реализующей систему XOR и подготовленной к обучению программой **BrainMaker**

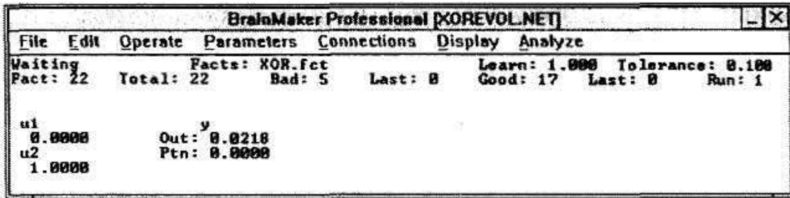


Рис. 5.65. Результат тестирования нейронной сети из рис. 3.11 с начальными значениями весов, представленными на рис. 3.144, для $u_1 = 0$, $u_2 = 0$ и $d = 0$.

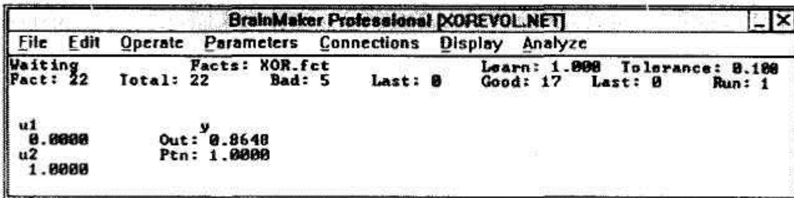


Рис. 5.66. Результат тестирования той же сети для $u_1 = 0$, $u_2 = 1$ и d = 1.

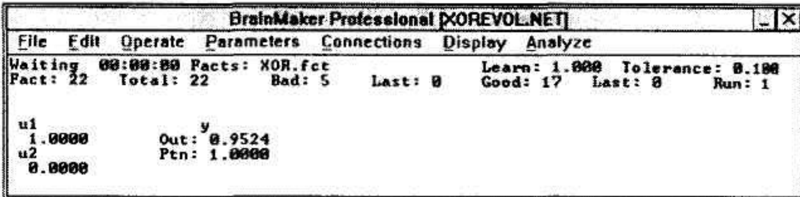


Рис. 5.67. Результат тестирования той же сети для $u_1 = 1$, $u_2 = 0$ и $d = 1$.

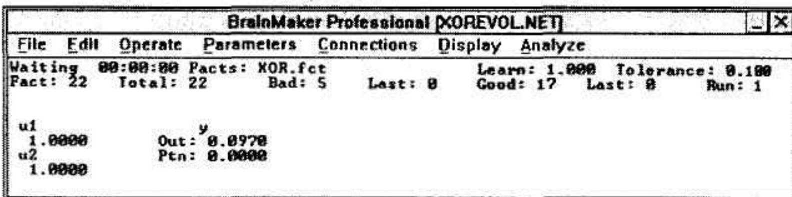


Рис. 5.68. Результат тестирования той же сети для $u_1 = 1$, $u_2 = 1$ и $d = 0$.

Кроме того, видно, что толерантность выходного значения на рис. 5.66 превышает уровень 0,1. Впоследствии сеть подверглась обучению с

помощью программы **BrainMaker** с толерантностью, равной 0,1. Результаты обучения иллюстрирует рис. 5.69.

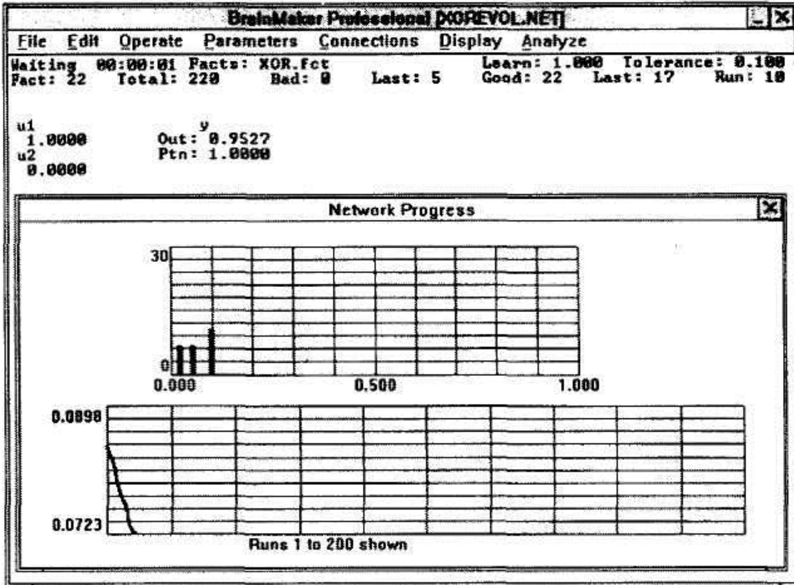


Рис. 5.69. Результат обучения програмой BrainMaker нейронной сети из рис. 3.11 с начальными значениями весов, представленными на рис. 5.64.

На нижнем графике заметно уменьшение среднеквадратичной погрешности RMS *(Root Mean Squared)* при выполнении 10 реализаций *(runs)* алгоритма. Погрешность RMS отличается от погрешности Q, минимизировавшейся в примере 3.27, тем, что

$$Q=\frac{1}{N}\sum_{i=1}^{N}\varepsilon_{i^2}$$

$$RMS=\frac{1}{N}\sqrt{\sum_{i=1}^{N}\varepsilon_{i^2}}$$

где $\varepsilon_i = d_i - y_i$ а значение N в примере 3.27 равно 4.

С учетом различий, связанных с взятием квадратного корня при расчете RMS, можно сравнить эту погрешность со значением Q из примера 3.27. Верхний график на рис. 5.69 представляет собой гис-

тограмму распределения погрешности, рассчитанной как абсолютная разность между заданным d_i и фактическим y_i выходным значением для каждой пары входов u_{1i} и u_{2i}. На горизонтальной оси отложены значения этой погрешности, а на вертикальной оси - количество выходных значений с такой погрешностью. Представлены три уровня погрешности с толерантностью 0,1. В верхней части рис. 5.69 указано значение у для $u_1 = 1$, $u_2 = 0$ и $d = 1$ (для упрощения здесь опущен индекс i). Результаты тестирования этой сети для трех остальных комбинаций входных значений (для системы XOR) приведены на рис. 5.70-5.72.

```
Waiting            Facts: XOR.fct                Learn: 1.000  Tolerance: 0.100
Fact: 22   Total: 220     Bad: 0     Last: 5     Good: 22    Last: 17    Run: 10

 u1               y
  0.0000       Out: 0.0269
 u2            Ptn: 0.0000
  0.0000
```

Рис. 5.70. Результат тестирования программой BrainMaker нейронной сети из рис. 3.2 с начальными значениями весов, представленными на рис. 5.64, с толерантностью 0,1 для $u_1 = 0$, $u_2 = 0$ и $d = 0$.

```
Waiting            Facts: XOR.fct                Learn: 1.000  Tolerance: 0.100
Fact: 22   Total: 220     Bad: 0     Last: 5     Good: 22    Last: 17    Run: 10

 u1               y
  0.0000       Out: 0.9002
 u2            Ptn: 1.0000
  1.0000
```

Рис. 5.71. Результат тестирования той же сети для $u_1 = 0$, $u_2 = 1$ и $d =$ **1.**

```
Waiting            Facts: XOR.fct                Learn: 1.000  Tolerance: 0.100
Fact: 22   Total: 220     Bad: 0     Last: 5     Good: 22    Last: 17    Run: 10

 u1               y
  1.0000       Out: 0.0979
 u2            Ptn: 0.0000
  1.0000
```

Рис. 5.72. Результат тестирования той же сети для $u_1 = 1$, $u_2 = 1$ и $d = 0$.

Абсолютная разность между эталонным (Ptn) и выходным (Out) значениями для рис. 5.69 равна 0,0473, для рис. 5.70 - 0,0269, для рис. 5.71 - 0,0998 и для рис. 5.72 - 0,0979. Из гистограммы на рис. 5.69 следует, что для шести из 22 входных пар u_1, u_2 эта погрешность равна 0,0269, такое же количество входных пар имеет погрешность 0,0473, а для оставшихся входных пар эта погрешность составила около 0,099. Веса обученной таким образом сети показаны на рис. 5.73.

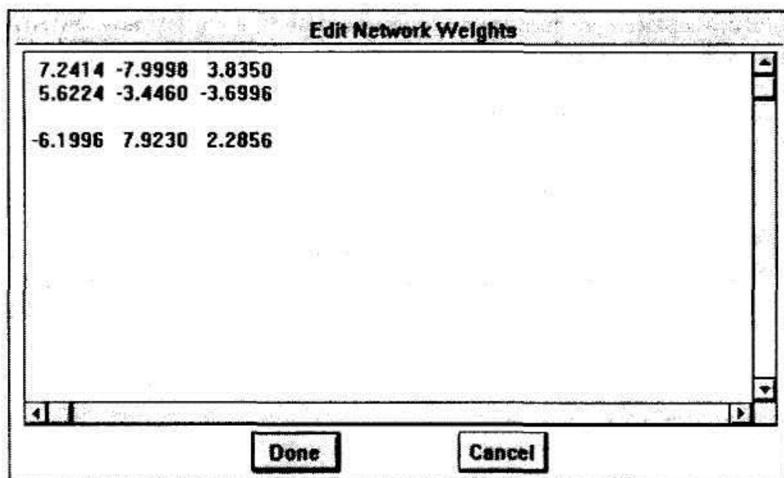


Рис.5.73. Веса нейронной сети, обученной программой **BrainMaker** с толерантностью 0,1

Сеть обучена с толерантностью 0,1. Это означает, что выходные значения должны не более чем на 10 % отличаться от значений 0 и 1 для того, чтобы модель системы XOR была признана корректной. Следовательно, при подаче эталона 0 считаются правильными значения y, меньшие или равные 0,1, а при подаче эталона 1 правильными будут значения y, большие или равные 0,9.

Конечно, показанные на рис. 5.73 значения весов не могут рассматриваться в качестве оптимальных для системы XOR. Поэтому толерантность была уменьшена до 0,025, и процесс обучения продолжился. Полученные результаты приведены на рис. 5.74.

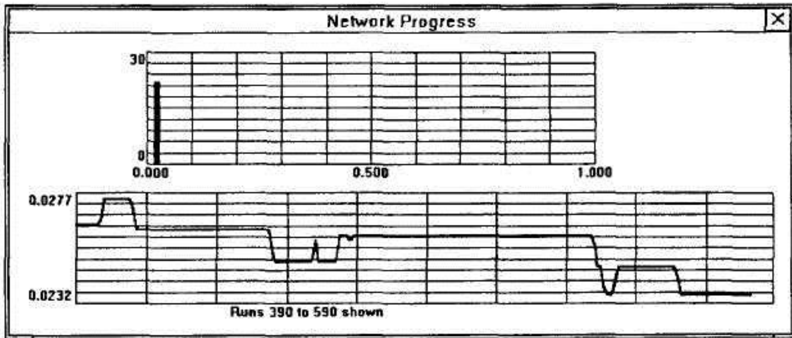


Рис. 5.74. Итоговая фаза последующего обучения программой **BrainMaker** сети с весами, показанными на рис. 3.153, при уровне толерантности 0,025.

Интересно сравнить достигнутое значение погрешности RMS с аналогичным показателем на рис. 5.69. Результаты тестирования обученной сети представлены на рис. 5.75 – 5.78.

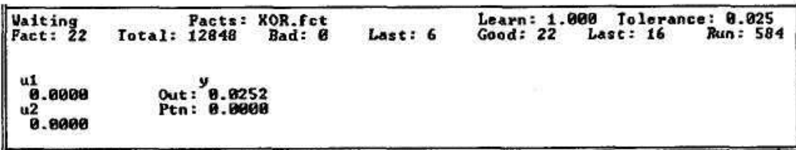


Рис. 5.75. Результат тестирования нейронной сети, обученной программой **BrainMaker** с толерантностью 0,025 (рис. 5.74), для $u_1 = 0$, $u_2 = 0$ и $d = 0$.

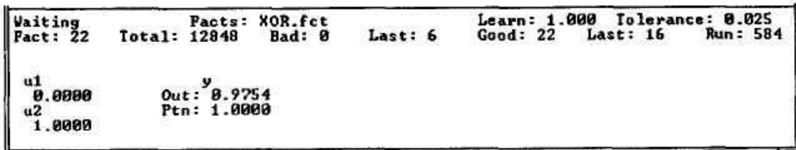


Рис. 5.76. Результат тестирования той же сети для $u_1 = 0$, $u_2 = 1$ и d = 1.

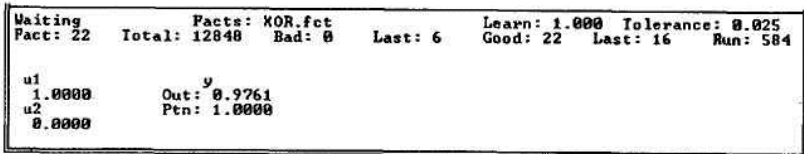


Рис. 5.77. Результат тестирования той же сети для $u_1 = 1$, $u_2 = 0$ и $d = 1$.

```
Waiting            Facts: XOR.fct                     Learn: 1.000  Tolerance: 0.025
Fact: 22    Total: 12848   Bad: 0    Last: 6     Good: 22    Last: 16      Run: 584

 u1               y
  1.0000       Out: 0.0247
 u2            Ptn: 0.0000
  1.0000
```

Рис. 5.78. Результат тестирования той же сети для $u_1 = 1$, $u_2 = 1$ и $d = 0$.

Значение абсолютной разности между эталоном и фактическим выходным значением на следующих друг за другом рисунках составляет 0,0252, 0,0246, 0,0239, 0,0247 соответственно, что отражает гистограмма на рис. 5.74.
Значения весов для сети, обученной с толерантностью 0,025, приведены на рис. 5.79.

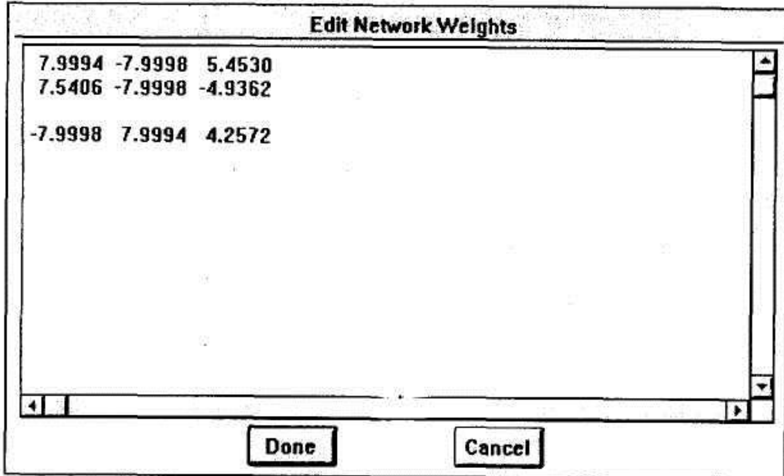


Рис. 5.79. Веса нейронной сети, обученной программой **BrainMaker** с толерантностью 0,025 (рис. 5.74).

Интересно сравнить его с рис. 5.73. Более поздние результаты оказываются гораздо ближе к оптимальным. Далее была предпринята попытка еще лучше обучить сеть с уменьшением толерантности до 0,02. К сожалению, эта попытка завершилась неудачей, поскольку способности сети к обучению оказались исчерпанными. Однако при фиксации толерантности на уровне 0,023 был достигнут конечный эффект, показанный на рис. 5.80.

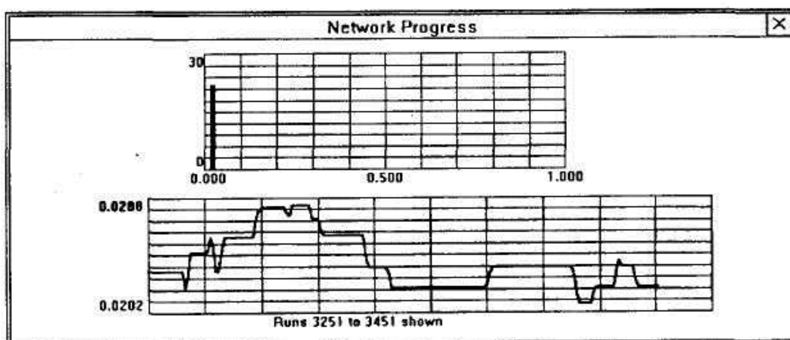


Рис. 5.80. Итоговая фаза последующего обучения программой **BrainMaker** сети с весами, показанными на рис. 5.79, при уровне толерантности 0,023.

Результаты тестирования сети, обученной подобным образом, представлены на рис. 5.81 – 5.84, а значения весов этой сети - на рис. 5.85.

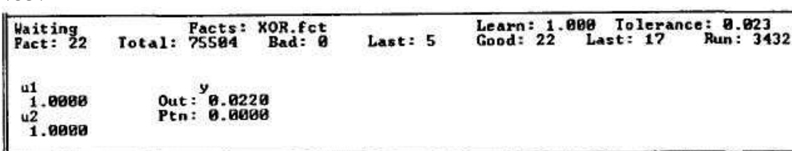


Рис. 5.81. Результат тестирования нейронной сети, обученной программой **BrainMaker** с толерантностью 0,023 (рис. 5.80), для $u_1 = 0$, $u_2 = 0$ и d = 0.

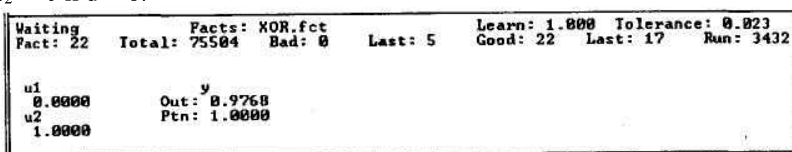


Рис. 5.62. Результат тестирования той же сети для $u_1 = 0$, $u_2 = 1$ и $d=1$.

```
Waiting            Facts: XOR.fct               Learn: 1.000  Tolerance: 0.023
Fact: 22    Total: 75504   Bad: 0     Last: 5   Good: 22   Last: 17    Run: 3432

u1                    y
 1.0000         Out: 0.9768
u2              Ptn: 1.0000
 0.0000
```

Рис. 5.83. Результат тестирования той же сети для $u_1 = 1$, $u_2 = 0$ и $d=1$.

```
Waiting            Facts: XOR.fct               Learn: 1.000  Tolerance: 0.023
Fact: 22    Total: 75504   Bad: 0     Last: 5   Good: 22   Last: 17    Run: 3432

u1                    y
 1.0000         Out: 0.0220
u2              Ptn: 0.0000
 1.0000
```

Рис. 5.84. Результат тестирования той же сети для $u_1 = 1$, $u_2 = 1$ и $d = 0$.

```
                 Edit Network Weights
 7.9994 -7.9998  3.7880
 7.9994 -7.9998 -4.1242

-7.9522  7.9994  3.8480

          Done            Cancel
```

Рис. 5.85. Веса нейронной сети, обученной программой **BrainMaker** с толерантностью 0,023 (рис. 5.80**).**

Еще лучший результат обучения, который иллюстрируется рис. 5.86, удалось получить при толерантности 0,022.

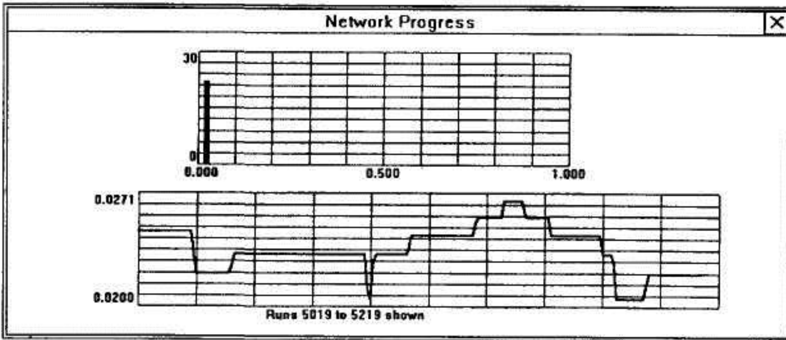


Рис. 5.86. Итоговая фаза последующего обучения программой **BrainMaker** сети с весами, показанными на рис. 5.85, при уровне толерантности 0,022.

Результаты тестирования сети, обученной подобным образом, представлены на рис. 5.87-5.90, а значения весов этой сети - на рис. 5.91.

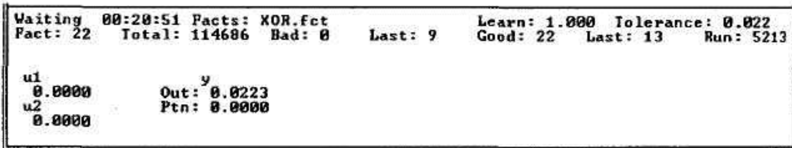


Рис. 5.87. Результат тестирования нейронной сети, обученной программой **BrainMaker** с толерантностью 0,022 (рис. 5.86), для $u_1 = 0$, $u_2 = 0$ и $d = 0$.

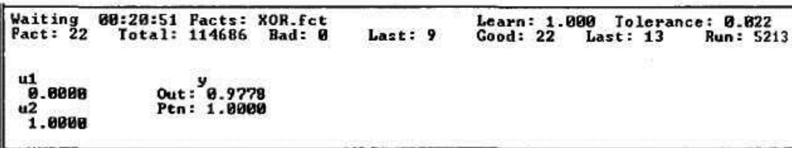


Рис. 5.88. Результат тестирования той же сети для $u_1 = 0$, $u_2 = 1$ и $d = 1$.

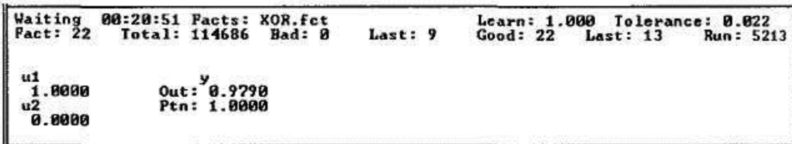


Рис. 5.89. Результат тестирования той же сети для $u_1 = 1$, $u_2 = 0$ и $d = 1$.

```
Waiting  00:20:51 Facts: XOR.fct              Learn: 1.000  Tolerance: 0.022
Fact: 22   Total: 114686  Bad: 0    Last: 9   Good: 22   Last: 13    Run: 5213

 u1             y
  1.0000      Out: 0.0223
 u2           Ptn: 0.0000
  1.0000
```

Рис. 5.90. Результат тестирования той же сети для $u_1 = 1$, $u_2 = 1$ и $d = 0$.

```
Edit Network Weights

 7.9994 -7.9998  4.6684
 7.9994 -7.9998 -4.4176

-7.9998  7.9994  4.0536
```

Рис. 5.91. Веса нейронной сети, обученной программой **BrainMaker** с толерантностью 0,022 (рис. 5.86).

Дальнейшее снижение уровня толерантности до 0,021, к сожалению, уже не ведет к большей обученности сети, даже если бы программа работала еще в течение многих часов. Таким образом, наилучшим решением считается сеть со значениями весов, показанными на рис. 5.91, которые следует сравнить с набором весов для примера 3.27, приведенными на рис. 5.37. Нетрудно заметить, что объединение генетического алгоритма программы **Evolver** с программой **BrainMaker** дало лучшие результаты, чем на рис. 5.37. Если бы вычисления сразу проводились с толерантностью 0,022, а в качестве начальных весов принимались значения с рис. 5.64, то результат был бы лучше, чем на рис. 5.37. Аналогичный результат можно было ожидать и в примере 5.1 при установлении уровня толерантности, равным 0,022.

Пример 5.3

Обучить с помощью программы **BrainMaker** нейронную сеть, реализующую логическую систему XOR (рис. 3.11) с начальными значениями весов, рассчитанными в примере 3.27 генетическим алгоритмом программы **Evolver** и представленными на рис. 5.35. Это еще один пример гибридного подхода к обучению весов нейронной сети, реализующей логическую систему XOR. Он очень похож на предыдущий пример, однако отличается набором начальных значений весов,

полученных за меньшее время выполнения генетического алгоритма и, следовательно, более далеких от оптимальных.

Толерантность погрешности принята равной 0,025, что означает 2,5 % допустимой погрешности, т.е. разницы между фактическим выходным значением и заданным значением - эталоном. Поэтому для эталона, равного 0, корректным будет признаваться выходное значение от 0 до 0,025, а для эталона, равного 0 - значение от 0,975 до 1. Начальный набор весов для программы **BrainMaker** представлен на рис. 5.92, а на рис. 5.93 – 5.96 показаны результаты тестирования сети.

```
Edit Network Weights

 5.9400 -6.4616 -6.8336
-7.5970  5.4952 -4.4936

 6.8216  7.4104 -0.6816
```

Рис. 5.92. Исходное множество весов для программы **BrainMaker,** полученное с помощью генетического алгоритма программы **Evolver.**

```
Waiting          Facts: XOR.fct              Learn: 1.000  Tolerance: 0.100
Fact: 22   Total: 22     Bad: 17   Last: 0   Good: 5    Last: 0      Run: 1

u1                 y
 0.0000       Out: 0.3558
u2            Ptn: 0.0000
 0.0000
```

Рис. 5.93. Результат тестирования нейронной сети с весами, показанными на рис. 5.92, для $u_1 = 0$, $u_2 = 0$ и $d = 0$.

```
Waiting          Facts: XOR.fct              Learn: 1.000  Tolerance: 0.100
Fact: 22   Total: 22     Bad: 17   Last: 0   Good: 5    Last: 0      Run: 1

u1                 y
 0.0000       Out: 0.9915
u2            Ptn: 1.0000
 1.0000
```

Рис. 5.94. Результат тестирования той же сети для $u_1 = 0$, $u_2 = 1$ и $d = 1$.

```
Waiting          Facts: XOR.fct              Learn: 1.000  Tolerance: 0.100
Fact: 22   Total: 22     Bad: 17   Last: 0   Good: 5    Last: 0      Run: 1

u1                 y
 1.0000       Out: 0.7859
u2            Ptn: 1.0000
 0.0000
```

Рис. 5.95. Результат тестирования той же сети для $u_1 = 1$, $u_2 = 0$ и $d = 1$.

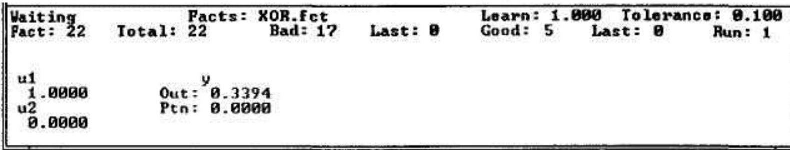


Рис. 5.96. Результат тестирования той же сети для $u_1 = 1$, $u_2 = 1$ и $d= 0$.

Видно, что выходные значения у для конкретных пар входов весьма близки к приведенным на рис. 5.35. Сеть не может считаться обученной. На 22 тестирующих выборках только 5 раз реакция на выходе была корректной, а в 17 случаях - ошибочной. Набор весов с рис. 5.92 был получен после всего лишь 96 «тактов» функционирования программы **Evolver** (что соответствует менее чем 2 итерациям классического генетического алгоритма). Результаты обучения нейронной сети с этими весами программой **BrainMaker** с толерантностью погрешности, равной 0,025, представлены на рис. 5.97.

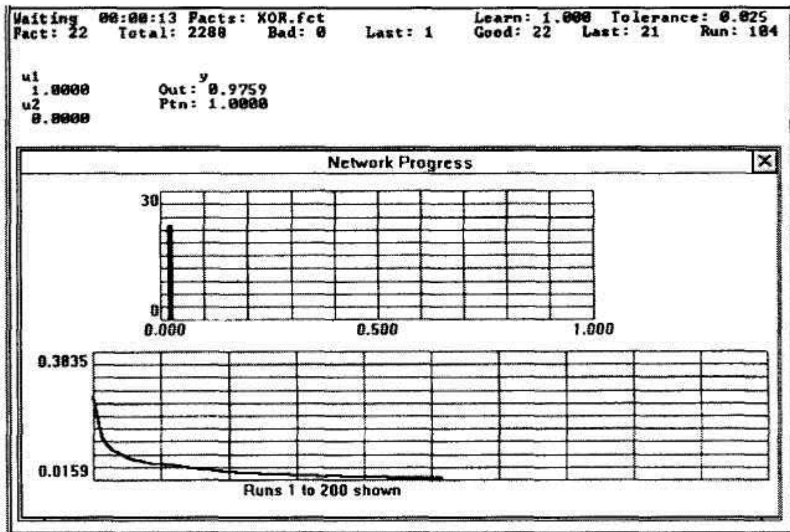


Рис. 5.97. Результат обучения програмой **BrainMaker** нейронной сети с весами, представленными на рис. 5.64, при уровне толерантности 0,025.

Значение погрешности RMS с графика этого рисунка легко сравнить со значением погрешности Q на рис. 5.35, 5.36 и 5.37. Заметно, что выходные значения y для каждой пары входов системы XOR укладываются в границы 2,5 % толератности. Сеть обучилась достаточно быстро - за 104 прогона *(runs).* На рис. 5.101 представлены веса обученной сети, а на рис. 5.97 – 5.100 - результаты ее тестирования.

```
Waiting            Facts: XOR.fct                 Learn: 1.000  Tolerance: 0.025
Fact: 22   Total: 2288    Bad: 0     Last: 1      Good: 22   Last: 21    Run: 104

 u1              y
  0.0000       Out: 0.9778
 u2            Ptn: 1.0000
  1.0000
```

Рис. 5.98. Результат тестирования нейронной сети, обученной программой **BrainMaker** с толерантностью 0,025 (рис. 5.97), для $u_1 = 0$, $u_2 = 1$ и $d = 1$.

```
Waiting            Facts: XOR.fct                 Learn: 1.000  Tolerance: 0.025
Fact: 22   Total: 2288    Bad: 0     Last: 1      Good: 22   Last: 21    Run: 104

 u1              y
  0.0000       Out: 0.0252
 u2            Ptn: 0.0000
  0.0000
```

Рис. 5.99. Результат тестирования той же сети для $u_1 = 0$, $u_2 = 0$ и $d = 0$.

```
Waiting            Facts: XOR.fct                 Learn: 1.000  Tolerance: 0.025
Fact: 22   Total: 2288    Bad: 0     Last: 1      Good: 22   Last: 21    Run: 104

 u1              y
  1.0000       Out: 0.0227
 u2            Ptn: 0.0000
  1.0000
```

Рис. 5.100. Результат тестирования той же сети для $u_1 = 1$, $u_2 = 1$ и $d = 0$.

Представляет интерес сопоставление рис. 5.101 и 5.79, поскольку они отражают обучение с одной и той же толерантностью, равной 0,025.

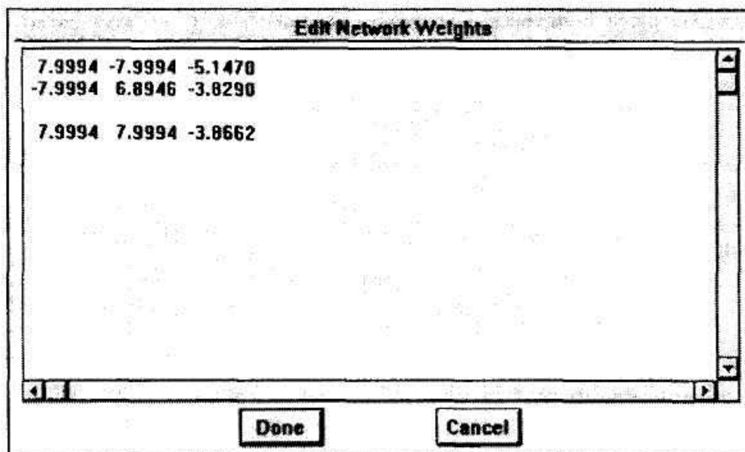


Рис. 5.101. Веса нейронной сети (при начальных значениях, показанных на рис. 5.92), обученной программой **BrainMaker** с толерантностью 0,025 .

Если продолжить обучение сети при меньшем значении толерантности, то (по аналогии с примером 3.31) можно найти значения весов, еще более близкие к оптимальным и практически совпадающие с показанными на рис. 5.91.

Рассмотрим теперь эффект обучения нейронной сети, реализующей логическую систему XOR, с помощью только программы **BrainMaker** без применения генетического алгоритма.

Пример 5.4

Обучить с помощью программы **BrainMaker** нейронную сеть, реализующую логическую систему XOR (рис. 3.2) с начальными значениями весов, представленными на рис. 5.32.

Показанные на рис. 5.32 значения весов сгенерированы случайным образом. На рис. 5.102 эти веса представлены в формате программы **BrainMaker.**

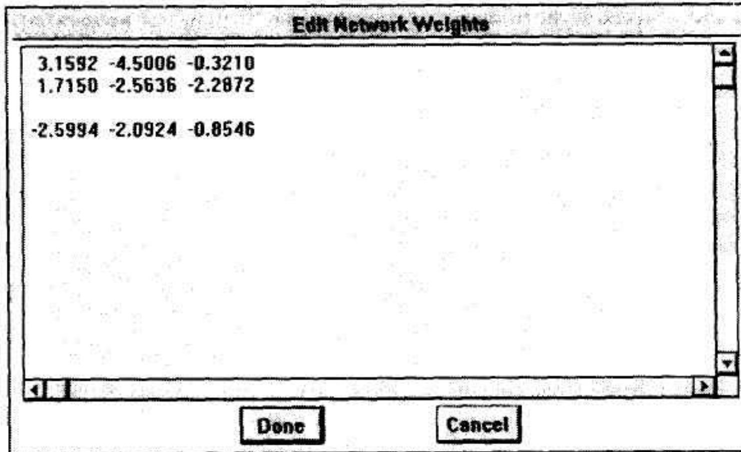


Рис. 5.102. Сгенерированный случайным образом исходный набор весов для нейронной сети, реализующей систему XOR.

Нейронная сеть с этими весами не может считаться обученной. Результаты ее тестирования приведены на рис. 5.103 – 5.06.

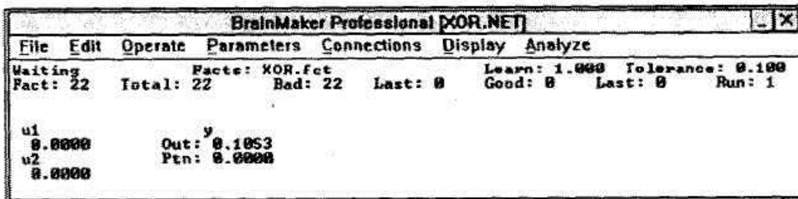


Рис. 5.103. Результат тестирования нейронной сети с весами, показанными на рис. 5.86, для $u_1 = 0$, $u_2 = 0$ и $d = 0$.

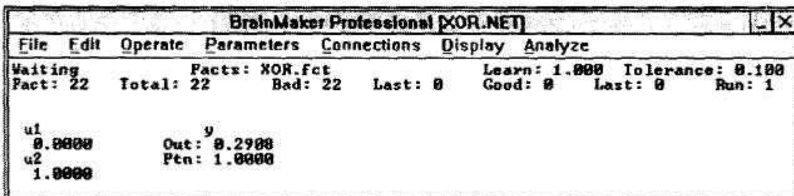


Рис. 5.104. Результат тестирования той же сети для $u_1 = 0$, $u_2 = 1$ и $d = 1$.

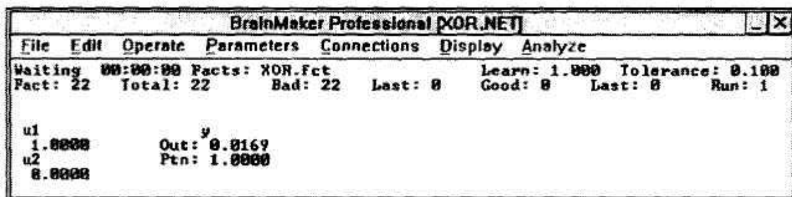


Рис. 5.105. Результат тестирования той же сети для $u_1 = 1, u_2 = 0$ и d=1.

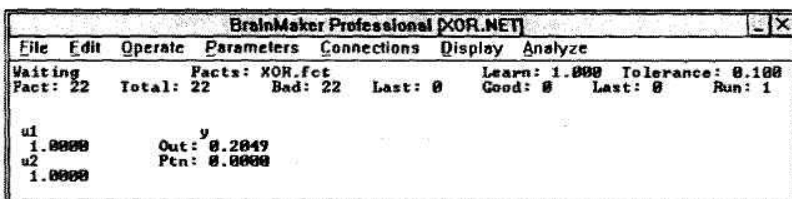


Рис. 5.106. Результат тестирования той же сети для $u_1 = 1$, $u_2 = 1$ и $d = 0$.

Заметно, что выходные значения *у* совершенно не соответствуют эталону 1, а для эталона 0 выходные значения также не попадают в границы 10% толерантности. Для всех 22 тестирующих выборок получены ошибочные выходные сигналы.

Вначале обучение сети проводилось с толерантностью погрешности, равной 0,1. Процесс обучения иллюстрируют графики на рис. 5.107.

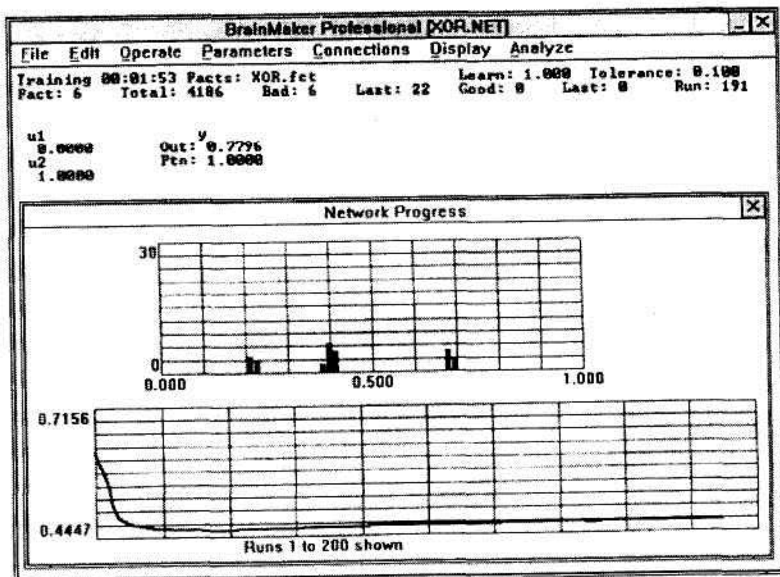


Рис. 5.107. Начальная фаза обучения программой **BrainMaker** сети с весами, показанными на рис. 5.102, при уровне толерантности 0,1.

На рис. 5.108 показаны значения весов, полученные после 191 прогона (runs) алгоритма.

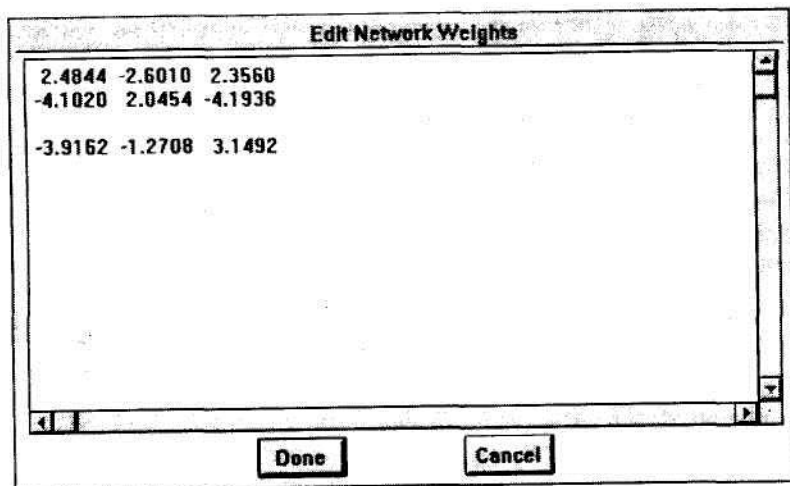


Рис. 5.108. Веса, полученные после 191 прогона алгоритма обучения программы **BrainMaker.**

Продолжение графика с рис. 5.107 демонстрируется на рис. 5.109, а его завершение - на рис. 5.110.

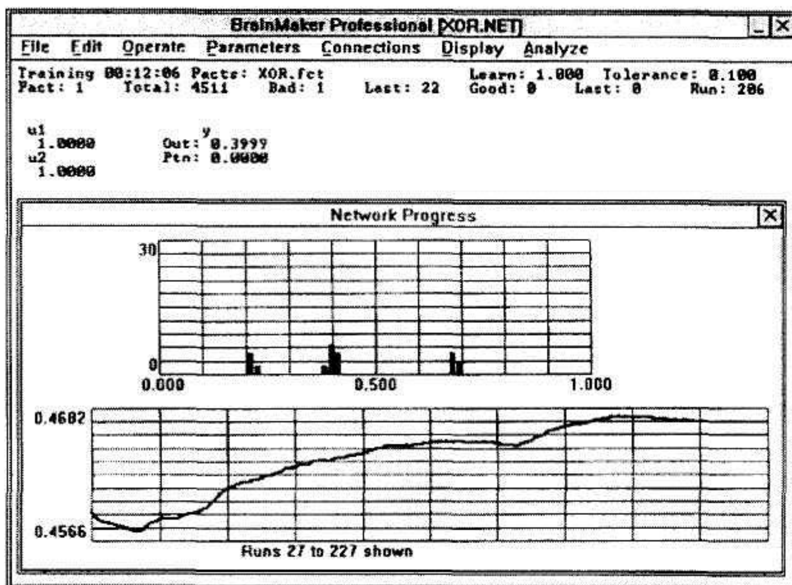


Рис. 5.109. Продолжение обучения, показанного на рис. 5.107.

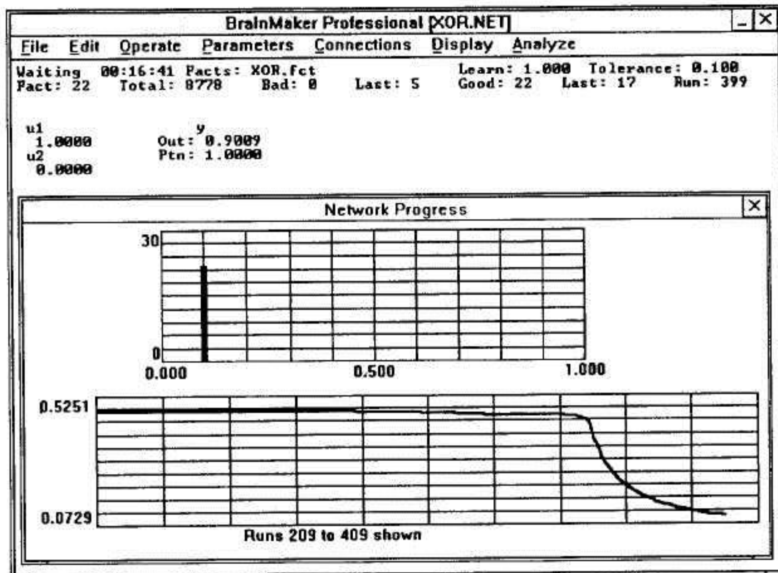


Рис. 5.110. Завершающая фаза обучения, показанного на рис. 3.187.

Полученные значения весов нейронной сети с толерантностью погрешности, равной 0,1, представлены на рис 5.111.

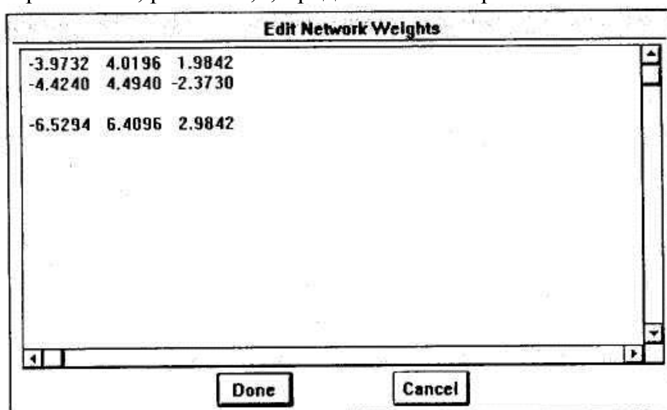


Рис.5.111. Веса нейронной сети, обученной программой **BrainMaker** с толерантностью 0,1**.**

Заметим, что значение погрешности RMS для этого случая (см. рис. 5.110) достаточно близко к значению погрешности Q, рассчитанному программой **Evolver** в примерах 3.23 - 3.27 для значений весов с рис. 5.111 (среднеквадратичная погрешность Q = 0,009887).
Нейронная сеть с весами, показанными на рис. 5.102, была обучена с толерантностью погрешности 0,1 за 399 прогонов алгоритма программы **BrainMaker.** Результаты тестирования сформированной сети представлены на рис. 5.112-5.114.

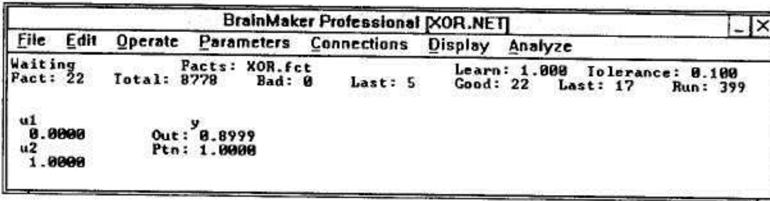


Рис. 5.112. Результат тестирования нейронной сети, обученной программой **BrainMaker** с толерантностью 0,025, для $u_1 = 0$, $u_2 = 1$ и $d = 1$.

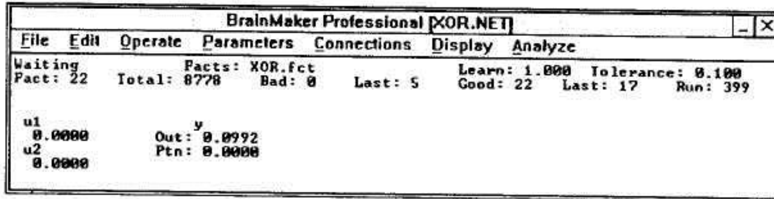


Рис. 5.113. Результат тестирования той же сети для $u_1 = 0$, $u_2 = 0$ и $d = 0$.

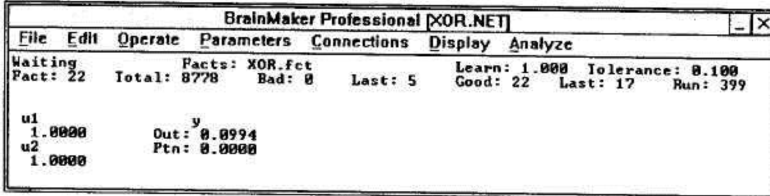


Рис. 5.114. Результат тестирования той же сети для $u_1 = 1$, $u_2 = 1$ и $d = 0$.

Для входов $u_1 = 1$ и $u_2 = 0$ решение приведено на рис. 5.110. Сеть может считаться обученной, поскольку в ходе тестирования не

зарегистрированы ошибочные отклики. Далее обучение сети продолжилось с толерантностью погрешности, равной 0,025. После 510 прогонов получены графики, изображенные на рис. 5.115, и значения весов, приведенные на рис. 5.116.

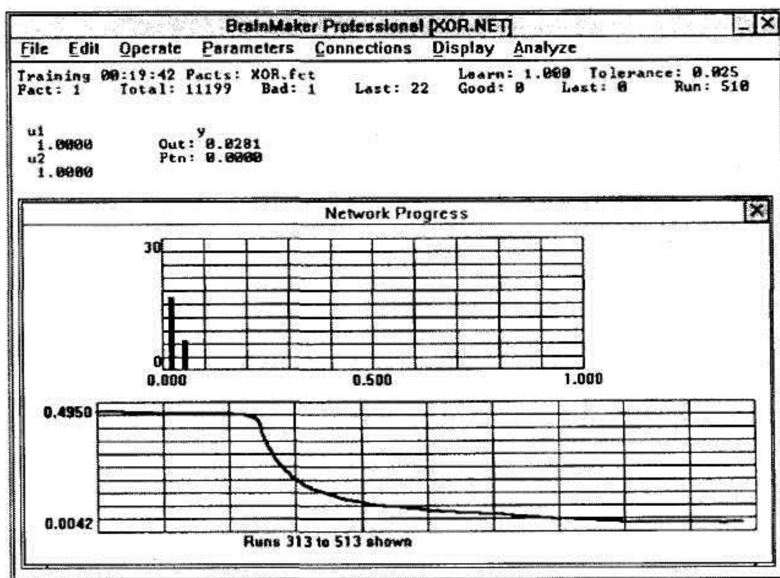


Рис. 5.115. Графики, полученные при последующем обучении программой **BrainMaker** сети с весами, показанными на рис. 5.111, при уровне толерантности 0,025.

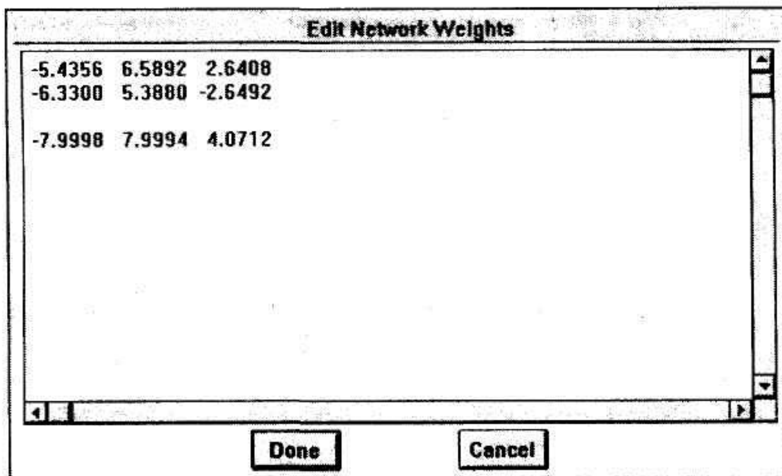


Рис. 5.116. Веса, полученные в результате последующего обучения сети при уровне толерантности 0,025 (после 510 прогонов - рис. 5.115).

Значение погрешности RMS для этой комбинации весов можно получить с графика на рис. 5.115. При толерантности погрешности 0,025 сеть плохо поддавалась обучению, поэтому после 4000 прогонов уровень толерантности был увеличен до 0,03. Графики и значения весов на момент изменения уровня показаны на рис.5.117 и 5.118 соответственно.

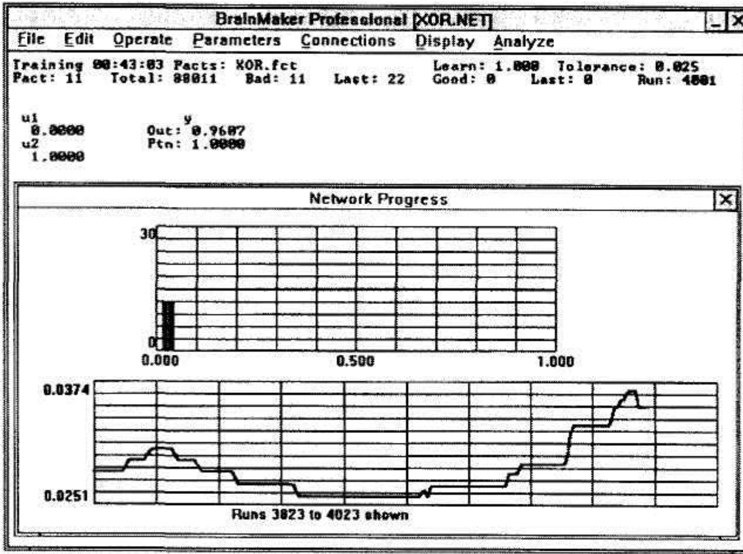


Рис. 5.117. Процесс обучения сети с толерантностью 0,025 после 4000 прогонов.

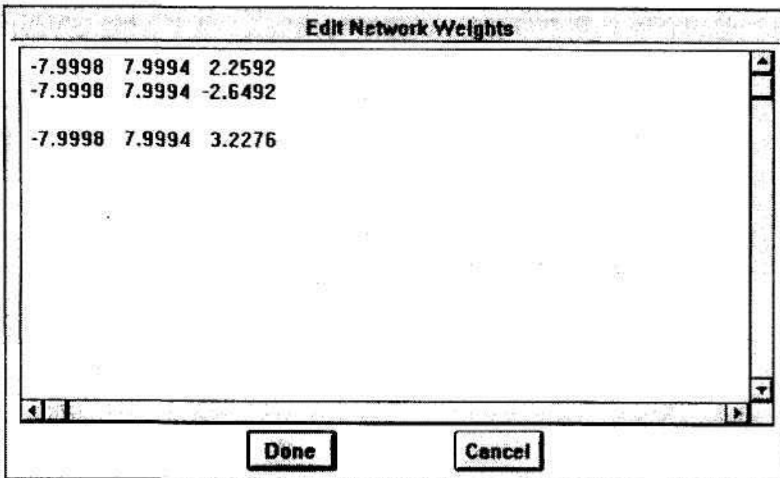


Рис. 5.118. Веса, полученные после 4000 прогонов обучения сети с толерантностью 0,025.

После увеличения значения толерантности выполнение программы очень быстро - после 402 прогонов - завершилось. Достигнутый эффект можно наблюдать на рис. 5.119, а полученные значения весов - на рис. 5.120.

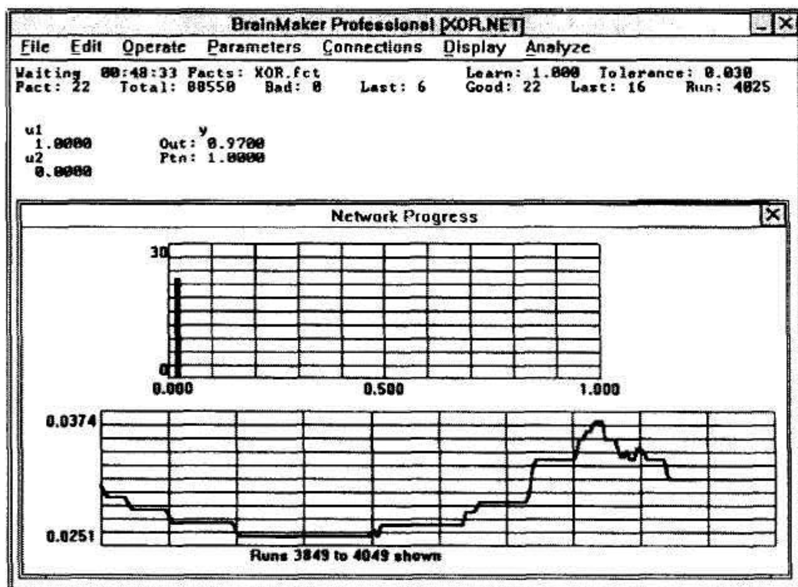


Рис. 5.119. Процесс обучения сети на рис. 5.117 при изменении толерантности до 0,03.

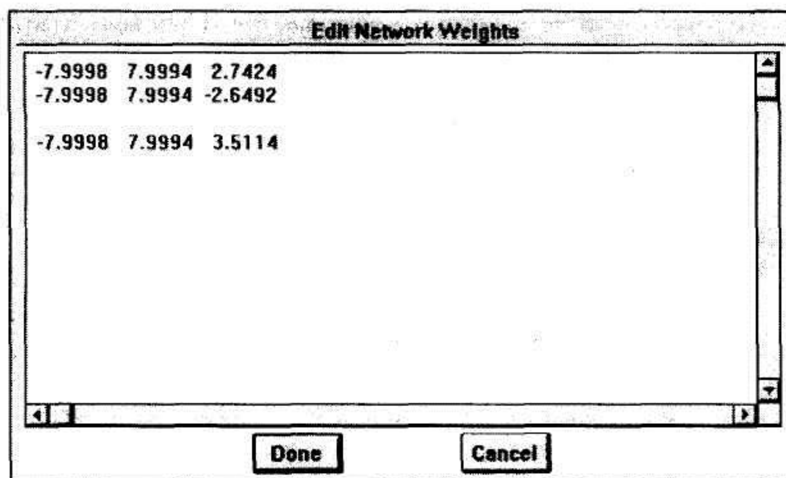


Рис. 5.120. Веса, полученные по завершении обучения сети с толерантностью 0,03 (рис. 5.119).

В результате обучения нейронной сети, реализующей логическую систему XOR с начальными значениями весов, представленными на рис. 5.102 и 5.32, была найдена еще одна комбинация весов, подобная полученным при минимизации погрешности Q с помощью программы **Evolver.** В принципе, сформированный в текущем примере и представленный на рис. 3.200 набор весов сети, обученной с помощью программы **BrainMaker,** отличается от двух аналогичных комбинаций «наилучших» весов из примера 3.27 только знаками. Например, если сравнить веса с рис. 3.200 с весами, показанными на рис. 5.37, то можно зафиксировать, что веса w_{11}, w_{12}, w_{21}, w_{22} изменили знаки на противоположные, а веса w_{31}, w_{32}, w_{10}, w_{20}, w_{30} имеют в обоих случаях одни и те же знаки, т.е. w_{31} и w_{20} остались отрицательными, а w_{32}, w_{10}, w_{30} - положительными. Результаты тестирования сформированной сети (с весами на рис. 3.200) для входов $u_1 = 1$ и $u_2 = 0$ приведены на рис. 5.119. Для пары входов $u_1 = 0$ и $u_2 = 1$ значение y осталось неизменным, т.е. равным 0,9700.

Для $u_1 = 0$ и $u_2 = 0$, а также для $u_1 = 1$ и $u_2 = 1$ выходное значение y равно 0,0301.

Рассмотренные примеры иллюстрируют гибридный подход, основанный на объединении двух различных методов - генетического

алгоритма и градиентного метода обучения весов нейронной сети, известного под названием обратного распространения ошибки. В примере 3.27 для обучения нейронной сети применялся генетический алгоритм программы **Evolver.** После 40 000 «тактов» функционирования этого алгоритма (что соответствует 800 поколениям классического генетического алгоритма) получено близкое к оптимальному решение. Однако, как показывают примеры 5.1 и 5.2, этот результат можно еще улучшить за счет «дообучения» с помощью градиентного алгоритма.

Примеры 5.2 и 5.3 представляют типичный способ такого гибридного подхода, когда генетический алгоритм применяется только для выбора начальной точки (в данном случае - исходного множества весов) для градиентного метода. В примере 5.2 генетический алгоритм функционировал дольше, чем в примере 5.3, поэтому начальная точка для метода обратного распространения ошибки в примере 5.3 находится на большем расстоянии от точки оптимального решения, чем в примере 5.2. Следует обратить внимание на факт, что длительность «дообучения» не имеет большого значения. Градиентный алгоритм выполняется быстрее генетического, так как в последнем предполагается просмотр всей популяции возможных решений.

В примере 5.4 продемонстрирован способ обучения нейронной сети методом обратного распространения ошибки (программа **BrainMaker).** Полученный результат оказался еще более близким к оптимальному, чем при использовании только генетического алгоритма (пример 3.30). Однако необходимо подчеркнуть, что градиентный метод не всегда приводит к достижению ожидаемого результата, который зависит от начальной точки. При другом, сгенерированном случайным образом исходном множестве весов сеть может оказаться «необучаемой», что встречается довольно часто. Кроме того, принципиальным недостатком градиентных методов оказывается их «застревание» в локальных оптимумах. Это можно предотвратить применением генетического алгоритма - только для того, чтобы найти какую-либо точку, настолько близкую к глобальному оптимуму, что градиентный алгоритм при старте из этой точки не «застрянет» ни в каком локальном оптимуме и быстро найдет глобальное решение.

В рассмотренных примерах использовался генетический (эволюционный) алгоритм программы **Evolver**. Конечно, для достижения тех же целей можно было применять и программу **FlexTool**, которая допускает такой же интервал изменения значений весов от -7 до 8, как и программа **BrainMaker**. Для выбора исходного

множества весов нейронной сети также пригодны и другие реализации генетического (эволюционного) алгоритма.

5.4.2. Программа GTO

Программа **GTO** (Genetic Training Option - Режим Генетического Обучения) взаимодействует с программой **BrainMaker.** В ней реализован эволюционный алгоритм с соответственно определенными операторами мутации и скрещивания. Мутация изменяет значения весов избранного нейрона, а скрещивание заключается в обмене весов избранных нейронов у пары родителей. Например, для нейронной сети, изображенной на рис. 3.11, можно получить потомка, имеющего веса первого нейрона, унаследованные от первого родителя, и веса второго и третьего нейронов - от второго родителя. Выраженные в процентах показатели мутации (mutation rate) и скрещивания (crossover rate) определяют количество нейронов, подвергающихся мутации или скрещиванию. Пользователь также может ввести значения дополнительных параметров скрещивания и мутации. Они относятся к так называемому полному скрещиванию либо к определенным функциям распределения показателей скрещивания и/или мутации.

Пример 5.5

Применить программу **GTO** для нейронной сети, реализующей логическую систему XOR (рис. 3.11) с начальными значениями весов, представленными на рис. 5.102, которая после обучения программой **BrainMaker** с толерантностью погрешности, равной 0,1 (пример 5.4) имеет веса, представленные на рис. 5.111.

Все параметры программы **GTO** имели значения, установленные по умолчанию. В частности, предусматривалось обучение каждой сети программой **BrainMaker** в течение 100 прогонов (runs) и смена 30 поколений. Для оценки приспособленности сети использовались результаты как обучения, и тестирования уже обученной сети. Показатели мутации и скрещивания были равны соответственно 10 и 50. Это означает, что 50% всех нейронов сети подвергалось скрещиванию, а 10% - мутации. Также принималось, что все скрещиваемые нейроны подвергаются полному скрещиванию, т.е. потомок наследует все веса нейрона от второго родителя. Помимо того, все подлежащие мутации нейроны мутировали в соответствии с распределением Гаусса с показателем 0,25. Это означает, что к весу добавлялась некоторая величина, выбиравшаяся по распределению Гаусса. В качестве

критерия оценивания потомков применялся тах{1 - RMS}, где RMS (также, как и в примере 3.31) означал погрешность, рассчитанную как квадратный корень из суммы квадратов разностей между заданным (эталонным) и выходным значением, деленный на количество эталонов. В результате выполнения программы **GTO** при ее взаимодействии с программой **BrainMaker** рассчитаны значения указанной выше погрешности для конкретных сетей, упорядоченные в порядке их уменьшения, т.е. от «наилучшей» сети к «наихудшей» (рис. 5.121).

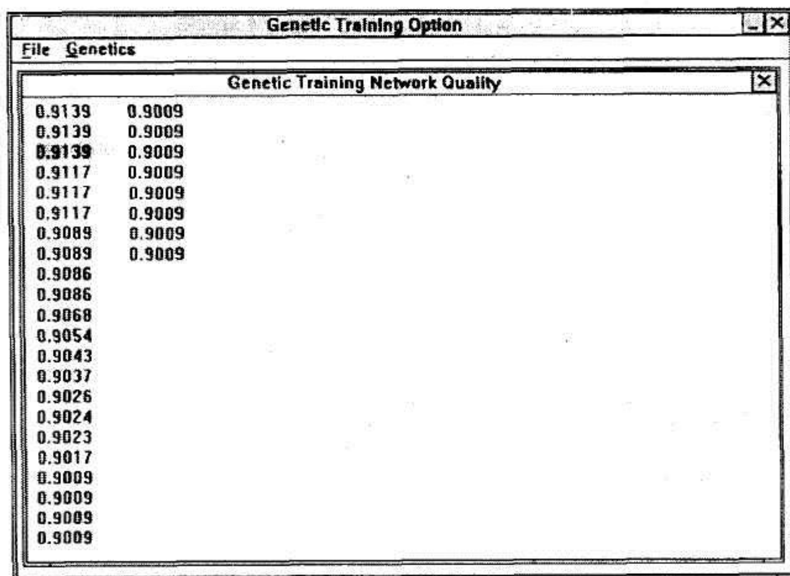


Рис. 5.121. Список нейронных сетей, сформированных программой **GTO** для примера 5.5 и упорядоченных в последовательности от «наилучшей» до «наихудшей».

Наилучшей из 30 сформированных программой **GTO** нейронных сетей со структурой, показанной на рис. 3.11, оказалась сеть, для которой значение max{1 - RMS} было равно 0,9139. Значения весов этой сети представлены на рис. 5.122, а результаты ее тестирования програм мой **BrainMaker** - на рис. 5.123 – 5.126.

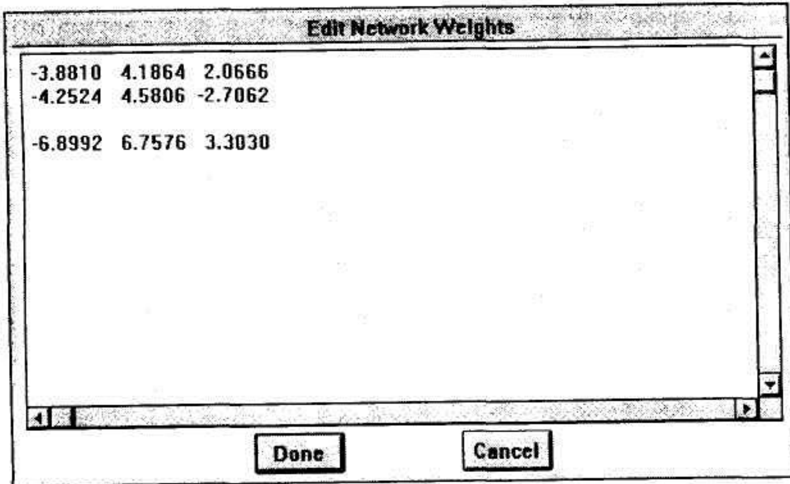


Рис. 5.122. Веса нейронной сети, реализующей систему XOR, полученные программой **GTO** (пример 5.5).

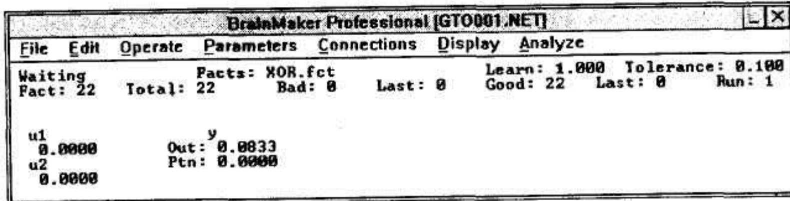


Рис. 5.123. Результат тестирования сформированной программой **GTO** нейронной сети с весами, показанными на рис. 5.122, для $u_1 = 0$, $u_2 = 0$ и $d = 0$.

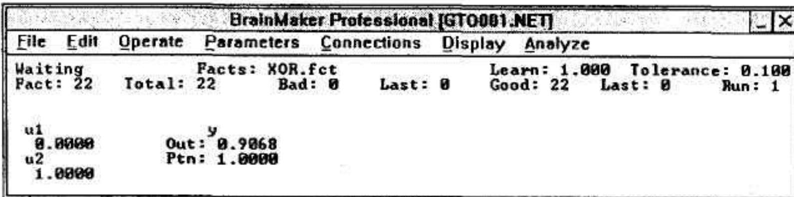


Рис. 5.124. Результат тестирования той же сети для $u_1 = 0$, $u_2 = 1$ и $d = 1$.

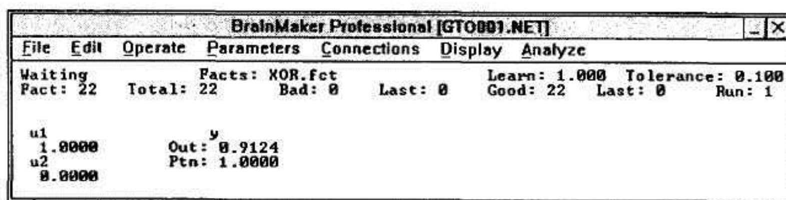


Рис. 5.125. Результат тестирования той же сети для $u_1 = 1$, $u_2 = 0$ и $d = 1$.

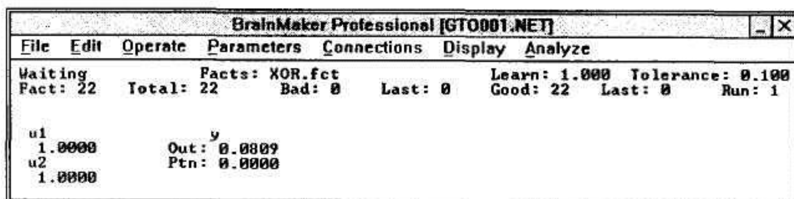


Рис. 5.126. Результат тестирования той же сети для $u_1 = 1$, $u_2 = 1$ и $d = 0$.

Заметим, что значение погрешности Q, рассчитываемой программой **Evolver**, для сети с весами, показанными на рис. 5.122, равно 0,007451. Эта погрешность меньше значения Q, рассчитанного для сети из примера 5.4 (веса см. на рис. 5.111). Конечно, новая сеть ненамного лучше сети из примера 5.4. Теперь следует вновь применить программу **BrainMaker** для проверки - можно ли улучшить характеристики сети, полученной в результате выполнения программы **GTO.** Очевидно, что попытка дообучения этой сети с толерантностью погрешности 0,1 ничего не изменит. Однако при задании уровня толерантности 0,025 мы получаем процесс, показанный на рис. 5.127 и значения весов, представленные на рис. 5.128.

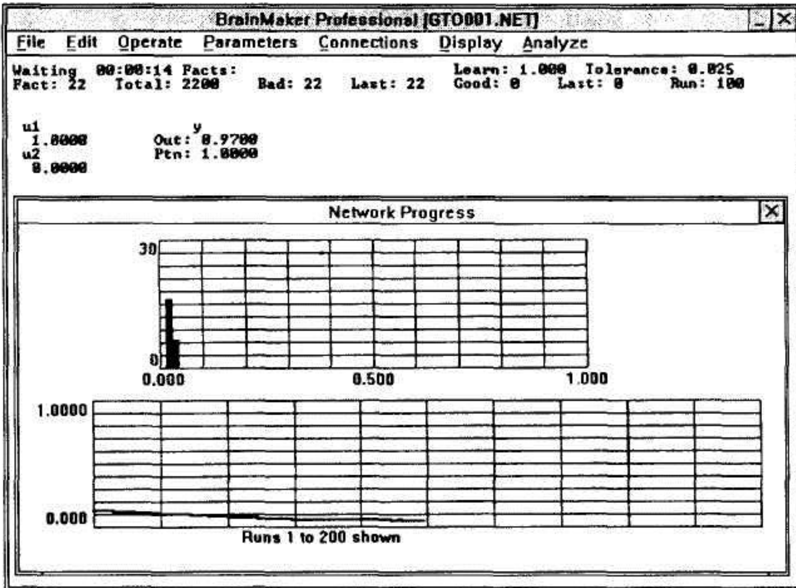


Рис. 5.127. Процесс обучения нейронной сети, сформированной программой **GTO**, осуществляемый программой **BrainMaker** с толерантностью 0,025.

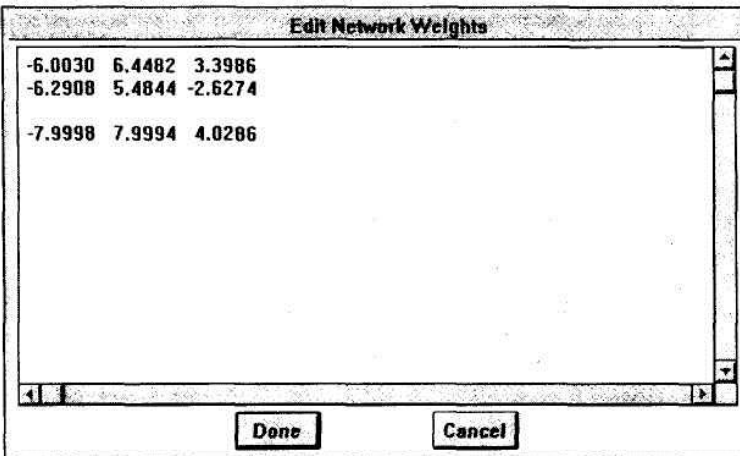


Рис. 5.128. Веса, полученные в результате гибридного обучения сети программами **BrainMaker, GTO** и повторно **BrainMaker**.

Выходное значение у для $u_1 = 1$ и $u_2 = 0$ также приведено на рис. 5.127. Пример 5.5 иллюстрирует следующее объединение традиционного (градиентного) алгоритма обучения, реализованного в программе **BrainMaker,** с генетическим алгоритмом программы **GTO:**

1) обучение нейронной сети программой **BrainMaker;**

2) применение программы **GTO** к сети, обученной в п.1;

3) продолжение обучения наилучшей сети, полученной в п.2, программой **BrainMaker;**

Следует отметить, что п.2 предполагает использование как генетического алгоритма программы **GTO,** так и соответствующей процедуры программы **BrainMaker,** вызываемой программой **GTO.**

Можно предложить и другой подход:

1) использование программы **BrainMaker** только для формирования сети со значениями весов, выбираемыми случайным образом;

2) применение программы **GTO** для поиска наилучших весов;

3) применение программы **BrainMaker** для завершения обучения.

Второй подход основан на обработке программой **GTO** сети со случайными значениями весов. При отключении опции обучения каждой вновь формируемой сети программой **BrainMaker** программа **GTO** вызывает ее только для тестирования сети. В последующем наилучшая из созданных таким образом сетей «дообучается» программой **BrainMaker** (п. 3). Такая методика подобна применявшейся в примерах 5.2 и 5.3 - вначале выполняется генетический, а затем - традиционный алгоритм обучения. В программе **GTO** дополнительно возможно обучение каждой вновь сформированной сети (потомка) за определенное количество прогонов с последующим ее тестированием. Это пример подхода, состоящего в «дообучении» нейронных сетей традиционным (в частности, градиентным) методом каждый раз перед оцениванием приспособленности. Программа **GTO** также предоставляет возможность выбрать один из нескольких критериев оценивания нейронных сетей (функцию их приспособленности). Решение о применении одного из двух рассмотренных гибридных подходов к совместному применению программ **GTO** и **BrainMaker** зависит от поставленной задачи и, как правило, принимается методом проб и ошибок. Практически невозможно априорно оценить - какой подход окажется лучшим для конкретной задачи.

Применительно к нейронной сети, предназначенной для реализации логической системы XOR, наилучшие результаты можно было бы ожидать от применения второго подхода. Однако с учетом ограни-

ченного объема настоящей работы мы не будем рассматривать дополнительные примеры, демонстрирующие различные возможности его реализации с использованием программы **GTO**.
Наиболее важные замечания относительно гибридного подхода, состоящего в объединении генетического алгоритма с градиентным методом обучения нейронных сетей (программа **BrainMaker**) представлены в п. 3.21.1. Программа **GTO** представляет собой пример равноправного объединения обоих методов, при котором в соответствии с типовым циклом эволюции приспособленность особей популяции рассчитывается генетическим алгоритмом по результатам обучения нейронных сетей. Существование такой программы, как **GTO**, подтверждает практическое применение гибридного подхода, объединяющего достоинства двух оптимизационных методов: генетического алгоритма, который легко находит точку, близкую к оптимальному решению, и градиентного алгоритма, который стартует из найденной точки и быстро приводит к настоящему оптимуму.

6. Применение генетических алгоритмов

6.1. Применение ГА для автоматической генерации тестов

При разработке и сопровождении программного обеспечения, значительная часть усилий тратится на поиск и устранение ошибок. Самым распространённым методом поиска ошибок является тестирование, то есть процесс выполнения программ с целью обнаружения ошибок. Здесь слово «программа» понимается в широком смысле, как любая запись алгоритма. В частности, программами являются отдельные процедуры, функции, классы и т.д. Процесс тестирования включает выполнение некоторого набора тестов и анализ полученных результатов. Тест - это последовательность обращений к тестируемой программе. Результатом выполнения теста является решение (вердикт) о том, отработала ли программа корректно или некорректно. Основной характеристикой тестового набора, определяющей качество тестирования, является класс возможных ошибок в программе, которые данный тестовый набор способен

обнаружить. Для количественной оценки качества тестирования используются различные метрики тестового покрытия. Для качественного тестирования необходимо построить полный тестовый набор, т. е. набор, удовлетворяющий некоторому критерию полноты. Зачастую критерий полноты для тестового набора определяют через пороговое значение метрики тестового покрытия. Построение полного тестового набора для больших систем вручную может быть крайне трудоёмкой задачей. Автоматизация этого процесса позволяет существенно снизить затраты на тестирование. Существуют различные подходы к решению задачи автоматической генерации тестов. Один из них основан на применении генетических алгоритмов. Этот подход во многих случаях даёт хорошие результаты. К сожалению, его эффективность существенно зависит от используемого критерия полноты. Цель данного роздела - проанализировать некоторые широко распространённые критерии полноты тестового набора на их применимость при использовании генетических алгоритмов для генерации тестов.

Основные понятия

Генетические алгоритмы

Как мы знаем, генетические алгоритмы - это метод решения задач оптимизации. В методе используются идеи, почерпнутые из эволюционной биологии: наследование признаков, мутация, естественный отбор и кроссовер. Определяется множество кандидатов, среди которых ищется решение задачи. Кандидаты представляются в виде списков, деревьев или иных структур данных. Общая схема работы генетического алгоритма подробно была описана в начальных разделах этого издания.

Генетические алгоритмы позволяют решать задачи, для которых не применимы традиционные методы оптимизации. Одной из областей применения генетических алгоритмов является автоматическая генерация тестов для программного обеспечения.

Критерии полноты тестового покрытия

Для тестирования программного обеспечения требуется создать репрезентативный набор тестов, то есть набор, охватывающий все возможные сценарии работы системы. Для оценки репрезентативности тестовых наборов используются различные критерии полноты тестового покрытия.

Пусть *P* - множество программных систем, *T* - множество тестов, а Σ - множество тестовых наборов, то есть множество всех конечных подмножеств множества *T*. Тогда задача генерации тестов может быть сформулирована следующим образом: для заданной тестируемой системы $S \in P$ построить тестовый набор $\sigma \in \Sigma$, удовлетворяющий заданному критерию полноты тестового покрытия $F : P \times \Sigma \rightarrow \{\bullet\ , \bot\}$, то есть такой набор σ, для которого $F(S, \sigma) = \bullet$.

Многие критерии полноты тестового покрытия, имеющие практическое применение, строятся по следующей схеме: для тестируемой системы *S* критерий *F* определяет множество элементов тестового покрытия Q_S^F. Элементом тестового покрытия можно считать некоторый класс событий, которые могут произойти в ходе работы тестируемой программной системы. По появлению в процессе исполнения программы элементов тестового покрытия и различных их комбинаций можно судить о полноте или качестве проверки, которую выполняет данный тестовый набор. Например, элементами тестового покрытия могут быть исполняемые строки исходного кода (соответствующие событиям их исполнения); рёбра графа потока управления; пути в графе потока управления; логические выражения, встречающиеся в исходном коде и т.п. Кроме того, критерий *F* определяет логическую функцию $f : Q_S^F \times T \rightarrow \{\bullet\ , \bot\}$, которая принимает значение $f(q, t) = \bullet$, если элемент тестового покрытия *q* покрывается тестом *t*. Тестовый набор σ для системы *S* удовлетворяет критерию полноты тестового покрытия *F*, если каждый элемент тестового покрытия из множества Q_S^F покрывается хотя бы одним тестом из тестового набора σ. Иными словами:

$$F(S,\sigma) \equiv \forall q \in Q_S^F : \exists t \in \sigma : f(q,t) = \bullet \quad (*)$$

Приведём несколько примеров часто упоминаемых критериев полноты тестового покрытия:

- каждый оператор в исходном коде выполняется хотя бы один раз;
- каждая ветвь графа потока управления выполняется хотя бы один раз;
- каждый путь графа потока управления исполнение выполняется хотя бы один раз;
- каждое логическое выражение хотя бы один раз вычисляется со значением «истина» и хотя бы один раз - со значением «ложь»;
- тестовый набор убивает всех мутантов из заданного набора.

Заметим, что все критерии, приведённые в качестве примеров, соответствуют ранее изложенной схеме.

Метрики тестового покрытия

Со многими критериями полноты тестового покрытия можно связать соответствующую метрику тестового покрытия. Метрика тестового покрытия - это функция вида $M : P \times \Sigma \to R$. Значение этой функции $M(S,\sigma)$ имеет смысл числовой оценки того, насколько хорошо тестовый набор σ покрывает тестируемую систему *S*. Сам критерий при этом можно записать в виде $M(S,\sigma) \geq \alpha_S$, где α_S - это минимальное пороговое значение метрики *M* для тестируемой системы *S*.

В частности, для критерия полноты тестового покрытия *F*, представимого в виде (*), можно ввести следующую метрику:

$$M^F(S,\sigma) = |\{q \in Q_S^F : \exists t \in \sigma : f(q,t) = \bullet \}| \quad (**)$$

Сам критерий при этом примет вид:

$$|\{q \in Q_S^F : \exists t \in \sigma : f(q,t) = \bullet\}| \geq |Q_S^F|$$

В некоторых случаях, когда не удаётся построить тестовый набор, удовлетворяющий такому критерию полноты тестового покрытия, можно использовать ослабленный критерий:

$$|\{q \in Q_S^F : \exists t \in \sigma : f(q,t) = \bullet\}| \geq \lambda |Q_S^F| \quad (***)$$

Параметр $\lambda \in (0,1]$ указывает, какая доля элементов тестового покрытия должна быть покрыта тестовым набором. Приведём несколько примеров часто упоминаемых метрик тестового покрытия:

- количество покрытых (выполненных хотя бы один раз) операторов в исходном коде;
- количество покрытых ветвей графа потока управления;
- количество покрытых путей графа потока управления;
- количество распознанных мутантов (версий тестируемой системы с искусственно привнесёнными ошибками).

Все эти метрики могут быть представлены в виде (**). Подробное описание этих и других, используемых на практике, метрик полноты тестового покрытия можно найти литератере

Генетический алгоритм генерации тестов

Рассмотрим следующую задачу генерации тестов:

Задача 1. Для заданной тестовой системы *S* построить тестовый набор $\sigma \in \Sigma$, удовлетворяющий критерию (***).

Для построения генетического алгоритма решения этой задачи необходимо определить:

- множество кандидатов;
- структуру представления кандидатов;
- оценочную функцию;
- оператор кроссовера;
- оператор мутации;
- условие останова.

Простейший алгоритм

Рассмотрим простейший генетический алгоритм для решения задачи 1. В качестве множества кандидатов возьмём множество Σ; в качестве оценочной функции возьмём метрику тестового покрытия $M^F(S,\sigma)$ для заданной тестируемой системы *S*. Условием останова будет наличие в текущем поколении решения σ, удовлетворяющего критерию (***). Структуру представления кандидатов, а также операторы кроссовера и мутации мы пока уточнять не будем. Заметим, что такой алгоритм допускает ситуацию, в которой критерий (***) не выполняется ни для одного тестового набора из текущего поколения, но, тем не менее, выполняется для некоторого объединения тестовых наборов из текущего и предшествующих поколений. Иными словами, все тесты, необходимые для построения решения, уже найдены, но само решение ещё не построено. В этой ситуации алгоритм не способен эффективно построить искомое решение, целенаправленно объединив подходящие тесты из разных тестовых наборов. Причина проблемы в том, что при построении алгоритма не использовалась имеющаяся информация о структуре критерия (***). Заметим также, что каждое последующее поколение тестов формируется путём применения операторов кроссовера и мутации к тестам из предыдущего поколения. Если в предыдущем поколении не было ни одного теста, покрывающего некоторый элемент тестового покрытия *q*, то в последующем поколении такой тест может появиться только как результат кроссовера или мутации тестов, не покрывающих *q*. Как бы мы не определяли операторы кроссовера и мутации, нет никаких оснований полагать, что получить таким способом тест, покрывающий *q*, проще, чем при полностью случайной генерации.

Из этих замечаний следует, что эффективность данного генетического алгоритма, вообще говоря, не выше, чем у полностью случайного алгоритма генерации тестов.

Целенаправленный поиск

Учитывая структуру критерия (***), из задачи 1 можно выделить следующую подзадачу:

Задача 2. Для заданной тестовой системы *S* и заданного элемента тестового покрытия *q*, построить тест $t \in T$, удовлетворяющий условию $f(q,t) = \bullet$.

Для решения исходной задачи 1, достаточно решить задачу 2 для $n \geq \lambda |Q_S^F|$ попарно различных элементов тестового покрытия $q_1, q_2, \dots, q_n \in Q_S^F$, то есть построить тесты $t_1, t_2, \dots, t_n$ такие, что

$$\begin{cases} f(q_1, t_1) = \bullet \ , \\ f(q_2, t_2) = \bullet \ , \\ \qquad \dots \\ f(q_n, t_n) = \bullet \ . \end{cases}$$

Решением задачи 1 будет множество $\{t_1, t_2, \dots, t_n\}$.

Рассмотрим генетический алгоритм решения задачи 2. В качестве множества кандидатов возьмём множество тестов *T*. Условие останова: в текущей популяции присутствует тест *q* такой, что $f(q,t) = \bullet$. Оценочная функция $m_q : T \to R$ каждому тесту *t* ставит в соответствие числовую меру $m_q(t)$ того, насколько тест *t* близок к тому, чтобы покрыть элемент тестового покрытия *q*. При этом

оценочная функция f_q достигает своего максимального значения на тех и только на тех тестах, которые удовлетворяют условию $f(q,t) = \bullet$. Иными словами:

$$f(q,t) = \bullet \Leftrightarrow m_q(t) = \max_{t \in T} m_q(t) \qquad (****)$$

В частности, в качестве оценочной функции можно использовать следующую функцию, удовлетворяющую условию (****):

$$m_q(t) = \begin{cases} 0, \text{при } f(q,t) = \perp, \\ 1, \text{при } f(q,t) = \bullet \end{cases}.$$

В такой оценочной функции считается, что все тесты, не покрывающие элемент тестового покрытия q, одинаково далеки от того, чтобы покрыть элемент q. При использовании этой оценочной функции эффективность генетического алгоритма будет не выше, чем при случайном поиске. Примеры более эффективных оценочных функций для некоторых метрик полноты тестового покрытия можно найти в литературе.

Оценочные функции

В этом разделе подробно рассматриваются три известных критерия полноты тестового покрытия, и для каждого из них предлагается оценочная функция.

Покрытие операторов исходного кода

Тестовый набор удовлетворяет критерию покрытия операторов исходного кода, если при выполнении этого тестового набора каждый оператор исходного текста программы выполняется хотя бы один раз. Элементами тестового покрытия в данном случае являются операторы исходного текста. Для заданного оператора q значение оценочной

функции $m_q(t)$ тем больше, чем ближе тест *t* к тесту, покрывающему оператор *q*. Для построения оценочной функции рассмотрим граф потока управления тестируемой системы *S*. Вершинами графа являются операторы исходного кода, то есть множество Q_S^F. В графе существует ребро, идущее из вершины q_1 в вершину q_2 тогда и только тогда, когда оператор q_2 может быть выполнен непосредственно после оператора q_1. Пусть $\Pi(q,t)$ - это множество всех элементов q' из Q_S^F, для которых выполняются следующие условия:

- существует путь в графе потока управления ведущий из q' в q или $q' = q$;
- $f(q',t) = \bullet$.

Обозначим через *dist(q', q)* длину кратчайшего пути в графе потока управления, ведущего из q' в q ($dist(q, q) \equiv 0$). Тогда оценочную функцию можно определить следующим образом:

$$-m_q(t) = \min_{q' \in \Pi(q,t)} dist(q',q) \quad (*****)$$

Выражение, стоящее справа, определяет, за какое минимальное количество переходов можно добраться до элемента покрытия *q* от уже покрытых элементов из множества Q_S^F. Функция $m_q(t)$ принимает значение 0 на тех и только тех тестах, которые покрывают элемент *q*. Заметим, что

$$\begin{cases} m_q(t) = 0 \iff f(q,t) = \bullet \ ; \\ m_q(t) < 0 \iff f(q,t) = \perp . \end{cases}$$

Покрытие ветвей потока управления

Тестовый набор удовлетворяет критерию покрытия ветвей потока управления, если при выполнении этого тестового набора управление

хотя бы один раз проходит по каждому ребру графа потока управления. Заметим, что любой тестовый набор, удовлетворяющий этому критерию, удовлетворяет также и критерию покрытия операторов исходного кода. Обратное утверждение, однако, неверно. Элементами тестового покрытия являются переходы в графе потока управления. С каждым переходом в графе потока управления можно связать условие, при котором этот переход может быть выполнен. Переход от оператора *q* к оператору *r*, с которым связано условие *p*, обозначим как $q \xrightarrow{p} r$. Для выполнения перехода $q \xrightarrow{p} r$ необходимо и достаточно, чтобы был выполнен оператор *q*, и чтобы после этого условие *p* обратилось в истину. Соответственно, для тестов, не покрывающих оператор *q*, в качестве оценочной подходит функция m_q, определённая уравнением (*****), так как истинность условия *p* для оценки таких тестов роли не играет. Для тестов, покрывающих оператор *q*, функция $m_q(t)$ обращается в 0. Для таких тестов оценочная функция должна определять, насколько близок тест к тесту, для которого после выполнения оператора *q* будет истинным условие *p*. Таким образом, в общем виде оценочную функцию для критерия покрытия ветвей потока управления можно определить следующим образом:

$$m_{q \xrightarrow{p} r}(t) = \left[\begin{array}{l} m_q(t), \textbf{при } f(q,t) = \perp, \\ \mu_{q,p}(t), \textbf{при } f(q,t) = \bullet \end{array} \right. .$$

Значение функции $\mu_{q,p} : T \to R^+$ тем больше, чем ближе заданный тест к тесту, в котором условие *p* выполняется после выполнения оператора *q*. При этом функция $\mu_{q,p}(t)$ достигает своего максимума на тех и только тех тестах, в которых после выполнения оператора *q* выполняется условие *p*, то есть тех, которые покрывают переход $q \xrightarrow{p} r$.

Функцию $\mu_{q,p}(t)$ можно определять по-разному в зависимости от характера условия *p*. Если условие имеет форму простого (не)равенства $x\Diamond y$, где « $\Diamond$ » обозначает одно из отношений « < », « > », « = », « ≤ » или « ≥ », то для определения функции $\mu_{q,p}(t)$ можно использовать значение $|x-y|$, например, следующим образом:

$$\mu_{q,p}(t) = \left[\begin{array}{l} 2, \text{при } x\Diamond y, \\ e^{-|x-y|}, \text{при } \neg(x\Diamond y). \end{array}\right.$$

Если условие представляет собой конъюнкцию $p = p_1 \wedge p_2 \wedge \ldots \wedge p_n$, то в качестве значения функции $\mu_{q,p}(t)$ можно взять количество членов этой конъюнкции, принимающих значение «истина». В общем случае эффективно определить функцию $\mu_{q,p}(t)$ затруднительно.

Покрытие путей потока управления

Тестовый набор удовлетворяет критерию покрытия путей потока управления, если его выполнение хотя бы один раз проходит по каждому возможному пути в графе потока управления ведущему от точки входа до точки завершения работы. Этот критерий сильнее критерия покрытия ветвей потока управления. Каждый путь представляет собой последовательность переходов $R = \tau_1, \tau_2, \ldots, \tau_n$, где τ_i имеет вид $q_i \xrightarrow{p_i} q_{i+1}$, при $1 \le i \le n$. Упорядоченным подмножеством пути $\tau_1, \tau_2, \ldots, \tau_n$ назовём последовательность $\tau_{i_1}, \tau_{i_2}, \ldots, \tau_{i_m}$ такую, что $1 \le i_1 < i_2 < \ldots < i_m \le n$. Заметим, что в упорядоченном подмножестве пути конечный оператор перехода может не совпадать с начальным оператором следующего за ним перехода.

Пусть есть два пути $R' = \tau'_1, \tau'_2, \dots, \tau'_l$ и $R'' = \tau''_1, \tau''_2, \dots, \tau''_k$, и пусть

$$\begin{cases} \tau'_{i_1} = \tau''_{j_1}, \\ \tau'_{i_2} = \tau''_{j_2}, \\ \quad \dots \\ \tau'_{i_m} = \tau''_{j_m}, \end{cases}$$

причём $1 \le i_1 < i_2 < \dots < i_m \le l$ и $1 \le j_1 < j_2 < \dots < j_m \le k$. Тогда пути R' и R'' имеют общее упорядоченное подмножество размера *m*.

Обозначим через *length(R)* длину пути *R*, а через $common(R', R'')$ - максимальный размер общего упорядоченного подмножества путей R' и R''. Определим оценочную функцию для критерия покрытия путей потока управления следующим образом:

$$-m_R(t) = length(R) + length(path(t)) - 2 \cdot common(R, path(t)).$$

Здесь *path(t)* - это путь, по которому приходит управление при выполнении теста *t*. Значение в правой части равно количеству переходов в путях *R* и *path(t)*, не входящих в максимальное общее упорядоченное подмножество этих путей. Оно равно 0 тогда и только тогда, когда пути *R* и *path(t)* совпадают.

В заключение отметим, что применение генетических алгоритмов для генерации тестов предъявляет дополнительные требования к используемым критериям полноты тестового покрытия. Это вызвано тем, что критерий полноты используется не только для оценки качества сгенерированных тестов, но и непосредственно в процессе генерации для оценки близости полученных тестов к нужным результатам. Таким образом, нужно иметь оценочную функцию, позволяющую измерить эту близость, определить, насколько

перспективными являются уже построенные тесты с точки зрения их использования в качестве основы для построения новых тестов. Кроме того, нужно иметь в виду, что тривиальные решения - функции вида «покрыто - 0, не покрыто - 1» - работают очень плохо. Для критериев, связанных с покрытием тех или иных путей в коде программы, удается построить достаточно удобные оценочные функции, основанные на количестве непокрытых дуг в пути, который нужно покрыть.

6.2. Генетические алгоритмы, распознающие изображений

Генетические алгоритмы достаточно широко используются в задачах оптимизации и обучения нейросетей.
Сами алгоритмы являются итеративными, и, дают лишь приближенноезначение, что, однако, с лихвой компенсируется областью их применения.
Разберем устройство одного из таких алгоритмов на примере распознавания простейшего изображения.
Оперировать мы будем популяциями хромосом (особей), так как алгоритм является итеративным, номер текущей итерации назовем текущей эпохой.
Перед составлением алгоритма определим, что же будет являться нашей задачей, и что будет являться её решением:
Рассмотрим пример нахождения коэффициентов, в уравнении параболы, исходя из нарисованного от руки изображения. В этом случае задача – найти такие коэффициенты, при которых парабола будет максимально точно совпадать с рисунком, решение задачи – набор из трех коэффициентов в уравнении параболы.

Алгоритм предусматривает популяцию неких объектов (хромосом), которые будут бороться за выживание.
Итак, **Хромосома** – это возможное решение нашей задачи, не важно какое, правильное или нет.
Ген – элементарная частичка информации, в рамках данной задачи, у нас будет три гена – соответственно по одному на каждый коэффициент.
Популяция – набор хромосом текущей эпохи.
Первоначально мы создаём популяцию (желательно из нескольких тысяч хромосом), и заполняем гены произвольной информацией,

которая не противоречит условию задачи.
Как и в реальном мире, наши хромосомы будут размножаться и подвергаться различным мутациям. За эти действия отвечают операторы скрещивания (кроссовер) и мутации.

Кроссовер отвечает за передачу признаков родителей своим потомкам, в самой простой реализации он создаёт новую хромосому из генов двух родителей, «донор» очередного гена выбирается произвольным образом.
Оператор мутации призван внести разнообразия в наш островок цифровой жизни, под его действием у хромосомы изменяется произвольный ген.
Fitness-функция: Как и в жизни – не приспособленные виды должны погибать, «чистить» нашу популяцию будет последняя функция, называемая функцией приспособления, или Fitness-функцией.
Для каждой хромосомы эта функция возвращает число, обратно пропорциональное «приспособленности» этой хромосомы, на нее накладываются условия:
Значение Fitness от идеального решения = 0
Это основное условие, но все же желательно подобрать функцию так, чтобы ее значение росло, по мере удаления решения от идеального. Иными словами – чтобы она имела только один минимум.

Теперь можно сформулировать вариант алгоритма
1. Обозначить номер эпохи = 0, создать популяцию хромосом, в генах которых будет произвольная, не противоречащая задаче, информация
2. Посчитать приспособленность популяции
3. Выбрать из популяции особь А
4. С вероятностью Р(скрещивания) выбрать особь В и применить оператор кроссовера, результирующую особь занести в новую популяцию.
5. С вероятностью Р(мутации) выполнить мутацию произвольной особи, результирующую особь занести в новую популяцию
6. Выполнить операции 3,4,5 n раз, где n >= размера популяции
7. Создать новую популяцию из лучших особей существующей и только что сформированной популяции
8. Увеличить номер текущей эпохи
9. Если результат работы удовлетворителен – остановка алгоритма, иначе – переход к шагу 2.

Генетический алгоритм не является строго детерминированным, например, мы можем сначала выполнить скрещивание всех особей, а потом мутацию (в рамках данной задачи именно этот подход оказался более эффективным). Итак, попробуем, на основе описанного, решить поставленную задачу. Сначала нам понадобится определить хромосому, которая и будет хранилищем информации о генах.

```
1.  public class Chromosome : IComparable
2. {
3.      public double[] gens;
4.      static Random random = new Random();
5.
6.      public Chromosome()
7.      {
8.          this.gens = new double[3];
9.
10.                 for (int i = 0; i < 3; i++)
11.                 {
12.                     this.gens[i] =
    random.NextDouble() * 100;
13.                 }
14.                 this.fit = fitness(this);
15.             }
16.             public int CompareTo(object obj)
17.         }
```

* This source code was highlighted with Source Code Highlighter.

Значение 100 выбрано для примера, это не ограничит общности, даже если искомая парабола будет иметь кофэффициенты >100
Отметим метод CompareTo, позволяющий нам сортировать списки объектов Chromosome.

```
1.  public class Population
```

```
2. {
3.      Random random = new Random();
4.      public List<Chromosome> population;
5.      public int count;
6.      public double crossProbability;
7.      public double MutationProbability;
8.      public int age;
9.
10.             public Population()
11.             {
12.                 for (int i = 0; i < 5000; i++)
13.                 {
14.                     this.population.Add(new
   Chromosome());
15.                 }
16.             }
17.
18.             public void GoToNextGeneration()
19.             {
20.                 Chromosome chromosome;
21.                 for (int i = 0; i < count; i++)
22.                 {
23.                     chromosome = population[i];
24.
25.                     if (random.NextDouble() <
   crossProbability)
26.                     {
27.                         population.Add(Cross(ch
   romosome,population[random.Next(count)]));
28.                     }
29.                 }
30.
31.                 for (int i = 0; i <
   population.Count; i++)
32.                 {
33.                     if (random.NextDouble() <
   MutationProbability)
34.                     {
35.                         population.Add(Mutate(p
   opulation[i]));
```

```
36.                     }
37.                 }
38.
39.                 population.Sort();
40.                 population.RemoveRange(count,po
   pulation.Count-count);
41.
42.                 age++;
43.             }
44.
45.             Chromosome Cross(Chromosome a,
   Chromosome b)
46.             Chromosome Mutate(Chromosome a)
47.             double Fitness(Chromosome a)
48.         }
```

* This source code was highlighted with Source Code Highlighter.

Вот она, эволюция в двадцати строчках! Это и будет сердцем нашей программы.
В первой части мы получаем потомство от уже существующей популяции, и добавляем его в конец списка, т.е. все «дети» будут находится подномерами большими чем count.
Второй шаг – мутация, было бы логично мутировать саму хромосому, а не добавлять мутировавший клон, но в этом случае мы теряем содержащееся в исходной хромосоме решение, а оно может быть лучшим в популяции.
Наконец, в конце мы сортируем всю нашу получившуюся популяцию по возрастанию Fitness функции, (следовательно – по убыванию правильности решения) удаляем наименее «приспособленные» хромосомы, и увеличиваем номер текущей эпохи.

Давайте рассмотрим, что же представляют собой три последние функции, производящии операции над нашими хромосомами?

```
1. Chromosome Cross(Chromosome a, Chromosome b)
2. {
```

```
3.      double[] pair = new double[3];
4.
5.      for (int i = 0; i < 3; i++)
6.      {
7.          if ((random.Next() % 2) == 0)
8.          {
9.              pair[i] = a.Gens[i] + (b.Gens[i]
   - a.Gens[i]) * 0.1;
10.              }
11.              else
12.              {
13.                  pair[i] = b.Gens[i] -
   (b.Gens[i] - a.Gens[i]) * 0.1;
14.              }
15.          }
16.
17.          Chromosome result = new
   Chromosome(pair);
18.          return result;
19.      }
```

* This source code was highlighted with Source Code Highlighter.

Реализация оператора кроссовера может быть достаточно разнообразной, и, обычно, подбирается под конкретную задачу. Здесь, с вероятностью 0.5, для каждого гена, потомок получит его от первого родителя или, с такой же вероятностью, от второго родителя. Поправка (b.Gens[i] — a.Gens[i]) * 0.1 сделана чтобы както разнообразить популяцию новыми генотипом.

```
1.  Chromosome Mutate(Chromosome a)
2.  {
3.      double[] pair = (double[])a.Gens.Clone();
4.      int geneNum = random.Next(3);
5.      pair[geneNum] = random.NextDouble() *
   100;
```

```
6.      return new Chromosome(pair,
   a.chromosomeSettings);
7. }
```

* This source code was highlighted with Source Code Highlighter.

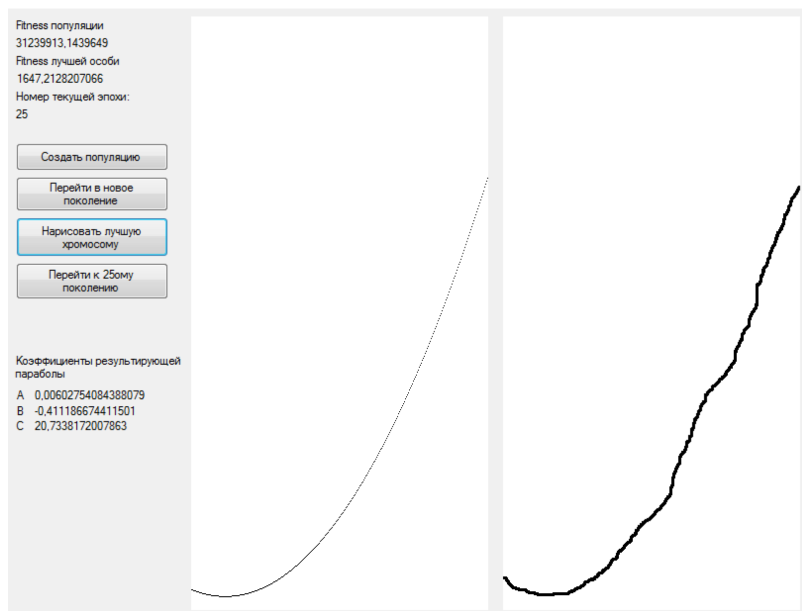
Как и обещано, оператор мутации изменяет один произвольный ген. Взглянем на самое интересное, как же мы будем определять, насколько близка наша хромосома к идеальному решению?

```
1. double GetFitness(Chromosome a)
2. {
3.      double result = 0;
4.      double temp;
5.      for (int i = 0; i < funcLength; i++)
6.      {
7.          temp = Math.Abs(Func(i, a.Gens) -
   funcArray[i]);
8.          result += temp;
9.      }
10.
11.         return result;
12.     }
13.
14.     double Func(double x, double[] gens)
15.     {
16.         double result = 0;
17.         for (int i =2; i >= 0; i--)
18.         {
19.             result += gens[2-i] *
   Math.Pow(x, i);
20.         }
21.         return result;
22.     }
```

* This source code was highlighted with Source Code Highlighter.

Массив funcArray получается сканированием картинки снизу вверх по столбцам, каждый столбец – индекс массива, высота первой найденной черной точки в столбце – значение.
Функция Func – возвращает значение уравнения 2й степени в точке x, с коэффициентами из массива gens
Получается Fitness – это сумма «погрешностей», для каждой точки нашей параболы, которая есть на рисунке. Такая функция не имеет единственный минимум, но, тем не менее, обеспечивает достаточную сходимость алгоритма.

Результат:



При количестве хромосом в популяции = 5000, на 25ом поколении мы получаем достаточно близкое сходство.
Данный алгоритм можно расширить на кривые любого порядка =)

6.3. Генетические алгоритмы в MATLAB

6.3.1. Суть генетических алгоритмов

Данный раздел посвящен решению оптимизационных задач при помощи генетических алгоритмов в среде **MATLAB**. В работе используется большой объем данных: он обусловлен тем, что основной поставленной задачей было подробно раскрыть каждый из настраиваемых в MATLAB параметров работы генетических алгоритмов.

Генетические алгоритмы – это метод решения оптимизационных задач, основанный на биологических принципах естественного отбора и эволюции. Генетический алгоритм повторяет определенное количество раз процедуру модификации популяции (набора отдельных решений), добиваясь тем самым получения новых наборов решений (новых популяций). При этом на каждом шаге из популяции выбираются «родительские особи», то есть решения, совместная модификация которых (скрещивание) и приводит к формированию новой особи в следующем поколении. Генетический алгоритм использует три вида правил, на основе которых формируется новое поколение: правила отбора, скрещивания и мутации. Мутация позволяет путем внесения изменений в новое поколение избежать попадания в локальные минимумы оптимизируемой функции.

(Под катом основная часть + несколько скриншотов).

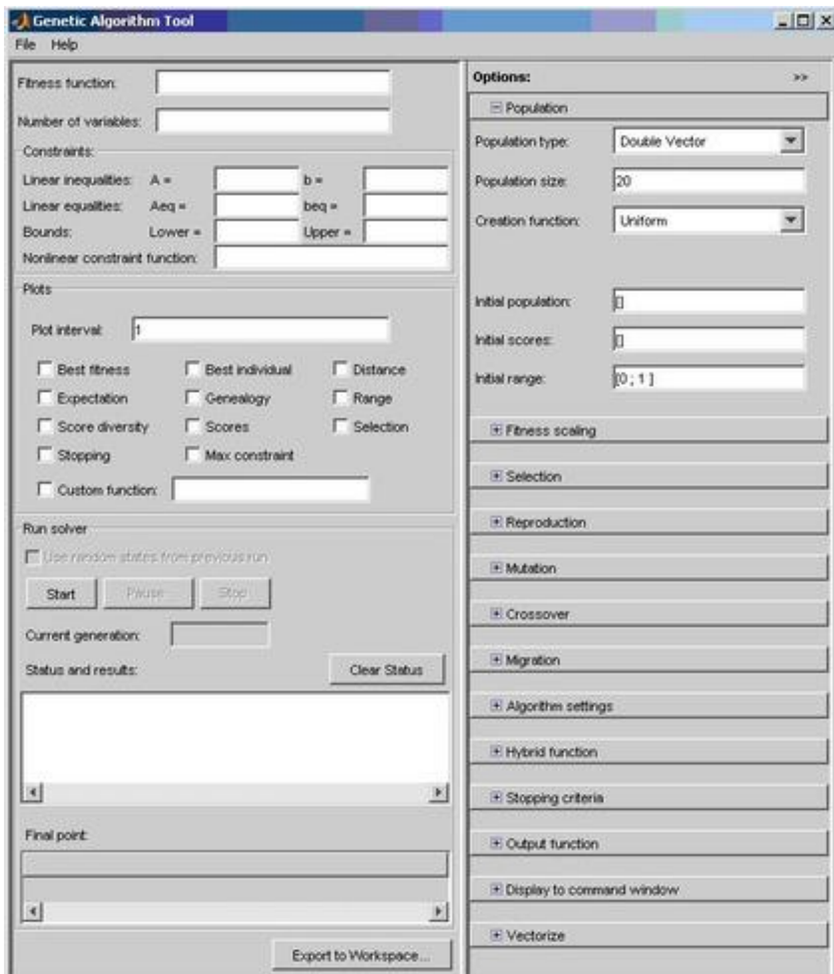
Механизм работы с генетическими алгоритмами в среде MATLAB реализован двумя способами:

1. Вызов функции генетических алгоритмов
2. Использование комплекта Genetic Algorithm Tool

Оба способа поставляются в числе стандартного набора функций и модулей MATLAB. Мы считаем , что намного более удобным и наглядным является второй способ работы с генетическими алгоритмами в MATLAB, связанный с использованием модуля Genetic Algorithm Tool. Его и рассмотрим подробнее.

6.3.2. Работа с GENETIC ALGORITHM TOOL

Для запуска пакета Genetic Algorithm Tool следует в командной строке MATLAB выполнить команду gatool. После этого запустится пакет генетических алгоритмов и на экране появится основное окно утилиты.



В поле Fitness function указывается оптимизируемая функция в виде @fitnessfun, где fitnessfun.m – название M-файла, в котором предварительно следует описать оптимизируемую функцию. Отметим, что M-файл создается в среде MATLAB через меню File->New->M-File. Пример описания некоторой функции my_fun в M-файле:

```
function z = my_fun(x)
z = x(1)^2 — 2*x(1)*x(2) + 6*x(1) + x(2)^2 — 6*x(2);
```

Вернемся к основному окну утилиты GATool. В поле Number of variables указывается длина входного вектора оптимизируемой функции. В рассмотренном выше примере функция my_fun имеет входной вектор длины 2.
В панели Constraints можно задать ограничения или ограничивающую нелинейную функцию. В поле Linear inequalities задается линейное ограничение неравенством вида:

$A*x \leq b.$

В поле Linear equalities данной панели задаются линейные ограничения равенством:

$A*x = b.$

В обоих случаях A – некоторая матрица, b – вектор.

В поле Bounds в векторном виде задаются нижнее и верхнее ограничения переменных, а в поле Nonlinear constraint function можно задать произвольную нелинейную функцию ограничений.

Если в конкретной задаче не требуется задание ограничений, все поля панели Constraints следует оставить незаполненными.

Ниже находится панель настройки графиков. Она позволяет выводить различные графики, отображающие информацию о работе генетического алгоритма. На основе этой информации можно менять настройки алгоритма с целью повышения эффективности его работы. Например, выбор опции Best Fitness в этой панели позволяет вывести на одном графике лучшее и среднее значение оптимизируемой функции для каждого поколения работы алгоритма. Подробнее эта

панель будет описана ниже наравне с остальными панелями вкладки Options утилиты GATool.

Панель Run Solver содержит управляющие элементы (кнопки Start, Pause и Stop для начала, временной и полной остановки работы генетического алгоритма). Также она содержит поля Status and results, в которое выводятся текущие результаты работы запущенного генетического алгоритма, и Final point, в котором выводится значение конечной точки работы алгоритма — наилучшей величины оптимизируемой функции (то есть, искомое значение).

В правой части основного окна утилиты GATool находится панель Options. Она позволяет устанавливать различные настройки для работы генетических алгоритмов. При щелчке мышью по кнопкам [+], которые находятся напротив названия каждого из настраиваемых параметров в панели Options, появляются выпадающие списки (вкладки), содержащие поля для ввода и изменения соответствующих параметров генетического алгоритма.

Основными настраиваемыми параметрами в GATool являются:
— популяция (вкладка Population);
— масштабирование (вкладка Fitness Scaling);
— оператор отбора (вкладка Selection);
— оператор репродукции (вкладка Reproduction);
— оператор мутации (вкладка Mutation);
— оператор скрещивание (вкладка Crossover);
— перенесение особей между популяциями (вкладка Migration);
— специальные параметры алгоритма (вкладка Algorithm settings);
— задание гибридной функции (вкладка Hybrid function);
— задание критерия остановки алгоритма (вкладка Stopping criteria);
— вывод различной дополнительной информации по ходу работы генетического алгоритма (вкладка Plot Functions);
— вывод результатов работы алгоритма в виде новой функции

(вкладка Output function);
— задание набора информации для вывода в командное окно (вкладка Display to command window);
— способ вычисления значений оптимизированной и ограничивающей функций (вкладка User function evaluation).

Рассмотрим подробнее все вышеперечисленные вкладки панели Options и элементы, которые они содержат.

Во вкладке настройки популяций пользователь имеет возможность выбрать тип математических объектов, к которому будут относиться особи всех популяций (двойной вектор, битовая строка или пользовательский тип). При этом стоит учитывать, что использование битовой строки и пользовательских типов накладывают ограничения на перечень допустимых операторов создания, мутации и скрещивания особей. Так, например, при выборе в качестве формы представления особей битовой строки для оператора скрещивания нельзя использовать гибридную функцию или нелинейную ограничивающую функцию.
Также вкладка популяции позволяет настраивать размер популяции (из скольких особей будет состоять каждое поколение) и каким образом будет создаваться начальное поколение (Uniform – если отсутствуют накладываемые ограничения, в противном случае — Feasible population). Кроме того, в рассматриваемой вкладке имеется возможность задать вручную начальное поколение (используя пункт Initial population) или его часть, начальный рейтинг особей (пункт Initial scores), а также ввести ограничительный числовой диапазон, которому должны принадлежать особи начальной популяции (Initial range).

Во вкладке масштабирования (Fitness Scaling) пользователь имеет возможность указать функцию масштабирования, которая конвертирует достигаемые оптимизируемой функцией значения в значения, лежащие в пределах, допустимых для оператора отбора. При выборе в качестве функции масштабирования параметра Rank масштабирование будет приводиться к рейтингу, то есть особям присваивается рейтинговый номер (для лучшей особи – единица, для следующей – двойка, и так далее). Пропорциональное масштабирование (Proportional) задает вероятности пропорционально заданному числовому ряду для особей. При выборе опции Top

наибольшее рейтинговое значение присваивается сразу нескольким наиболее выдающимся особям (их число указывается в виде параметра). Наконец, при выборе масштабирования типа Shift linear имеется возможность указать максимальную вероятность наилучшей особи.

Вкладка Selection позволяет выбрать оператор отбора родительских особей на основе данных из функции масштабирования. В качестве доступных для выбора вариантов оператора отбора предлагаются следующие:

— Tournament – случайно выбирается указанное число особей, среди них на конкурсной основе выбираются лучшие;
— Roulette – имитируется рулетка, в которой размер каждого сегмента устанавливается в соответствии с его вероятностью;
— Uniform – родители выбираются случайным образом согласно заданному распределению и с учетом количества родительских особей и их вероятностей;
— Stochastic uniform – строится линия, в которой каждому родителю ставится в соответствие её часть определенного размера (в зависимости от вероятности родителя), затем алгоритм пробегает пот линии шагами одинаковой длины и выбирает родителей в зависимости от того, на какую часть линии попал шаг.

Вкладка Reproduction уточняет каким образом происходит создание новых особей. Пункт Elite count позволяет указать число особей, которые гарантировано перейдут в следующее поколение. Пункт Crossover fraction указывает долю особей, которые создаются путем скрещивания. Остальная доля создается путем мутации.

Во вкладке оператора мутации выбирается тип оператора мутации. Доступны следующие варианты:

— Gaussian – добавляет небольшое случайное число (согласно распределению Гаусса) ко всем компонентам каждого вектора-особи;
— Uniform – выбираются случайным образом компоненты векторов и вместо них записываются случайные числа из допустимого диапазона;
— Adaptive feasible – генерирует набор направлений в зависимости от последних наиболее удачных и неудачных поколений и с учетом налагаемых ограничений продвигается вдоль всех направлений на

разную длину;
— Custom – позволяет задать собственную функцию.

Вкладка Crossover позволяет выбрать тип оператора скрещивания (одноточечное, двухточечное, эвристическое, арифметическое или рассеянное (Scattered), при котором генерируется случайный двоичный вектор соответствия родителей). Также имеется возможность задания произвольной (custom) функции скрещивания.

Во вкладке Migration можно настраивать правила, согласно которым особи будут перемещаться между подпопуляциями в пределах одной популяции. Подпопуляции создаются, если в качестве размера популяции указан вектор, а не натуральное значение. В данной вкладке можно указать направление миграции (forward – в следующую подпопуляцию, both – в предыдущую и следующую), долю мигрирующих особей и частоту миграции (сколько поколений проходит между миграциями). Если создание подпопуляций не требуется, эту вкладку всегда стоит оставлять без изменений.

Вкладка специальных опций алгоритма позволяет настраивать параметры решения системы нелинейных ограничений, налагаемых на алгоритм. Значение параметра Initial penalty определяет начальное числовое значение критики алгоритма, Penalty factor используется как множитель этого значения в случаях, когда разработчика не устраивает точность оптимизации или при выходе за границы, определенные во вкладке ограничений. Как правило, эти опции детально настраиваются для решения задач высокой сложности.

Вкладка Hybrid function позволяет задать ещё одну функцию минимизации, которая будет использоваться после окончания работы алгоритма. В качестве возможных гибридных функций доступны следующие встроенные в саму среду MATLAB функции:
− none (не использовать гибридную функцию);
− fminsearch (поиск минимального из значений);
− patternsearch (поиск по образцу);
− fminunc (для неограниченного алгоритма);
− fmincon (для алгоритма с заданными ограничениями).

Во вкладке критерия остановки (Stopping criteria) указываются ситуации, при которых алгоритм совершает остановку. При этом,

настраиваемыми являются следующие параметры:

— Generations – максимальное число поколений, после превышения которого произойдет остановка;
— Time limit – лимит времени на работу алгоритма;
— Fitness limit – если оптимизируемое значение меньше или равно данного лимита, то алгоритм остановится;
— Stall generations – количество мало отличающихся поколений, по прошествии которых алгоритм остановится;
— Stall time limit – то же, что и предыдущий параметр, но применимо к времени работы алгоритма;
— Function tolerance и Nonlinear constraint tolerance – минимальные значения изменений оптимизируемой и ограничивающей функций соответственно, при которых алгоритм продолжит работу.

Особый интерес представляет вкладка Plot Functions, которая позволяет выбирать различную информацию, которая выводится по ходу работы алгоритма и показывает как корректность его работы, так и конкретные достигаемые алгоритмом результаты. Наиболее важными и используемыми для отображения параметрами являются:

— Plot interval – число поколений, по прошествии которого происходит очередное обновление графиков;
— Best fitness – вывод наилучшего значения оптимизируемой функции для каждого поколения;
— Best individual – вывод наилучшего представителя поколения при наилучшем оптимизационном результате в каждом из поколений;
— Distance – вывод интервала между значениями особей в поколении;
— Expectation – выводит ряд вероятностей и соответствующие им особи поколений;
— Genealogy – вывод генеалогического дерева особей;
— Range – вывод наименьшего, наибольшего и среднего значений оптимизируемой функции для каждого поколения;
— Score diversity – вывести гистаграмму рейтинга в каждом поколении;
— Scores – вывод рейтинга каждой особи в поколении;
— Selection – вывод гистограммы родителей;
— Stopping – вывод информации о состоянии всех параметров, влияющих на критерии остановки;
— Custom – отображение на графике некоторой указанной

пользователем функции.

Вкладка вывода результатов в виде новой функции (Output function) позволяет включить вывод истории работы алгоритма в отдельном окне с заданным интервалом поколений (флаг History to new window и поле Interval соответственно), а также позволяет задать и вывести произвольную выходную функцию, задаваемую в поле Custom function.

Вкладка User function evaluation описывает, в каком порядке происходит вычисление значений оптимизируемой и ограничивающей функций (отдельно, параллельно в одном вызове или одновременно).

Наконец, вкладка Display to command window позволяет настраивать информацию, которая отображается в основном командном окне MATLAB при работе алгоритма. Возможны следующие значения: Off — нет вывода в командное окно, Iterative — вывод информации о каждой итерации работающего алгоритма, Diagnose — вывод информации о каждой итерации и дополнительных сведениях о возможных ошибках и измененных ключевых параметрах алгоритма, Final – выводится только причина остановки и конечное значение.

6.4. Аппроксимация изображений генетическим алгоритмом при помощи EvoJ

В этом разделе рассматривается вопрос, как можно применить генетический алгоритм для аппроксимации изображений полигонами.

6.4.1. Выбор способа описания решения

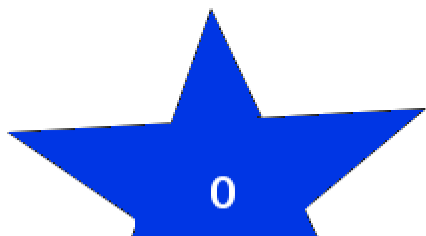
Следуя традиционному пути решения задач при помощи EvoJ, для начала выберем как мы опишем решения для нашей задачи. Прямолинейным решением будет описать аппроксимацию как простой список полигонов:

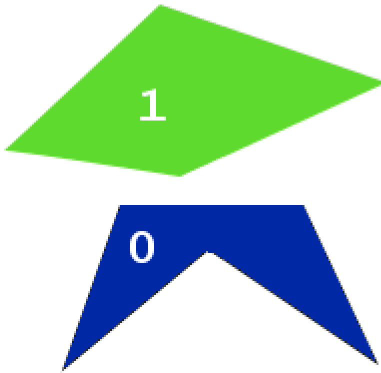
```
public interface PolyImage {

    List<Polygon> getCells();
}
```

Такой подход имеет право на жизнь, нам вполне удастся подобрать удовлетворительную картинку рано или поздно. Скорее всего поздно, так как такой подход снижает эффективность скрещивания хороших решений. И вот почему.

Представим для простоты следующую картинку.

Допустим у нас в генофонде есть два хороших решения, каждое из которых отличается от идеального на один полигон.

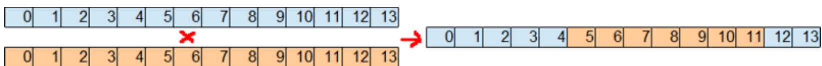




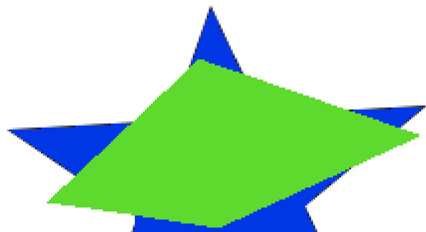
Обратите внимание на одну особенность: в обоих случаях полигон номер 0 идеально совпадает с одним из полигонов изображения, тогда когда другой лежит где-то в стороне. Был бы здорово скрестить эти два решения таким образом, чтобы в результате вместе оказались два первых полигона из обоих решений, однако это не возможно — EvoJ отображает все переменные на байтовый массив и при скрещивании работает с байтами, не меняя порядка их следования.

Polygon 1							Polygon 2						
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Процесс кроссинговера разъясняется на следующем рисунке.



Таким образом два элемента с одинаковым индексом никак не могут оказаться в результирующем решении.
Результат скрещивания скорее всего будет выглядеть так:



Что сильно скажется на эффективности скрещивания, значительно замедлив эволюцию.

Обойти проблему можно, «правильно» засеяв исходную популяцию — равномерно распределив полигоны по области изображения, таким образом, чтобы расположение полигона в списке, согласовывалось с его геометрическим расположением на рисунке. При этом придется ограничить мутацию, чтобы полигоны не могли слишком далеко «переползти».

Более продвинутый подход заключается в изначальном разделении изображения на ячейки, с назначением каждой ячейке собственного списка полигонов. В виде интерфейса Java это описывается так:

```
public interface PolyImage {

    Colour getBkColor();

    List<List<List<Polygon>>> getCells();
}
```

Внешний список задает ряды ячеек, следующий по вложенности — столбцы, и, наконец, внутренний список — полигоны находящиеся в в соответствующей ячейке. Такой подход автоматически обеспечивает примерное соответствие положения полигона в байтовом массиве и его геометрического места на рисунке.

Каждый полигон опишем как:

```
public interface Polygon {

    List<Point> getPoints();

    Colour getColor();

    int getOrder();
}
```

Здесь свойство order определяет глобальный порядок отрисовки полигона. Цвет полигона опишем как:

```
public interface Colour {

    @MutationRange("0.2")
    @Range(min = "0", max = "1", strict = "true")
    float getRed();

    @MutationRange("0.2")
    @Range(min = "0", max = "1", strict = "true")
    float getGreen();

    @MutationRange("0.2")
    @Range(min = "0", max = "1", strict = "true")
    float getBlue();

    void setRed(float red);

    void setGreen(float green);

    void setBlue(float blue);
}
```

Иными словами, просто как три компоненты цвета, желающие еще могут добавить альфа-канал, однако мы будем рисовать все полигоны с 50% прозрачностью, это снизит число безразличных мутаций. Назначение аннотаций здесь очевидно. Точки полигона опишем как:

```
public interface Point {

    int getX();

    int getY();
}
```

При этом координаты X и Y будем считать относительно центра ячейки, которой принадлежит полигон. Обратите внимание, что аннотации регулирующие допустимые значения переменных присутствуют только в описании цвета. Это связано с тем, что допустимые значения координат зависят от размера изображения и от конфигурации ячеек, а допустимые значения компоненты цвета — вещи предопределенные. Кроме того, аннотацию на два внутренних списка в интерфейсе `PolyImage` вообще никак повесить нельзя.

Чтобы задать все необходимые параметры мы воспользуемся механизмом под названием Detached

Перейдем к программированию.

6.4.2. Кофигурирование EvoJ при помощи Detached Annotations

```
        final HashMap<String, String> context = new
HashMap<String, String>();

        context.put("cells@Length",
ROWS.toString());
        context.put("cells.item@Length",
COLUMNS.toString());
        context.put("cells.item.item@Length",
POLYGONS_PER_CELL.toString());

context.put("cells.item.item.item.points@Length",
"6");
```

```
context.put("cells.item.item.item.points.item.x@Str
ictRange", "true");

context.put("cells.item.item.item.points.item.x@Min
", String.valueOf(-widthRadius));

context.put("cells.item.item.item.points.item.x@Max
", String.valueOf(widthRadius));

context.put("cells.item.item.item.points.item.x@Mut
ationRange", "20");

context.put("cells.item.item.item.points.item.y@Str
ictRange", "true");

context.put("cells.item.item.item.points.item.y@Min
", String.valueOf(-heightRadius));

context.put("cells.item.item.item.points.item.y@Max
", String.valueOf(heightRadius));

context.put("cells.item.item.item.points.item.y@Mut
ationRange", "20");

context.put("cells.item.item.item.order@Min", "0");

context.put("cells.item.item.item.order@Max",
"1000");
```

Здесь мы создаем контекст — `Map` похожий на `properties` файл, где имя свойства соответствует пути к свойству объекта в нотации похожей на принятую в `BeanUtils`, за исключением того, что элементы списка обозначаются ключевым словом `item`.
Таким образом, имя `cells` описывает свойство `cells` нашего корневого интерфейса `PolyImage`. А строчка `cells.item` – элементы этого списка, которые в свою очередь являются списками описываемыми строчкой `cells.item.item`, итд.

После имени свойства и знака `@` указывается имя detached аннотации, которое похоже на имя обычной аннотации. Обязательным является указание длины списка, чтобы EvoJ знала сколько места резервировать в байтовом массиве.
Аннотация `cells.item.item.item.points@Length` задает количество точек в полигоне (проследите путь — свойство `cells`, три вложенных списка, свойство `points` у элементов самого вложенного списка.
Схожим образом задаются границы значений координат точек, радиус их мутации, пределы значений свойства order.

Далее мы используем заполненный контекст при создании генофонда.

6.4.3. Создание фитнес функции

В качестве фитнес-функции выберем сумму квадратов отклонений подбираемого изображения от оригинального, со знаком минус — так как наилучшим решениям соответствует наименьшая сумма. Фитнес-функция имплементирована в классе `ImageRating`, остановимся на его ключевых моментах.

Здесь класс наследуется от helper-класса `AbstractSimpleRating`:

```
public class ImageRating extends
AbstractSimpleRating<PolyImage>
```

и имплементирует абстрактный метод doCalcRating следующим образом:

```
   @Override
    protected Comparable doCalcRating(PolyImage
solution) {
        BufferedImage tmp = drawImage(solution);
        return compareImages(originalImage, tmp);
    }
```

Здесь функция `drawImage` тривиально отрисовывает все полигоны в соответствии с их ячейкой, порядком, цветом, итд.

Функция же `compareImages` осуществляет попиксельное сравнение изображений. Рассмотрим ее подробнее

```
  private Comparable compareImages(BufferedImage
originalImage, BufferedImage tmp) {
        long result = 0;
        int[] originalPixels =
originalImage.getRaster().getPixels(0, 0,
originalImage.getWidth(),
originalImage.getHeight(), (int[]) null);
        int[] tmpPixels =
tmp.getRaster().getPixels(0, 0, tmp.getWidth(),
tmp.getHeight(), (int[]) null);

        for (int i = 0; i < originalPixels.length;
i++) {
            long d = originalPixels[i] -
tmpPixels[i];
            result -= d * d;
        }

        return result;
    }
```

Как видно, функция довольно тривиальна — она берет растры оригинального и свеженарисованного изображений и сравнивает их поэлементно, попутно складывая (со знаком минус) квадраты разниц.

6.4.4. Осуществление итераций

Для того, чтобы быстрее достичь результата необходимо выбрать оптимальные параметры мутации:

- вероятность того что решении вообще подвергнется мутации

- вероятность мутации каждой переменной внутри решения
- радиус изменения — максимальное расстояние на которое может сместиться точка в ходе мутации
- срок жизни решения (не совсем относится к мутации, но так же влияет на скорость сходимости)

Выбор стратегии — дело не тривиальное. С одной стороны, сильная мутация позволяет быстрее охватить пространство решений и «нащупать» глобальный экстремум, но с другой стороны не позволяет в него скатиться — решения будут его постоянно проскакивать. Долгий срок жизни решения страхует от ухудшения значения фитнес функции, но замедляет общий прогресс и способствует скатыванию в локальный экстремум вместо глобального. Еще нужно решить, что нужно мутировать с большей вероятностью — точки полигонов или их цвета.
Кроме того, выбранная стратегия через некоторое время себя исчерпывает — найденные решения начинают колебаться вокруг найденного экстремума, вместо того чтобы в него скатиться. Чтобы справиться с этой проблемой, стратегии надо менять. Генеральная идея при этом заключается в том, чтобы уровень мутации со временем уменьшался, а время жизни удачных решений росло. Стратегии мутации, использованные в примере, подобраны эмпирическим путем.

Учитывая все выше написанное, главный цикл программы устроен следующим образом:

1. произвести 10 итерации генетического алгоритма
2. взять значение фитнес-функции у наилучшего решения
3. определить не пришла ли пора менять стратегию
4. изменить стратегию, если необходимо
5. сохранить картинку во временный файл
6. обновить UI

Запустив программу, минут через 15 (в зависимости от мощности вашей машины) вы обнаружите, что подобранное изображение уже

отдаленно напоминает оригинал, а через час-два вы достигнете результата близкого к тому, что приведен в начале статьи.

6.4.5. Улучшение алгоритма

Процесс можно значительно ускорить применив следующие оптимизации:

- аппроксимацию производить в пространстве Lab вместо RGB, оно характеризуется тем, что декартово расстояние между точками в цветовом пространстве примерно согласуется с воспринимаемой глазом разницей. Количества итераций в секунду это не добавит, зато направит эволюцию в нужном направлении.
- создать маску важности областей изображения — черно-белое изображение с выделенными областями интереса, которое затем использовать при расчете фитнес-функции, это позволит сконцентрировать эволюцию на том, что действительно представляет интерес.
- для изображений использовать VolatileImage. На моей машине рисование на VolatileImage в 10 раз быстрее чем рисование на BufferedImage. Правда, потом результат приходится все равно конвертировать в BufferedImage, чтобы получить цвета пикселей, это приводит к существенному падению быстродействия, но все равно окончательный результат в 3 раза лучше, чем просто рисовать полигоны на BufferedImage.
- подобрать оптимальные параметры мутации на разных этапах. Задача это не простая, но и тут может помочь генетический алгоритм. Я проводил эксперименты, где переменными были параметры мутации, а фитнес-функцией — средняя скорость уменьшения ошибки за 100 итераций. Результаты обнадеживают, однако для решения этой задачи требуется значительная вычислительная мощность.
- завести несколько независимых генофондов и производить эволюцию в них независимо, через определенные промежутки времени скрещивать между собой особей из разных генофондов. Такой подход называют островной моделью ГА,

т. е. эволюция как бы протекает на изолированных друг от друга островах, скрещивания между особями с разных островов крайне редки.

- засевать изображение полигонами постепенно: сначала поместить по 1-2 полигона в каждую ячейку, позволить им «расползтись», затем добавить еще по 1-2 полигона в места, где наблюдается наибольшее отклонение изображения от оригинала и подвергать эволюции только вновь добавленные полигоны, так повторять пока не будет достигнут предел числа полигонов в ячейке, после чего запустить эволюцию по всему изображению, как описано в статье выше. Такой подход приводит к наиболее точным близким аппроксиммациям.

Итак, мы рассмотрели пример, как можно применить EvoJ для решения задачи аппроксиммации изображения. Стоит отметить, что аппроксимация изображений генетическим алгоритмом представляет собой скорее академический или эстетический интерес нежели практический, ибо подобного результата можно добиться другими, специально заточенными алгоритмами векторизации изображений.

В следующий раз я расскажу о применении генетического алгоритма для генерации дизайнерских идей.

UPD1: Так как ссылка на исходники данная в тексте работы не бросается в глаза, дублируем ее здесь http://evoj.sourceforge.net/static/demos/img-demo.rar

6.5. Разработка и исследование гибридного алгоритма решения сложных задач оптимизации

6.5.1. Генетический алгоритм (ГА)

Название данного алгоритма объясняется тем, что в основе него лежит имитация процессов происходящих в природе, среди особей какой-либо популяции. Индивид или особь представляет собой решение, закодированное произвольным образом, например в бинарную строку. Совокупность решений в фиксированный момент времени составляет популяцию. Индивиды текущей популяции конкурируют друг с другом за передачу своей генетической информации (создание потомков) в следующую популяцию. Отобранные индивиды из текущей популяции с помощью *селекции,* проходят этапы создания новых решений-потомков - *рекомбинации* и *мутации*. Основными операторами генетического алгоритма будем называть операторы *скрещивания, селекции и мутации*.

Селекция - это оператор, с помощью которого происходит выбор индивида из текущей популяции для участия его в рекомбинации и мутации при получении потомка. Рассмотрим основные виды селекций.

Пропорциональная селекция

Пропорциональная селекция может быть выполнена в виде алгоритма *колеса рулетки*. В данной селекции каждому индивиду из текущей популяции назначается пригодность быть отобранным пропорционально его пригодности. Пусть N - число индивидов в популяции и f_i - пригодность i- го индивида в популяции тогда, например, если $N = 4$, $f_1 = f_2 = 10$, $f_3 = 15$, и $f_4 = 25$ колесо рулетки представлено на рисунке 1:

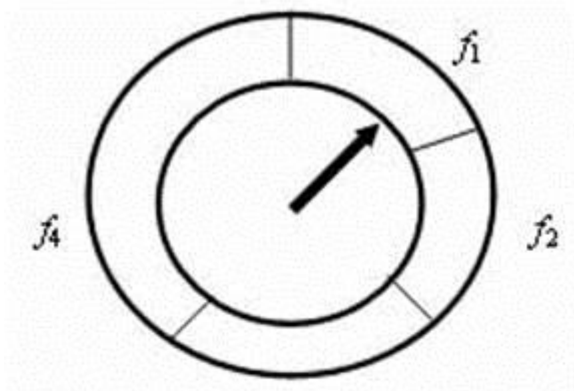


Рис. 1 **Пример колеса рулетки**

Проблемы с пропорциональной селекцией:

Преждевременная сходимость. Эта проблема возникает, когда на первых стадиях работы алгоритма в популяции появляется один или несколько супериндивидов, у которых достаточно большая пригодность, и они доминируют при выборе родителя для создания потомка. Вскоре каждая последующая популяция будет состоять из этих супериндивидов.

Стагнация

Ранговая селекция

В этом виде селекции индивиду назначается вероятность быть отобранным в зависимости от его места в упорядоченном ряду по пригодности индивидов текущей популяции. Таким образом, индивиды сортируются (ранжируются) на основе их пригодности таким образом, чтобы $f_i f_j$ для $i > j$.

Затем каждому индивиду назначается вероятность p_i быть отобранным.

Используемое распределение вероятностей:

Линейное: $p_i = ai + b$ ($a < 0$).

Преимущества:

Нет преждевременной сходимости, т.к. нет индивидов с $N_i >> 1$.

Нет стагнации, так как и к концу работы алгоритма N_1 N_2 … .

Нет необходимости в *явном вычислении пригодности*, т.к. для упорядочения индивидов достаточно иметь возможность их по парного сравнения.

Недостатки: значительные накладные расходы на переупорядочивание и трудность теоретического анализа сходимости.

Турнирная селекция

Одна из самых простых в реализации и эффективных в оптимизации является турнирная селекция. Для отбора индивида создается группа из M (M 2) индивидов, выбранных из текущей популяции случайным образом

Индивид с наибольшей пригодностью в группе отбирается, остальные - игнорируются

Преимущества:

Нет преждевременной сходимости

Нет стагнации

Не требуется глобальное переупорядочивание

Не требуется явное вычисление функции пригодности

Элитарная селекция

В данной селекции один или несколько лучших индивидов популяции всегда проходит в следующее поколение

Преимущество: гарантия сходимости, т.е. если глобальный максимум будет обнаружен, то ГА сойдется к этому максимуму

Недостаток: слабая глобальная сходимость, большой риск найти локальный минимумом

Скрещивание в ГА

Оператор *скрещивание* предназначен для поиска новых решений на основе отобранных *селекцией* родителей.

· *Одноточечное скрещивание* представляет собой разделение родительских хромосом в выбранной случайным образом общей точке и обмен правыми частями. (ТС - точка скрещивания). Пример одноточечного скрещивания представлена на рисунке 2.

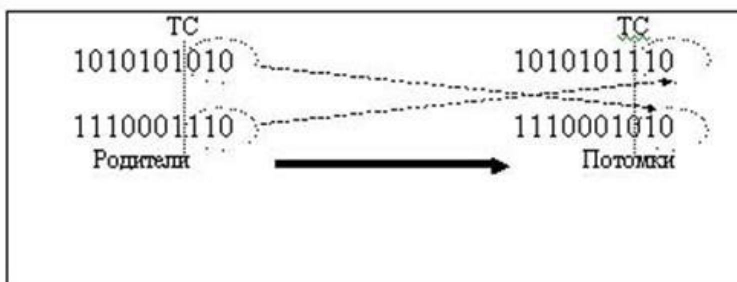


Рис. 2. **Пример одноточечного скрещивания**

При *двухточечном скрещивании* хромосому можно рассматривать как кольцо со связанными первым и последним генами. Кольцо рассекается на две части и полученные части обмениваются. Графическое представление двухточечного скрещивания представлено на рисунке 3, пример на рисунке 4.

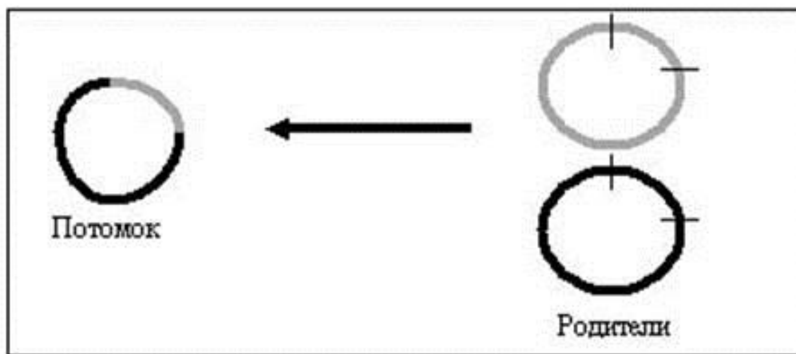


Рис. 3. **Двухточечное скрещивание**

Или

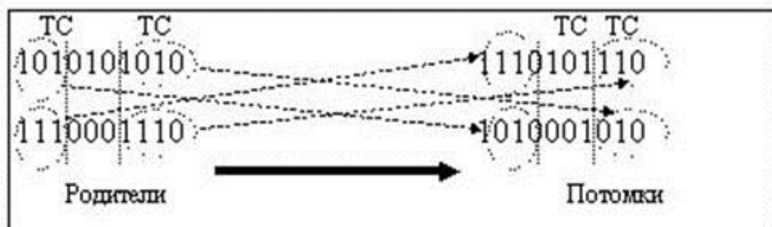


Рис. 4. Пример двухточечного скрещивания

Равномерное скрещивание предполагает, что каждый ген потомка выбирается случайным образом из соответствующих генов родителей.

Замечание: в этом случае родителей может быть больше двух, в том числе возможно участие всей популяции родителей в целом (*gene pool recombination*) .

Мутация в ГА

Мутация состоит из выполнения (обычно небольших) изменений в значениях одного или нескольких генов в хромосоме. Мутация обеспечивает исследование пространства поиска.

В двоичных хромосомах мутация состоит в инвертировании случайным образом выбранного бита генотипа, например 1010 1000.

В ГА мутация является методом восстановления потерянной генетической информации, а не методом поиска лучшего решения.

В ГА мутация применяется к генам с очень низкой вероятностью p_m [0.001, 0.01]. Хорошим эмпирическим правилом считается выбор вероятности мутации из соотношения $p_m =$, где H - число бит в хромосоме. На основе этого правила можно произвести классификацию мутации таким образом:

1. Слабая ($p_m <$),

2. Средняя ($p_m =$)

3. Сильная ($p_m >$)

Обобщенная пошаговая структура ГА

1. Сгенерировать случайным образом начальную популяцию.

2. Оценить полученную популяцию.

3. Генерировать популяцию потомков.

Селекция (выбор двух индивидов из текущей популяции).

Рекомбинация (скрещивание выбранных индивидов).

Мутация (генетическое изменение полученного потомка).

4. Если не все поколения пройдены, то перейти на шаг 2, иначе выдать наилучшего найденного индивида и его значение целевой функции в качестве решения оптимизации.

Схема генетического алгоритма представлена на рисунке 5.

Рис. 5. **Схема ГА**

6.6. Примеры генетического алгоритма

Приводится разъяснение условий установки опций для генетического алгоритма.

Для того, что бы получить наилучшие результаты, как правило, обычно проводят расчеты с различными значениями опций. Выбор наилучшего вида значений опций основан на методе проб и ошибок. В данном разделе приводятся определенные приемы выбора опций с целью улучшения полученных результатов. Полное описание опций можно найти в разделе Опции генетического алгоритма.

6.6.1. Масштаб пригодности

Операция с масштабом пригодности определенным способом стягивает, при помощи функции пригодности, множества необработанных значений пригодности в некий диапазон, который

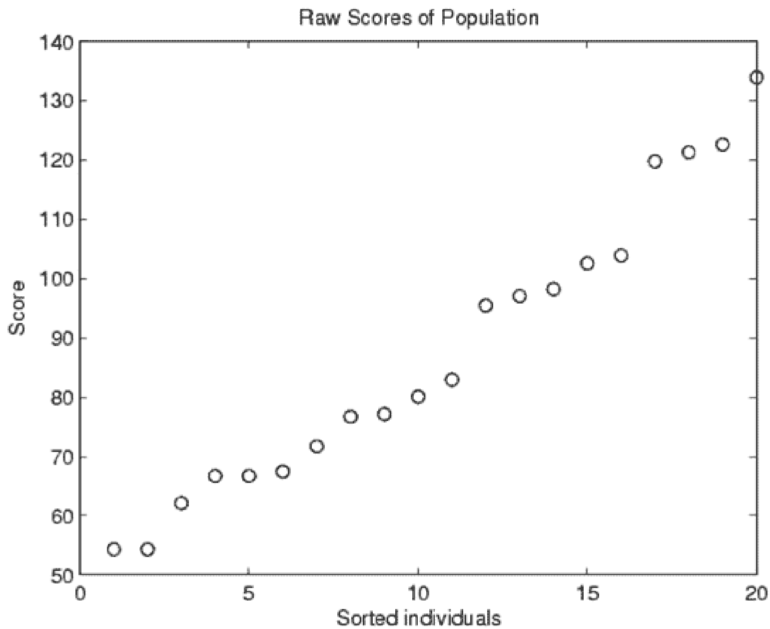
является приемлемым для действий функции отбора. Функция отбора использует отмасштабированные значения пригодности для отбора родительских значений для следующего поколения. Функция отбора назначает более высокую вероятность выбора тем индивидуализированным объектам, которые имеют более высокие отмасштабированные значения.

Величина диапазона отмасштабированных значений оказывает влияние на эффективность работы Генетического алгоритма. Если отмасштабированные значения изменятся в широком диапазоне, то индивидуализированные объекты с наибольшими значениями отмасштабированных значений воспроизводятся достаточно быстро и также быстро формируется генетическая совокупность наследственных факторов данной популяции. Такая ситуация предотвращает генетический алгоритм от поиска по другим областям пространства решений. С другой стороны, если отмасштабированные значения изменяются достаточно мало, то все индивидуализированные объекты имеют примерно одну и ту же вероятность участия в воспроизводстве и поиске решения, что заметно замедляет процесс решения.

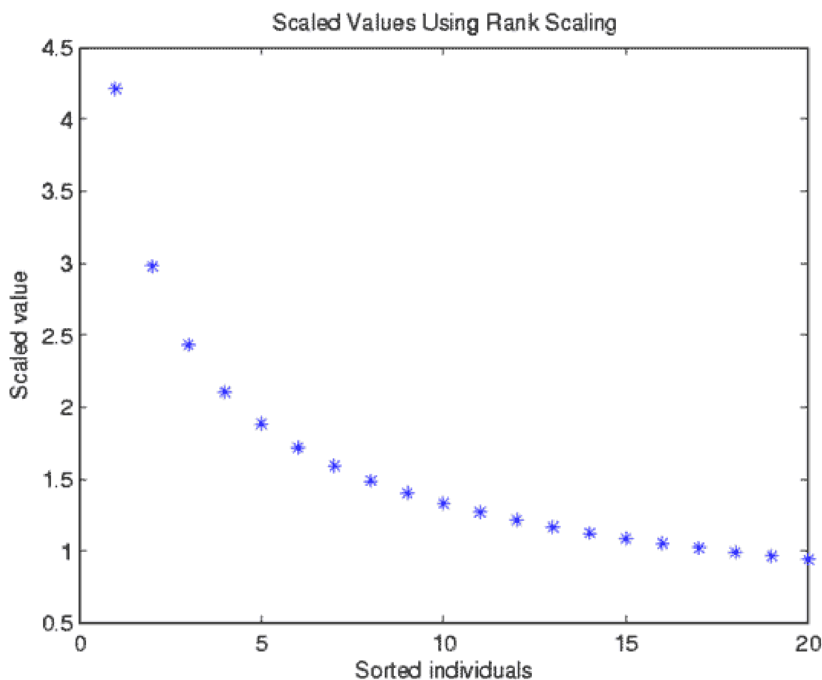
Принимаемая по умолчанию функция масштабирования значений пригодности, Rank, масштабирует необработанные множества на основе определенного ранга для каждого индивидуализированного объекта вместо значений этого множества. Рангом индивидуализированного объекта является его позиция в отсортированном множестве: ранг наибольшей пригодности индивидуализированного объекта равен 1, ранг следующей наибольшей пригодности равен 2 и так далее. Функция масштабирования ранга приписывает отмасштабированные значения таким образом, что бы:

1. Отмасштабированное значение индивидуализированного объекта с рангом n прямо пропорционально $1/(\sqrt{n})$.
2. Сумма отмасштабированных значений по всему пространству семейств равна числу родителей, необходимых для создания следующего поколения.

Далее на графике представлены необработанные множества типичного семейства из 20 индивидуализированных объектов, отсортированных по порядку увеличения.



Следующий график представляет отмасштабированные значения необработанных множеств согласно рангу масштабирования.



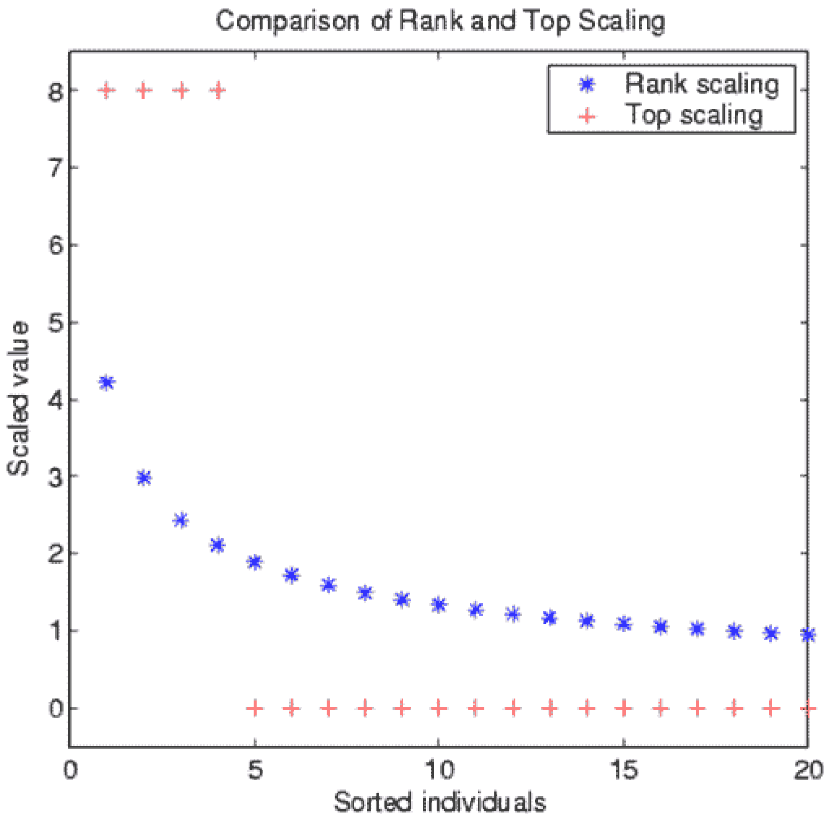
Поскольку в данном алгоритме минимизируется функция пригодности, то менее обработанные значения множеств имеют более высокие отмасштабированные значения. А так же, поскольку операция вычисления ранга придает значения, которые зависят только от ранга индивидуализированного объекта, то отмасштабированные значения, как это показано, есть те же самые как для любого семейства размера 20, так и для числа родителей равного 32.

6.6.2. Сопоставление ранга и Масштабирования высшего уровня.

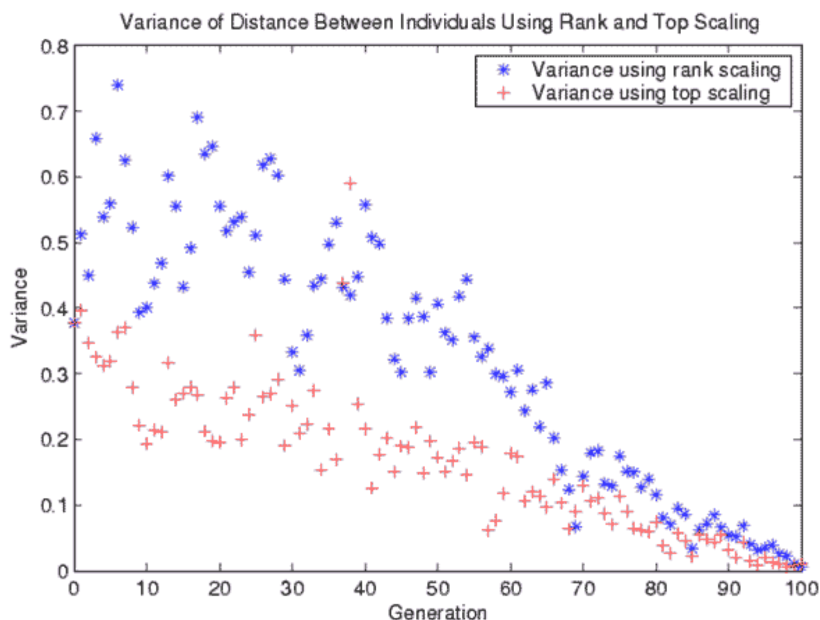
Для того, что бы оценить эффект от масштабирования, необходимо сравнить результаты работы Генетического алгоритма с включенным рангом масштабирования с действием одной из функций масштабирования, такой как Масштабирования высшего уровня. По умолчанию операция Масштабирования высшего уровня назначает четырем наиболее подходящим индивидуализированным объектам

одни и те же отмасштабированные значения, равные числу родителей и деленному на четыре, а остальным объектам назначается ноль. При использовании данной функции отбора только четыре наиболее подходящих индивидуализированных объекта могут быть выбраны в качестве родительских.

Далее на рисунке приведено сравнение отмасштабированных значений семейства размером 20 числом родителей 32 при использовании ранга и Масштабирования высшего уровня.



Поскольку в методе Масштабирования высшего уровня число родителей ограничено индивидуальными объектами лучшей пригодности, то в данном случае задействовано меньшее многообразие семейств, чем в случае масштабирования по рангу. На следующем рисунке приведено сравнение расстояний между индивидуализированными объектами для каждой генерации в случае Масштабирования высшего уровня и масштабирования по рангу.



6.6.3. Селекция

Операция селекции предназначена для выбора родителей следующего поколения на основе отмасштабированных величин полученных после использования функция масштабирования значений пригодности. Каждый индивидуализированный объект может быть избран в качестве родительского два и более раз, в этом случае он привносит свои гены в два и более дочерних параметра. Принимаемая по умолчанию функция Stochastic uniform составляет некую линию, на которой каждый родитель соответствует некому отрезку этой линии с длиной,

пропорциональной его отмасштабированному значению. Таким образом, алгоритм продвигается вдоль этой линии с шагами одного и того же размера. При этом, на каждом шаге алгоритм назначает родителя в соответствии с отрезком его расположения.

Более детерминированной функцией отбора является функция Remainder, которая выполняется в два этапа:

- На первом этапе данная функция отбирает родителей детерминированным способом согласно целой части отмасштабированного значения для каждого индивидуализированного объекта. Например, если отмасштабированное значения индивидуализированного объекта равно 2,3, то функция отбора принимает этот индивидуализированный объект в качестве родительского дважды.
- На втором этапе данная функция отбора подбирает дополнительных родителей на основе использования уже дробной части отмасштабированных значений в виде стохастического однородного отбора. Данная функция выделяет линию на отрезках, чьи длины являются пропорциональными дробной части отмасштабированного значения индивидуализированного объекта, и для выбора родительских значений движется вдоль этой линии с единым размером шага.
- Отметим, что если все дробные части отмасштабированных значений равны нулю, как это имеет место в методе Масштабирования высшего уровня, то данная операция отбора полностью детерминированной.

6.6.4. Опции репродуцирования.

Опции репродуцирования определяют способ формирования последующего поколения в Генетическом алгоритме. Имеются следующие опции:

- **Elite count** – элитный номер, это число индивидуализированных объектов с наилучшей функцией пригодности для текущего поколения, которые гарантировано останутся неизменными для следующего поколения. Эти

индивидуализированные объекты называются элитными дочерними объектами. Принимаемое по умолчанию значение **Elite count** равно 2.

- В случае, когда значение величины Elite count равно 1, то наилучшее значение функции пригодности может только уменьшаться при переходе от одного поколения к следующему. Это как раз наиболее желательный случай, поскольку в Генетическом алгоритме происходит минимизация функции пригодности. Установка опции Elite count в виде большого числа заставляет наиболее приспособленные индивидуализированные объекты доминировать в семействе, что может снижать эффективность операции поиска.
- Crossover fraction – доля индивидуализированных объектов в следующем поколении, отличном от элитного дочернего, и которое формируется в операции Кроссовера (перекрещивания). В разделе Установка кроссоверной доли приводится описание как значение опции Crossover fraction оказывает влияние на эффективность работы Генетического алгоритма.

6.6.5. Мутация и кроссовер

Индивидуализированные объекты используются в текущем поколении Генетического алгоритма в качестве дочерних объектов как основа для создания последующих поколений. Кроме элитных дочерних объектов, которые соответствуют индивидуализированным объектам текущего поколения с наилучшими значениями пригодности, данный алгоритм формирует:

- Кроссоверные дочерние объекты путем отбора векторных элементов или генов из пары индивидуализированных объектов текущего поколения с их последующей комбинацией формированием дочерних объектов.
- Мутационные дочерние объекты на основе использования стохастических операций для отдельно взятых индивидуальных объектах из текущего поколения и последующего формирования уже самих дочерних объектов.

Оба этих процесса представляют собой сущность Генетического алгоритма. Операция кроссовера позволяет данному алгоритму выделять наилучшие гены из индивидуумов различного типа и затем производить их рекомбинацию в дочерние объекты потенциально

высшего качества. Мутация служит дополнением к диверсификации семейства и, таким образом, увеличивает вероятность, что в алгоритме сгенерируются индивидуальные объекты с наилучшими значениями пригодности. Без операции мутации в алгоритме могут воспроизводиться только индивидуализированные объекты с генами, являющимися простой комбинацией начального семейства.

Смотри раздел Создание следующего поколения как пример использования операций мутации и кроссовера в Генетическом алгоритме.

В алгоритме имеется возможность установить следующие опции для задания числа дочерних объектов каждого типа:

- Значение **Elite count** в опции **Reproduction** устанавливает число элитных дочерних объектов.
- Значение **Crossover fraction** в опции **Reproduction** устанавливает долю семейства, отличного от элитных дочерних объектов, которая является уже кроссоверными дочерними объектами.

Например, если значение величины **Population size** равно 20, значение величины **Elite count** равно 2 и значение величины **Crossover fraction** равно 0.8 то число дочерних объектов каждого типа будет следующим:

- Число дочерних объектов 2.
- Число объектов, отличных от дочерних будет 18 так, что алгоритм после операции округления 0.8*18 = 14.4 до 14 дает число кроссоверных детей.
- Оставшиеся 4 индивидуализированных объекта, отличные от элитных дочерних объектов, составляют дочерние мутационные объекты.

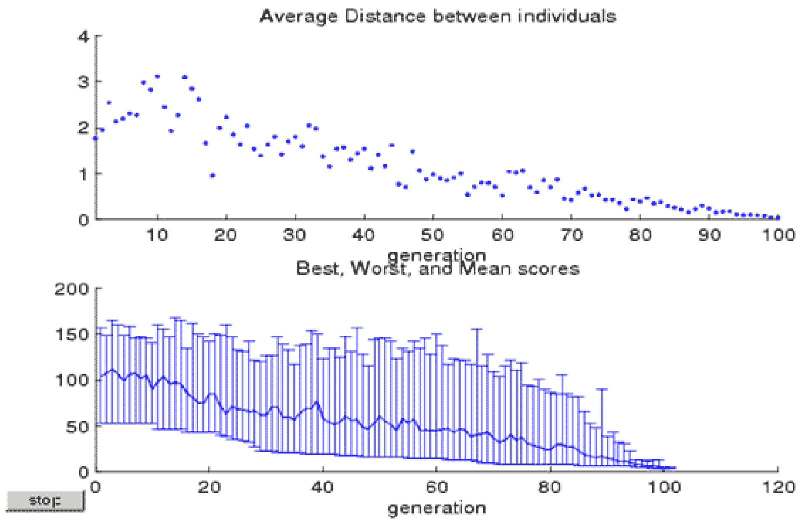
6.6.6. Установка числа мутаций

Операция мутации осуществляется в Генетическом алгоритме на основе использования специальных установок поля **Mutation function**. Используемая по умолчанию функция Гауссовской мутации добавляет случайно выбранное с помощью распределения Гаусса число, *mutation*,

к каждому элементу родительского вектора. Как правило, это число мутаций, которое пропорционально стандартному среднеквадратичному отклонению от распределения, уменьшается с каждой последующей итерацией. В алгоритме путем установки опций **Scale** и **Shrink** имеется возможность управлять усредненным числом мутаций для каждой итерации:

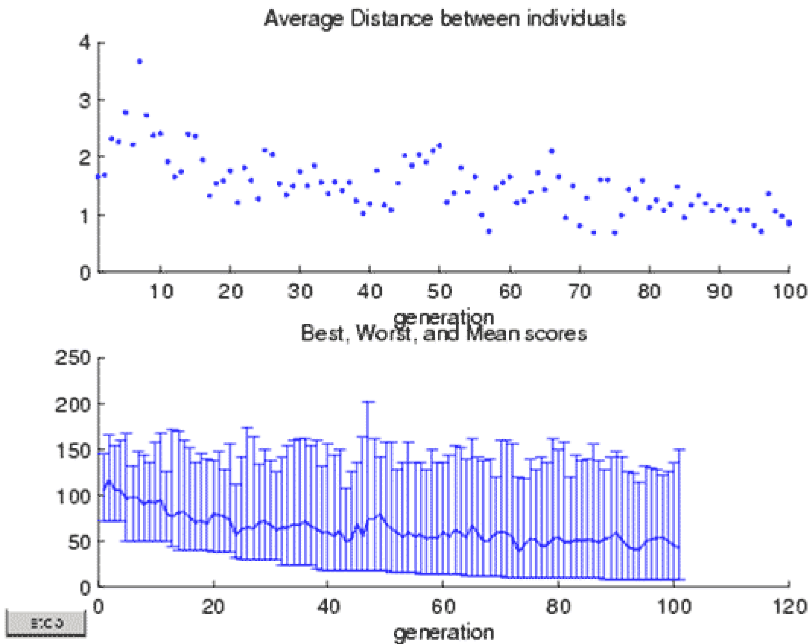
- Опция **Scale** управляет стандартным среднеквадратичным отклонением на первой итерации, которое равно параметру **Scale**, умноженному на ранг исходного семейства, задаваемого с помощью опции **Initial rang**е.
- Опция **Shrink** управляет скоростью уменьшения среднего числа мутаций. Стандартное среднеквадратичное отклонение линейно уменьшается до тех пор, пока его конечное значение будет равно 1 – **Shrink** от его начального значения для первого поколения. Например, если величина **Shrink** по умолчанию принимается равной 1, то тогда число мутаций уменьшается к конечному шагу до нуля.

Имеется возможность отследить влияние операции мутации с помощью установок для графических функций **Distance и Range**, с последующим выполнением Генетического алгоритма применительно к задаче, описанной в разделе Пример функции Растригана. Эта зависимость далее представлена на рисунке.



На верхнем графике приведено среднее расстояние между точками для каждого поколения. По мере выполнения алгоритма происходит уменьшение числа мутаций, что по своей сути представляет собой среднее расстояние между индивидуализированными объектами, примерно до нуля для конечной генерации. На нижнем графике представлена вертикальная линия для каждого поколения, отображающая диапазон изменения от наименьшего до наибольшего значения пригодности, а также среднего значения пригодности. По мере уменьшения числа мутаций уменьшается и данный диапазон. Эта зависимость так же показывает, что уменьшение числа мутации приводит к снижению диверсификации последующих поколений.

В качестве сравнения далее представлены графические зависимости для **Distance** и **Range** при установке параметра **Shrink** в 0.5.



При установке параметра **Shrink** в 0,5, среднее число мутаций уменьшается с коэффициентом ½ до достижения конечной итерации. Соответственно, среднее расстояние между индивидуализированными объектами уменьшается примерно с тем же коэффициентом.

6.6.7. Установка кроссоверной доли

Поле **Crossover fraction** опции **Reproduction** устанавливает долю объектов каждой семейства, в отличие от элитных дочерних объектов, которые составляют в итоге кроссоверные дочерние объекты. Кроссоверная доля равная 1 означает,что все дочерние объекты, в отличие от элитных индивидуумов, являются кроссоверными дочерними объектами, при этом кроссоверная доля равная 0 означает, что все дочерние объекты дочерними мутационными объектами. Приведенный далее пример показывает, что ни один из рассмотренных здесь экстремумов не представляет собой эффективную стратегию для оптимизации некой функции.

В примере используется функция пригодности, чьи значения в произвольной точке равны сумме абсолютных значений координат этих точек. То есть,

- $f(x_1,x_2,...,x_n)=|x_1|+|x_2|+...+|x_n|$

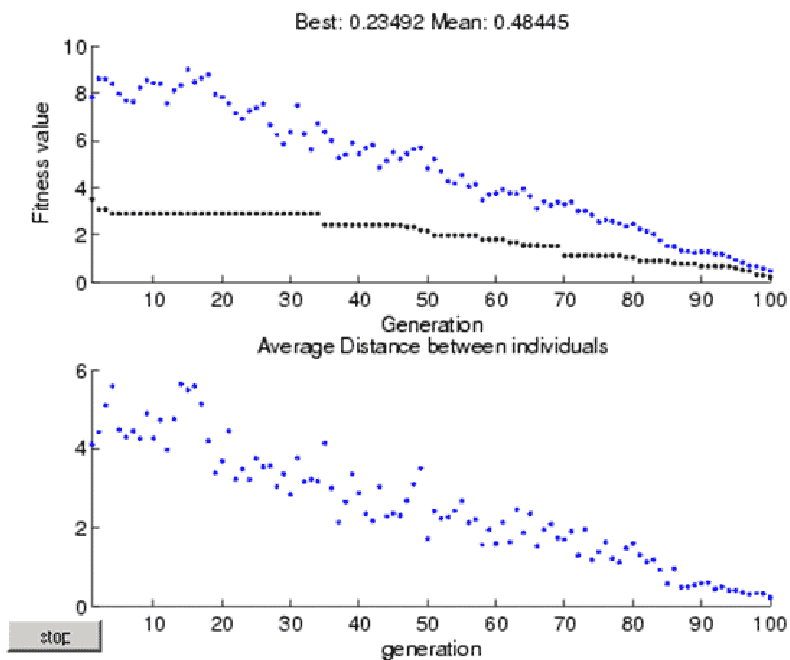
Можно эту функцию задать и в виде анонимной функции, если в поле **Fitness function** внести следующую запись

```
@(x)sum(abs(x))
```

Для выполнения данного примера следует:

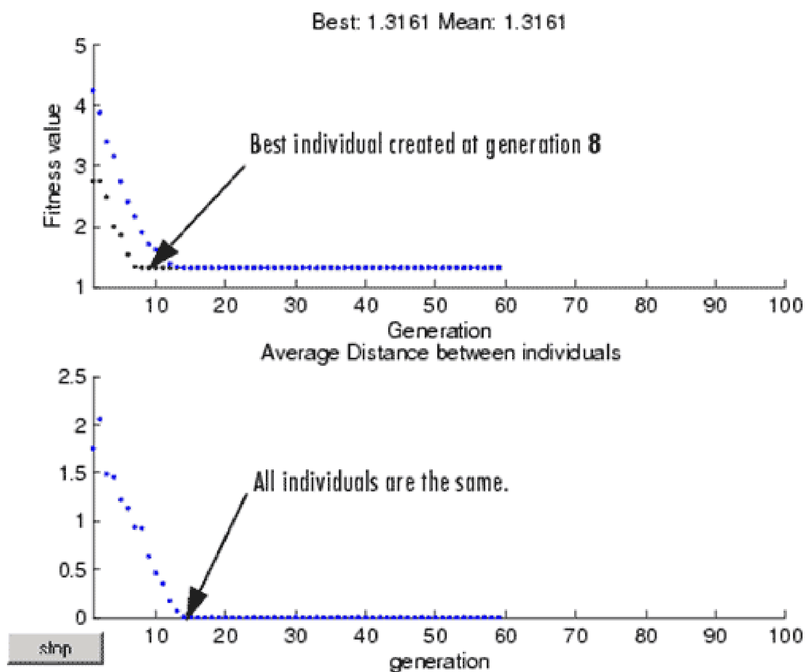
- В поле **Fitness function** внести @(x)sum(abs(x)).
- В поле **Number of variables** внести 10.
- В поле **Initial range** внести [-1;1].
- Выбрать параметры **Best fitness** и **Distance** на панели **Plots**.

В начале выполним пример с принимаемым по умолчанию значением 0,8 параметра **Crossover fraction**. После выполнения примера получим наилучшее значение функции пригодности примерно равное 0,2 и следующие типы графических зависимостей.



6.6.8. Установка без Мутаций

Для того, что проконтролировать работу Генетического алгоритма без включения операции мутации, следует установить параметр **Crossover fraction** в 1.0 и кликнуть мышкой на кнопку **Start**. В результате выполнения такой команды получим наилучшее значение функции пригодности примерно равное 1,3 и следующие типы графических зависимостей.

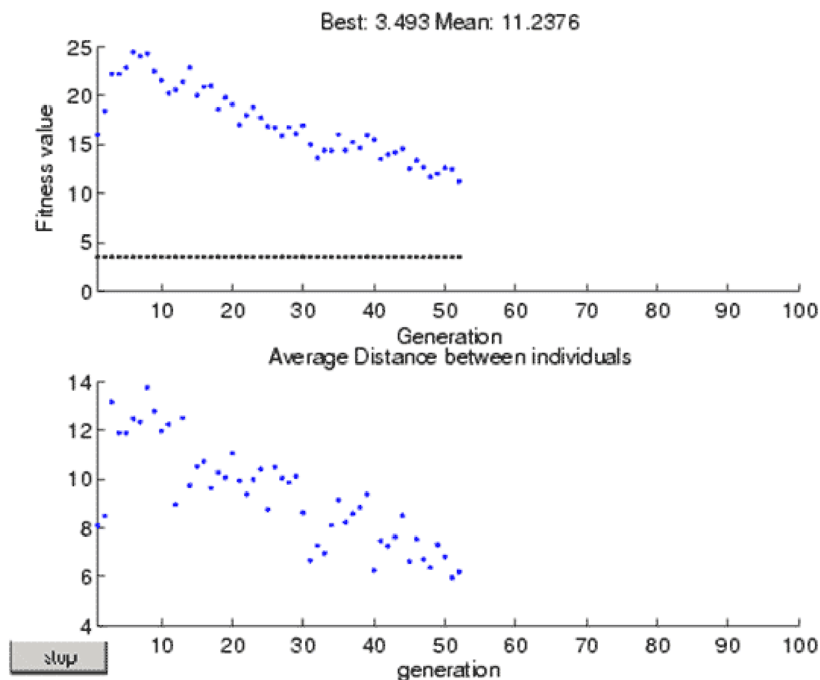


В данном случае в алгоритме производится отбор геномов из индивидуумов для начального семейства с последующей их рекомбинацией. Поскольку нет режима мутации, то алгоритм не может воспроизводить какие-либо новые геномы. В алгоритме генерируются наилучшие индивидуализированные объекты на основе возможного использования геномов с номером поколения, равным 8, когда график наилучшей пригодности достигает определенного уровня. После этого, формируются новые копии наилучших индивидуализированных объектов, которые далее выбираются для формирования новых поколений. К поколению с номером 17 уже все индивидуализированные объекты для данного семейства являются одними и теми же, а именно, наилучшими индивидуализированными объектами. В этом случае среднее расстояние между индивидуализированными объектами будет равно 0. Поскольку уже после поколения с номером 8 в данном алгоритме не может быть улучшено значение наилучшей пригодности, то алгоритм выходит на

останов после 50 поколений, поскольку параметр **Stall generations** равен 50.

6.6.9. Мутации без кроссовера

Для того, что проконтролировать работу Генетического алгоритма без включения операции кроссовера, следует установить параметр **Crossover fraction** в 0 и кликнуть мышкой на кнопку **Start**. В результате выполнения такой команды получим наилучшее значение функции пригодности примерно равное 3,5 и следующие типы графических зависимостей.



В данном случае в результате стохастических операций в алгоритме не возникают какие-либо улучшения значений пригодности для наилучших индивидуализированных объектов первого поколения. Поскольку происходит улучшение геномов индивидуумов и для других

индивидуализированных объектов, как это можно отследить из верхней графической зависимости по уменьшению среднего значения функции пригодности, то эти улучшенные геномы не участвуют в комбинациях с геномами наилучших индивидуализированных объектов вследствие отсутствия операции кроссовера. Как результат всего этого, график наилучшей пригодности выходит на некий уровень и алгоритм останавливается при числе генераций равном 50.

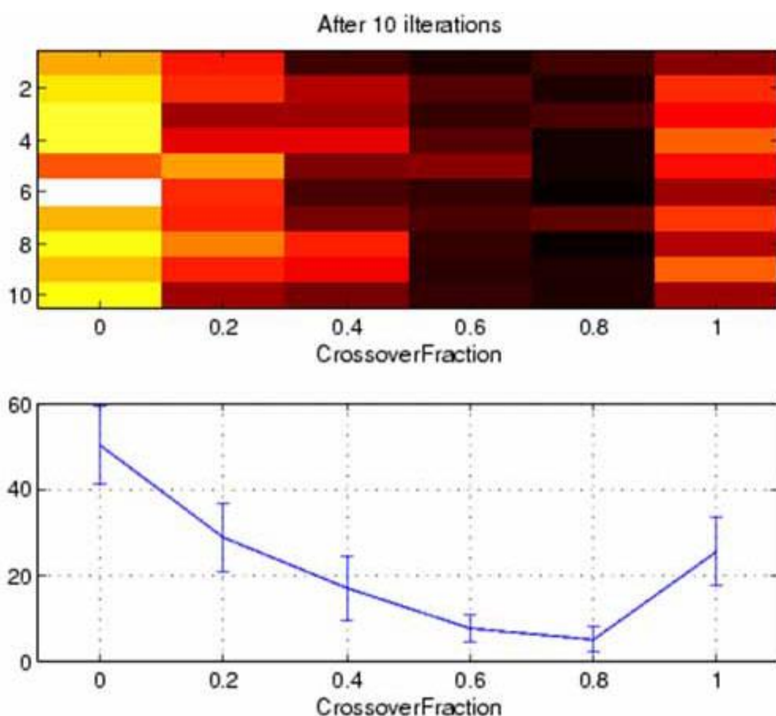
6.6.10. Сравнение результатов с фракциями измененных после операции кроссовера

С помощью включенной в данный тулбокс демонстрационной программы deterministicstudy.m имеется возможность провести сравнение результатов использования Генетического алгоритма для оптимизации функции Растригина с установками параметра Crossover fraction в 0, .2, .4, .6, .8 и 1. Демонстрационная программа выполняется для 10 поколений. Для каждого поколения в программе воспроизводится графическая зависимость средних и стандартных отклонений для значений наилучшей пригодности для всех предшествующих поколений и для каждого значения параметра Crossover fraction.

Для выполнения демонстрационной программы следует ввести команду

```
deterministicstudy
```

в командном окне MATLAB. По окончании демонстрационной программы отобразятся следующие графические зависимости.



На нижнем графике приведены средние величины и стандартные среднеквадратичные отклонения значений функции пригодности для 10 поколений применительно для каждого из значений кроссоверной функции. На верхнем графике представлена цветовая кодировка для значений наилучшей пригодности в каждом поколении.

Для выбранного вида функции пригодности установка параметра Crossover fraction в 0,8 дает наилучшие результаты. Тем не менее, для других видов функции пригодности установка различные значений параметра Crossover fraction может дать более лучшие результаты.

6.6.11. Пример – сравнение глобального и локального минимумов

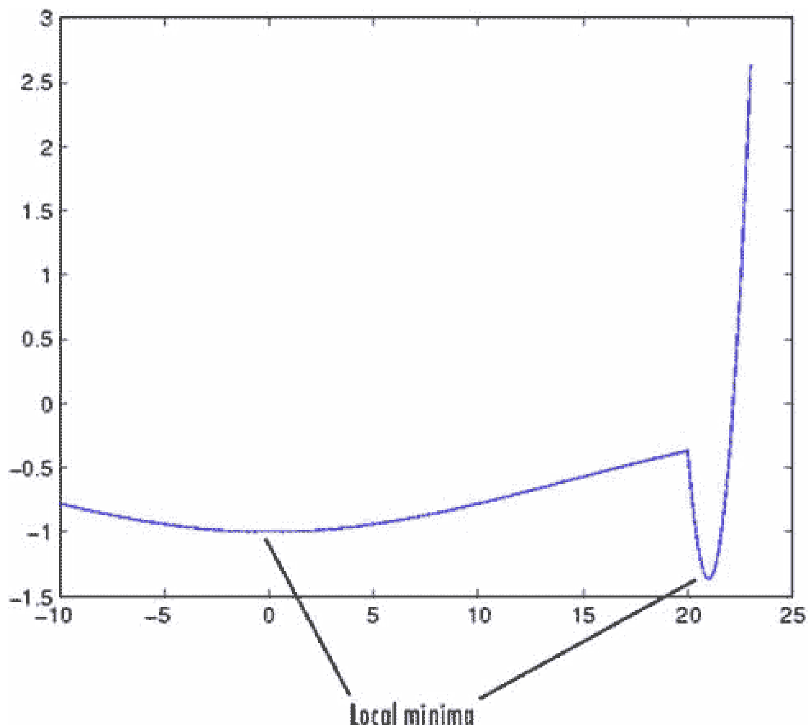
Иногда целью оптимизации является поиск глобального максимального или минимального значения функции, т.е. точки, где

рассматриваемая функция является больше или меньше значения для любой другой точки в пространстве поиска. Однако, оптимизационные алгоритмы иногда возвращают значения в точку локального минимума, т.е. точку, где значение функции меньше, чем значения функции для близлежащих точек, но не исключена возможность существования удаленных точек пространства поиска с еще более меньшими значениями функций. При правильной установке соответствующих опций генетический алгоритм может преодолеть указанный недостаток.

В качестве примера рассмотрим следующую функцию:

$$f(x) = \begin{cases} -\exp\left(-\left(\frac{x}{20}\right)^2\right) & \text{for } x \le 20 \\ -\exp(-1) + (x-20)(x-22) & \text{for } x > 20 \end{cases}$$

Далее на рисунке представлен график данной функции



Функция имеет два локальных минимум, один в точке x = 0, где значение функции равно - 1, и другой в точке = 21, где значение функции равно 1 - 1/е. Поскольку последнее значение является более меньшим, то глобальный минимум находится в точке = 21.

6.6.12. Пример решения данной задачи с помощью Генетического алгоритма

Для выполнения данного примера с помощью Генетического алгоритма следует выполнить следующие действия:

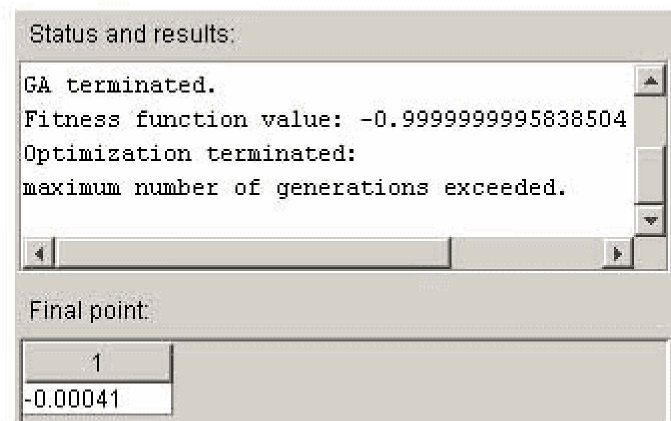
1. Скопировать и записать следующие коды в новый М-файл редактора MATLAB

```
2.      function y = two_min(x)
```

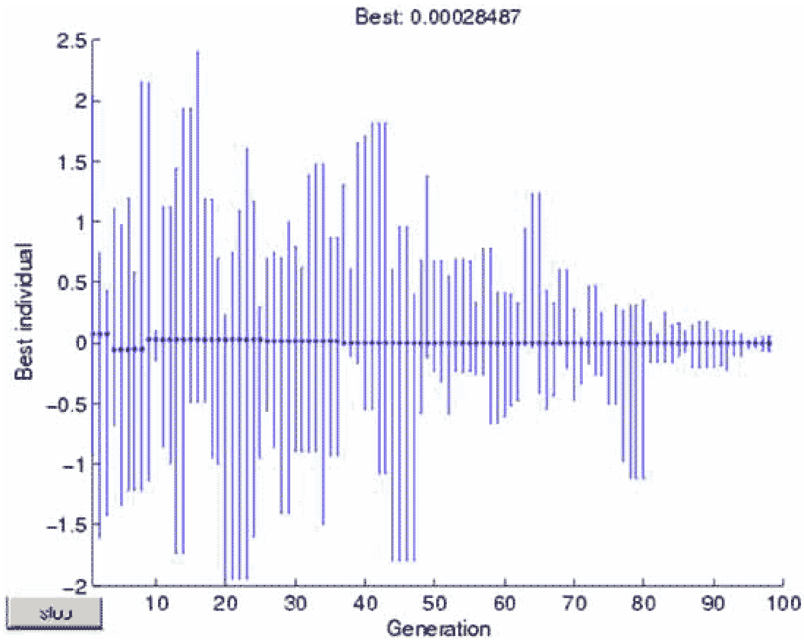
```
3.       if x<20
4.           y = -exp(-(x/20).^2);
5.       else
6.           y = -exp(-1)+(x-20)*(x-22);
7.       end
```
8. Записать файл под именем two_min.m in в рабочую директорию пространства MATLAB.
9. В инструментарии генетического алгоритма выполнить следующие действия.

- Установить **Fitness function** в @two_min.
- Установить **Number of variables** в 1
- Кликнуть кнопку Start.

Генетический алгоритм возвратит точку очень близкую к локальному минимуму в точке x = 0.



Далее на следующем специальном графике приведено отображение процесса как алгоритм скорее всего обнаруживает локальный минимум, а не глобальный. На графике представлены диапазон разброса индивидуумов для каждого поколения и их наилучшие значения.



Отметим, что все индивидуализированные объекты находятся в пределе от -2до 2.5. Поскольку этот диапазон больше принимаемого по умолчанию параметра **Initial range** [0;1], то вследствие принятого типа функции мутационных процессов, он не является достаточно большим, что бы обрабатывать точки вблизи глобального минимума в точке x = 21.

Один из способов заставить Генетический алгоритм обрабатывать точки из более широкого диапазона, т.е. расширить диверсификацию семейств, заключается в увеличении параметра **Initial range**. По своей сути параметр **Initial range** не должен включать в себя точку x = 21, но он должен быть достаточно большим, так что бы данный алгоритм генерировал индивидуализированные объекты вблизи 21. Установим параметр **Initial range** в [0;15], как это представлено на рисунке далее.

Population

Population type: Double Vector

Population size: 20

Creation function: Uniform

Initial population: []

Initial scores: []

Initial range: [0 ; 15]

Set Initial range to [0; 15].

Затем кликнем мышкой кнопку Start. Генетический алгоритм возвратит точку очень близкую к 21.

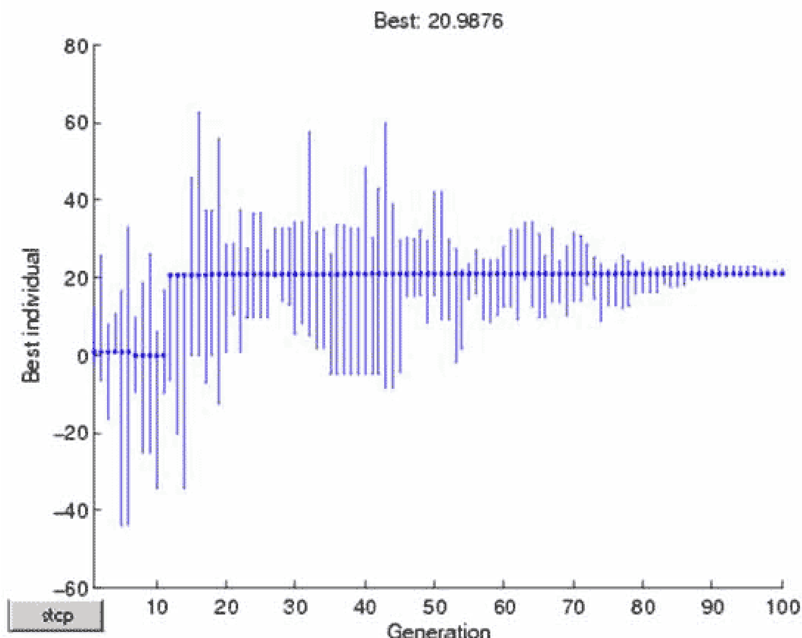
Status and results:

```
GA running.
GA terminated.
Fitness function value: -1.36772525220865
Optimization terminated:
maximum number of generations exceeded.
```

Final point:

1
20.98758

К данному моменту времени на специализированном графике можно видеть много больший диапазон индивидуализированных объектов. Уже для второго поколения индивидуализированные объекты уже охватывает точку со значением 21 и к 12 поколению алгоритм определяет наилучший индивидуализированный объект, который примерно равен 21.



6.6.13. Использование гибридной функции

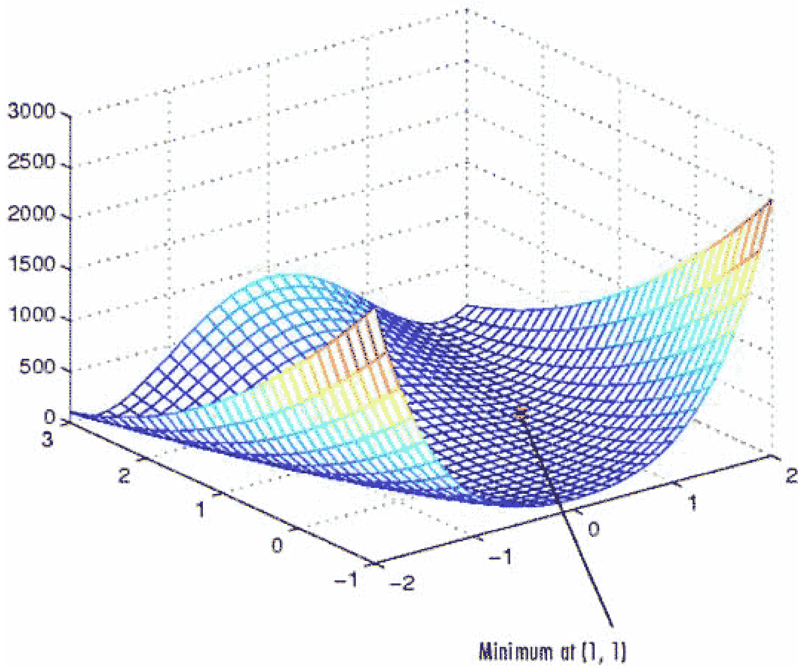
Гибридной функцией является некая оптимизационная функция, которая выполняется по окончанию работы Генетического алгоритма и предназначена для улучшения значений функции пригодности. В качестве исходной точки в гибридной функции используется конечная точка Генетического алгоритма. Определить гибридную функцию можно с помощью опции **Hybrid function**.

Приведенный нише пример основан на использовании функции fminunc, или функция минимизации без ограничений из состава тулбокса Оптимизация. В данном примере в первую очередь для поиска решения наиболее близкого к оптимальной точке выполняется Генетический алгоритм, а затем найденная точка используется уже в качестве исходной для функции fminunc.

В качестве примера выбран поиск минимума функции Розенброка, определяемой как

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

Далее на рисунке представлен график функции Розенброка.



Для расчета данной функции в тулбоксе имеется М-файл dejong2fcn.m. С целью демонстрации примера выполним команду

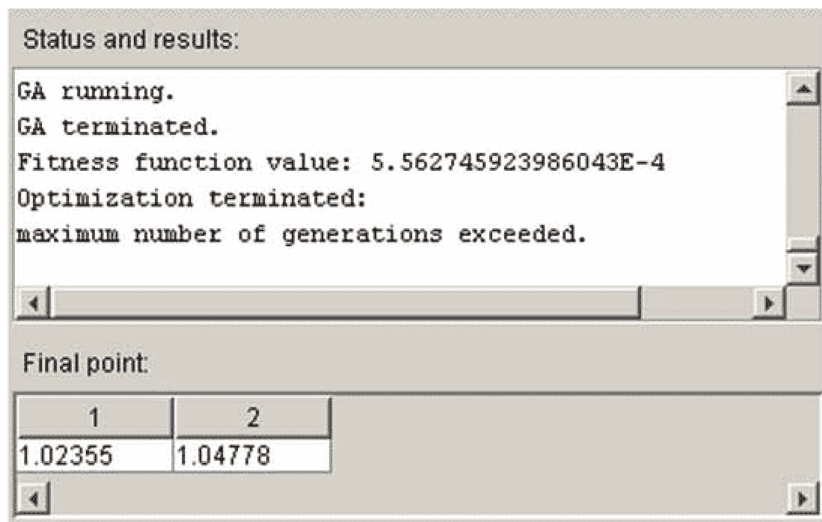
```
hybriddemo
```

в окне команд пакета MATLAB.

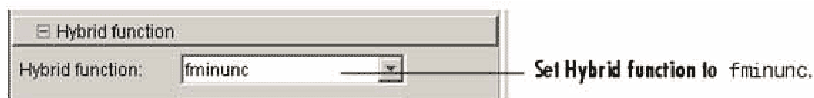
С начала, для запуска данного примера откроем инструментарий Генетического алгоритма с помощью команды gatool и введем следующие установки:

- установим в поле Fitness function @dejong2fcn.
- установим в поле t Number of variables 2.
- установим в поле Population size 10.

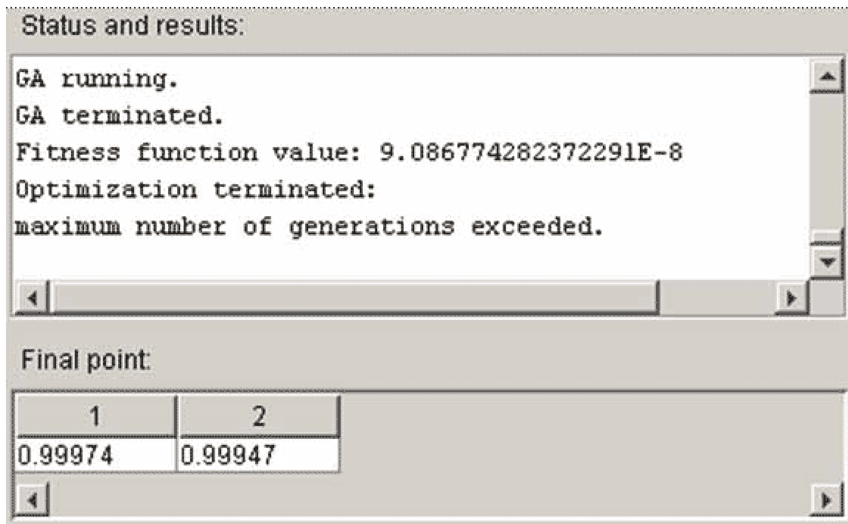
Перед введением гибридной функции постараемся выполнить Генетический алгоритм сам по себе при помощи кнопки Start. После выполнения Генетического алгоритма на панели Status and results можно будет увидеть следующие результаты.



Конечная точка расчетов близка к точке истинного минимума (1, 1). Имеется возможность улучшить результаты расчета путем установки в опции Hybrid function требуемой гибридной функции fminunc.



По окончании выполнения Генетического алгоритма функция fminunc воспринимает конечную точку Генетического алгоритма в качестве исходной и возвращает более точный результат, как это представлено далее на панели **Status and results**.



6.6.14. Установка максимального числа поколений

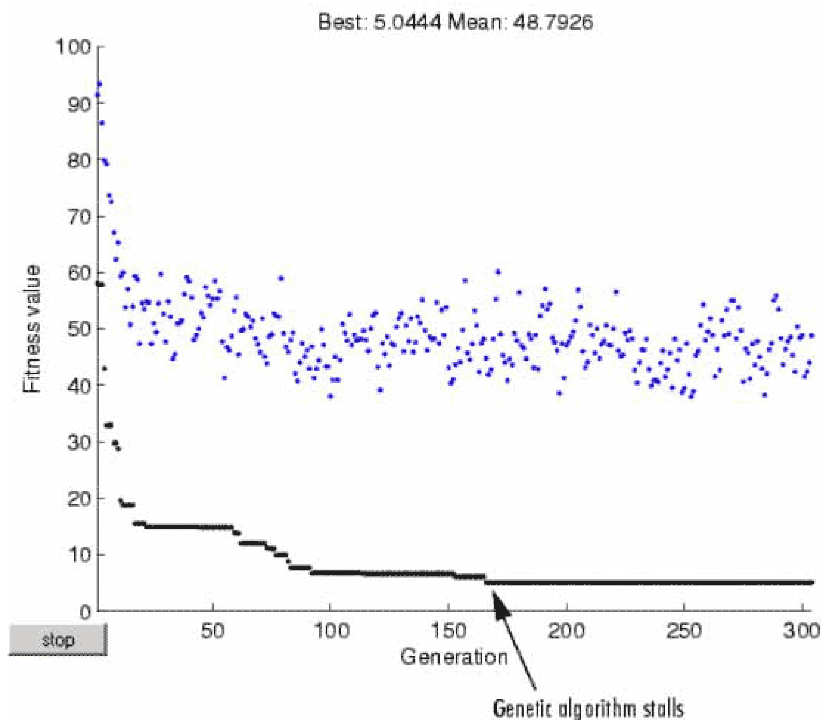
Опция **Generations** в разделе **Stopping criteria** определяет максимальное число итераций при выполнении функции Генетического алгоритма, более подробно смотри раздел Условия останова алгоритма. Увеличение параметра **Generations**, как правило, приводит к улучшению конечного результата.

В качестве примера изменим установки инструментария Генетического алгоритма следующим образом:

- установим в поле **Fitness function** @rastriginsfcn.
- установим в поле **Number of variables** 10.
- выберемт Best **fitnes**s на панели **Plots**.
- установим в поле **Generations** Inf.

- установим в поле **Stall generations** Inf.
- установим в поле *Stall time* Inf.

Затем выполним Генетический алгоритм примерно на 300 итераций и кликнем на кнопку **Stop**. Далее на рисунке представлена зависимость для итогового значения параметра **Best fitness** после 300 итераций.



Отметим, что алгоритм практически прекращается примерно после выполнения примерно 170 поколений – то есть, нет заметного улучшения в функции пригодности после 170 поколений. Если восстановить параметр **Stall generations** в его принимаемое по умолчанию значение равное 50, то алгоритм остановится на числе поколений примерно равном 230. Если происходит постоянный останов Генетического алгоритма с текущим значением параметра **Generations**, то имеется возможность улучшить итоговые результаты

путем увеличения обоих параметров **Generations** и **Stall generations**. Хотя, конечно, не исключена возможность более эффективного улучшения работы алгоритма с помощью изменения других опций.

Примечание. Если параметр **Mutation function** установить как Gaussian, то увеличение значения параметра **Generations** действительно может привести к ухудшению конечного результата. Это обстоятельство можно объяснить тем, что Гауссова функция мутации снижает среднее количество мутаций для каждого поколения на некий коэффициент, величина которого зависит от параметра **Generations**. Следовательно, установка параметра **Generations** оказывает влияние на эффективность работы алгоритма.

6.6.15. Векторизация функции пригодности

Генетический алгоритм может выполняться значительно быстрее, если провести операцию *векторизации* функции пригодности. Это означает, что генетический алгоритм обращается к функции пригодности только один раз. Одновременно предполагается, что функция пригодности подходит для всех индивидуализированных объектов текущего поколения. Для векторизации функции при годности необходимо выполнить следующее.

Записать М-файл расчета искомой функции таким образом, что бы он был представлен в виде матрицы с произвольным числом строк, которые соответствовали бы индивидуализированным объектам данного поколения. Например, для векторизации функции.

$$f(x_1, x_2) = x_1^2 - 2x_1x_2 + 6x_1 + x_2^2 - 6x_2$$

Запишем М-файл в виде следующих кодов:

```
z =x(:,1).^2 - 2*x(:,1).*x(:,2) + 6*x(:,1) +
x(:,2).^2 - 6*x(:,2);
```

Колонка первого элемента переменной *x* указывает на строчные элементы от *x*, так что *x* (:, 1) есть вектор. Операторы .^ и .* указывают на поэлементные операции с данными векторами.

Установить опцию **Vectorize** как `On`.

Примечание. Для корректного использования опции **Vectorize** необходимо, что бы функция пригодности допускала использованиепроизвольного числа строк.

С помощью следующего сравнения, выполненного из командной строки, можно увидеть эффект увеличения скорости выполнения, достигаемого с помощью установки опции **Vectorize**.

```
tic;ga(@rastriginsfcn,20);toc

elapsed_time =

    4.3660

options=gaoptimset('Vectorize','on');

tic;ga(@rastriginsfcn,20,options);toc

elapsed_time =

    0.5810
```

6.6.16. Минимизация при наличии ограничений с использованием функции ga

Пусть необходимо найти минимум простой функции пригодности с двумя переменными x1 and x2

$$\min_{x} f(x) = 100\,(x_1^2 - x_2)^2 + (1 - x_1)^2$$

При условии выполнения следующих нелинейных неравенств и ограничений

$$
\begin{array}{ll}
x_1 \cdot x_2 + x_1 - x_2 + 1.5 \le 0 & \text{(nonlinear constraint)} \\
10 - x_1 \cdot x_2 \le 0 & \text{(nonlinear constraint)} \\
0 \le x_1 \le 1 & \text{(bound)} \\
0 \le x_2 \le 13 & \text{(bound)}
\end{array}
$$

Начнем с создания функций пригодности и ограничений. В первую очередь, создадим следующий М-файл под именем simple_fitness.m

```
function y = simple_fitness(x)
y = 100 * (x(1)^2 - x(2)) ^2 + (1 - x(1))^2;
```

В функции Генетического алгоритма, ga, предполагается, что функция пригодности имеет только одно обращение к x, где x имеет столько же элементов, что и число переменных решаемой задачи. Функция пригодности вычисляет значение функции и возвращает скалярное значение в виде одного выходного аргумента y.

Далее создадим следующий М-файл под именем simple_fitness.m, который включал бы в себя вводимые в задачу ограничения

```
function [c, ceq] = simple_constraint(x)
c = [1.5 + x(1)*x(2) + x(1) - x(2);
-x(1)*x(2) + 10];
ceq = [];
```

Предполагается, что функция ga имеет функцию ограничений с одним входом x, где x имеет столько же элементов, что и число переменных решаемой задачи. В функции ограничений вычисляются значения всех ограничений типа равенств и неравенств и возвращаются два вектора, ceq и c, соответственно.

Для минимизации функции пригодности, необходимо передать указатель функции в функцию пригодности на месте первого аргумент функции ga, а так же как определить число переменных на месте

второго аргумента. Нижние и верхние границы задаются на месте параметров LB и UB соответственно. Кроме того, также необходимо передать указатель функции и в функцию нелинейных ограничений.

```
ObjectiveFunction = @simple_fitness;
nvars = 2;    % Number of variables
LB = [0 0];   % Lower bound
UB = [1 13];  % Upper bound
ConstraintFunction = @simple_constraint;
[x,fval] =
ga(ObjectiveFunction,nvars,[],[],[],[],LB,UB,Constr
aintFunction)
Warning: 'mutationgaussian' mutation function is
for unconstrained minimization only;
using @mutationadaptfeasible mutation function.
Set @mutationadaptfeasible as MutationFcn options
using GAOPTIMSET.

Optimization terminated: current tolerance on f(x)
1e-007 is less than options.TolFun
 and constraint violation is less than
options.TolCon.

x =

    0.8122   12.3122

fval =

  1.3578e+004
```

Примечание для задачи минимизации при наличии ограничений функция ga изменила мутационную функцию на новую @mutationadaptfeasible. Принимаемая по умолчанию функция мутации @mutationgaussian подходит только для задач минимизации без наличия ограничений

Решатель Генетического алгоритма обращается с линейными ограничениями и границами несколько отличным образом по сравнению со случаем с нелинейными ограничениями. Все линейные

ограничения и границы выполняются по всему ходу оптимизации. Однако функция ga применительно к части поколений может и не соблюдать все необходимые нелинейные ограничения. Как только ga будет сходиться к некоему решению, то и нелинейные ограничения будут выполняться в этом случае.

Функция ga использует функции мутации и кроссовера в каждом поколении для новых индивидуализированных объектов. Функция ga выполняет все линейные и связанные ограничения при помощи функций мутации и кроссовера, которые генерируют только допустимые точки. Например, в предыдущем обращении к команде ga, принимаемая по умолчанию функция мутации mutationguassian не соблюдает линейные ограничения и, таким образом, вместо нее используется функция мутации mutationadaptfeasible. Если принимать произвольную функцию мутации, то эта произвольная функция мутации должна генерировать только такие точки, которые являются допустимыми относительно линейных ограничений и пределов. Все кроссоверные функции данного инструментария генерируют такие точки, которые удовлетворяют линейным ограничениям и пределам за исключением случая с выбором функции crossoverheuristic.

С помощью функции gaoptimset зададим mutationadaptfeasible в качестве функции мутации для выбранной задачи минимизации

```
options =
gaoptimset('MutationFcn',@mutationadaptfeasible);
```

Далее запустим решатель ga.

```
[x,fval] =
ga(ObjectiveFunction,nvars,[],[],[],[],LB,UB,Constr
aintFunction,options)

Optimization terminated: current tolerance on f(x)
1e-007 is less than options.TolFun
 and constraint violation is less than
options.TolCon.

x =
```

```
    0.8122   12.3122

fval =

  1.3578e+004
```

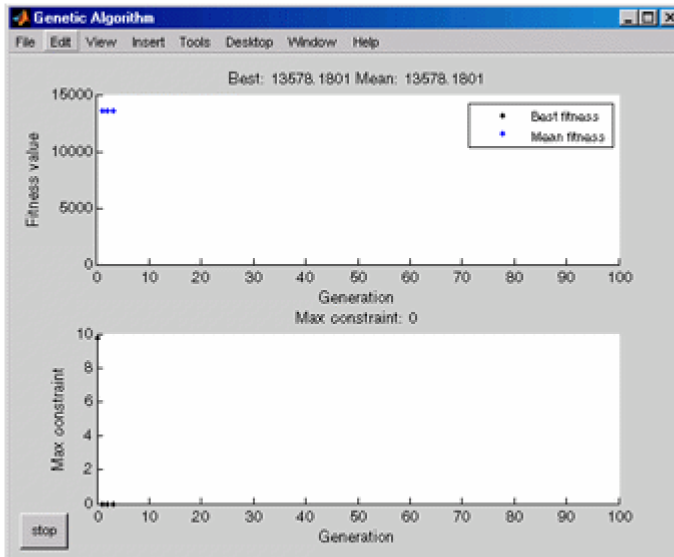
Теперь с помощью функции gaoptimset построим структуру опций, которая будет отбирать функции для двух графических зависимостей. Первой функцией графической зависимости будет gaplotbestf, которая отображает наилучшую и среднюю оценку семества для каждого поколения. Второй функцией графической зависимости будет gaplotmaxconstr, которая отображает максимальное нарушение заданного ограничения в виде нелинейных ограничений для каждого поколения. Так же имеется возможность с помощью опции 'Display' получить визуализацию прогресса алгоритма путем отображения информации в окне команд.

```
options =
gaoptimset(options,'PlotFcns',{@gaplotbestf,@gaplot
maxconstr},'Display','iter');
```

Перезапустим решатель ga, что даст:

```
[x,fval] =
ga(ObjectiveFunction,nvars,[],[],[],[],LB,UB,Constr
aintFunction,options)
                           Best        max
Stall
Generation  f-count        f(x)      constraint
Generations
    1         849      14915.8            0       0
    2        1567      13578.3            0       0
    3        2334      13578.3            0       1
    4        3043      13578.3            0       2
    5        3752      13578.3            0       3
Optimization terminated: current tolerance on f(x)
1e-009 is less than options.TolFun
```

```
 and constraint violation is less than
options.TolCon.

x =

    0.8122   12.3123


fval =

  1.3578e+004
```



Имеется возможность с помощью опции InitialPopulation задать стартовую точку минимизации для функции ga. Таким образом, функция ga будет использовать индивидуализированный объект из опции InitialPopulation в качестве стартовой точки минимизации при наличии ограничений.

```
X0 = [0.5 0.5]; % Start point (row vector)
```

```
options =
gaoptimset(options,'InitialPopulation',X0);
```
Далее перезапустим решатель ga.
```
[x,fval] =
ga(ObjectiveFunction,nvars,[],[],[],[],LB,UB,Constr
aintFunction,options)
                           Best       max
Stall
Generation  f-count        f(x)     constraint
Generations
    1        965      13579.6             0      0
    2       1728      13578.2   1.776e-015      0
    3       2422      13578.2             0      0
Optimization terminated: current tolerance on f(x)
1e-007 is less than options.TolFun
 and constraint violation is less than
options.TolCon.
x =

    0.8122   12.3122

fval =

  1.3578e+004
```

6.6.17. Параметризация функций, вызываемых с помощью ga

Иногда может быть необходимым записывать вызываемые с помощью команды ga функции с учетом дополнительных параметров к независимой переменной. Например, предположим, что необходимо минимизировать следующую функцию:

Для различных значений a, b, и c. Поскольку команда ga воспринимает функцию пригодности только как зависимую от x функцию, то необходимо до обращения к команде ga определить параметры a, b и c. Далее приводится детальное описание двух подходов к решению этой проблемы:

- Параметризация функций при помощи анонимных функциq для ga
- Параметризация функции при помощи вложенной функции для ga

Примеры из данного раздела представляют способы, как можно параметризовать целевую функцию. Однако эти же самые методы можно использовать и для любых иных функций пользователя, вызываемых при помощи команд patternsearch или ga, например для определяемого пользователем метода поиска по команде patternsearch или для определяемой пользователем масштабирующей функции для команды ga.

6.6.18. Параметризация функций при помощи анонимных функциq для ga

Для параметризации функции сначала запишем М-файл со следующими кодами:

```
function y = parameterfun(x,a,b,c)
y = (a - b*x(1)^2 + x(1)^4/3)*x(1)^2 + x(1)*x(2) +
...
         (-c + c*x(2)^2)*x(2)^2;
```

Сохраним М-файл в рабочей директории MATLAB под именем parameterfun.m

Пусть необходимо минимизировать данную функцию со следующими значениями параметров a = 4, b =2.1, и c = 4. Для этих целей определим посредством ввода в окне команд MATLAB так называемый функциональный указатель для анонимной функции.

```
a = 4; b = 2.1; c = 4;    % Define parameter values
fitfun = @(x) parameterfun(x,a,b,c);
NVARS = 3;
```

При использовании инструментария Генетического алгоритма,

- Установить параметр Fitness function в fitfun.
- Установить параметр Number of variables в NVARS.
- `a = 3.6; b = 2.4; c = 5;    % Define parameter values`
- `fitfun = @(x) parameterfun(x,a,b,c);`

6.6.19. Параметризация функции при помощи вложенной функции для ga

В качестве альтернативы методу параметризации при помощи анонимной функции можно использовать и обычный М-файл, составленный следующим образом:

- Принять a, b, c и NVARS в качестве входных параметров.
- Составить функцию пригодности в виде вложенной функции.
- Запустить команду ga.

Составим следующий М-файл.

```
function [x fval] =  runga(a,b,c,NVARS)
[x, fval] = ga(@nestedfun,NVARS);
% Вложенная функция для расчета функции пригодности
   function y = nestedfun(x)
        y = (a - b*x(1)^2 + x(1)^4/3)*x(1)^2 +
x(1)*x(2) + ...
         (-c + c*x(2)^2)*x(2)^2;
    end
end
```

Отметим, что функция пригодности рассчитывается во вложенной функции nestedfun, которая уже имеет обращение к переменным a, b и c. Для выполнения задачи оптимизации следует выполнить команду

```
 [x fval] = runga(a,b,c,NVARS)
```

После выполнения команды получим

```
Optimization terminated: average change in the
fitness value less than options.TolFun.

x =

   -0.1302    0.7170    0.2272

fval =

   -1.0254
```

Приложение

Практическая часть реализации генетических алгоритмов

Ниже приведен пример программы использующей генетический алгоритм. Программа создана посредством среды программирования С++. Алгоритм компонента представляет собой применение методов, известных в теории эволюции, для эвристического поиска решений переборных задач.

1. Математическое обоснование принципа работы программы

Проверим эффективность работы генетических алгоритмов на примере нахождения значений коэффициентов неизвестных в Диофáнтовом уравнении.

Диофáнтово уравнение или уравнение в целых числах — это уравнение с целыми коэффициентами и неизвестными, которые могут принимать только целые значения. Названы в честь древнегреческого математика Диофанта.

Рассмотрим диофантово уравнение: a+2b+3c+4d=30, где a, b, c и d — некоторые положительные целые. Применение ГА за очень короткое время находит искомое решение (a, b, c, d).

Архитектура ГА-систем позволяет найти решение быстрее за счет более 'осмысленного' перебора. Мы не перебираем все подряд, но приближаемся от случайно выбранных решений к лучшим.

Для начала выберем 5 случайных решений: 1 =< a,b,c,d =< 30. Вообще говоря, мы можем использовать меньшее ограничение для b,c,d, но для упрощения пусть будет 30.

Хромосома	**(a,b,c,d)**
1	(1,28,15,3)

2	(14,9,2,4)
3	(13,5,7,3)
4	(23,8,16,19)
5	(9,13,5,2)

Таблица 1: 1-е поколение хромосом и их содержимое

Чтобы вычислить коэффициенты выживаемости (fitness), подставим каждое решение в выражение a+2b+3c+4d. Расстояние от полученного значения до 30 и будет нужным значением.

Хромосома	**Коэффициент выживаемости**
1	|114-30|=84
2	|54-30|=24
3	|56-30|=26
4	|163-30|=133
5	|58-30|=28

Таблица 2: Коэффициенты выживаемости первого поколения хромосом (набора решений)

Так как меньшие значения ближе к 30, то они более желательны. В нашем случае большие численные значения коэффициентов выживаемости подходят, увы, меньше. Чтобы создать систему, где хромосомы с более подходящими значениями имеют большие шансы оказаться родителями, мы должны вычислить, с какой вероятностью (в %) может быть выбрана каждая. Одно решение заключается в том, чтобы взять сумму обратных значений коэффициентов, и исходя из этого вычислять проценты. (Заметим, что все решения были сгенерированы Генератором Случайных Чисел — ГСЧ)

Хромосома	**Подходящесть**
1	(1/84)/0.135266 = 8.80%
2	(1/24)/0.135266 = 30.8%
3	(1/26)/0.135266 = 28.4%

4	(1/133)/0.135266 = 5.56%
5	(1/28)/0.135266 = 26.4%

Таблица 3: Вероятность оказаться родителем

Для выбора 5-и пар родителей (каждая из которых будет иметь 1 потомка, всего — 5 новых решений), представим, что у нас есть 10000-стонняя игральная кость, на 880 сторонах отмечена хромосома 1, на 3080 — хромосома 2, на 2640 сторонах — хромосома 3, на 556 — хромосома 4 и на 2640 сторонах отмечена хромосома 5. Чтобы выбрать первую пару кидаем кость два раза и выбираем выпавшие хромосомы. Таким же образом выбирая остальных, получаем:

Хромосома отца	Хромосома матери
3	1
5	2
3	5
2	5
5	3

Таблица 4: Симуляция выбора родителей

Каждый потомок содержит информацию о генах и отца и от матери. Вообще говоря, это можно обеспечить различными способами, однако в нашем случае можно использовать т.н. «кроссовер» (cross-over). Пусть мать содержит следующий набор решений: a1,b1,c1,d1, а отец — a2,b2,c2,d2, тогда возможно 6 различных кроссоверов (| = разделительная линия):

Хромосома-отец	Хромосома-мать	Хромосома-потомок
a1 | b1 ,c1 ,d1	a2 | b2 ,c2 ,d2	a1 ,b2 ,c2 ,d2 or a2 ,b1 ,c1 ,d1
a1 ,b1 | c1 ,d1	a2 ,b2 | c2 ,d2	a1 ,b1 ,c2 ,d2 or a2 ,b2 ,c1 ,d1
a1 ,b1 ,c1 | d1	a2 ,b2 ,c2 | d2	a1 ,b1 ,c1 ,d2 or a2 ,b2 ,c2 ,d1

Таблица 5: Кроссоверы между родителями

Есть достаточно много путей передачи информации потомку, и кроссовер — только один из них. Расположение разделителя может быть абсолютно произвольным, как и то, отец или мать будут слева от черты.

А теперь попробуем проделать это с нашими потомками

Хромосома-отец	Хромосома-мать	Хромосома-потомок
(13 | 5,7,3)	(1 | 28,15,3)	(13,28,15,3)
(9,13 | 5,2)	(14,9 | 2,4)	(9,13,2,4)
(13,5,7 | 3)	(9,13,5 | 2)	(13,5,7,2)
(14 | 9,2,4)	(9 | 13,5,2)	(14,13,5,2)
(13,5 | 7, 3)	(9,13 | 5, 2)	(13,5,5,2)

Таблица 6: Симуляция кросс-оверов хромосом родителей

Теперь мы можем вычислить коэффициенты выживаемости (fitness) потомков.

Хромосома-потомок	Коэффициент выживаемости
(13,28,15,3)	|126-30|=96
(9,13,2,4)	|57-30|=27
(13,5,7,2)	|57-30|=22
(14,13,5,2)	|63-30|=33
(13,5,5,2)	|46-30|=16

Таблица 7: Коэффициенты выживаемости потомков (fitness)

Средняя приспособленность (fitness) потомков оказалась 38.8, в то время как у родителей этот коэффициент равнялся 59.4. Следующее поколение может мутировать. Например, мы можем заменить одно из значений какой-нибудь хромосомы на случайное целое от 1 до 30.

Продолжая таким образом, одна хромосома в конце концов достигнет коэффициента выживаемости 0, то есть станет решением.

Системы с большей популяцией (например, 50 вместо 5-и сходятся к желаемому уровню (0) более быстро и стабильно.

1. 1 Принцип работы программы

Обранимся к теоретическим пояснениям в практической реализации данной задачи в среде программирования С++ :

Первым делом посмотрим на заголовок класса:

```
#include <stdlib.h>

#include <time.h>

#define MAXPOP 25

struct gene {

int alleles[4];

int fitness;

float likelihood;

// Test for equality.

operator==(gene gn) {

for (int i=0;i<4;i++) {

if (gn.alleles[i] != alleles[i]) return false;

}

return true;
```

```
}
};
class CDiophantine {
public:
CDiophantine(int, int, int, int, int);
int Solve();
// Returns a given gene.
gene GetGene(int i) { return population[i];}
protected:
int ca,cb,cc,cd;
int result;
gene population[MAXPOP];
int Fitness(gene &);
void GenerateLikelihoods();
float MultInv();inverse.
int CreateFitnesses();
void CreateNewPopulation();
int GetIndex(float val);
```

```
gene Breed(int p1, int p2);

};
```

Существуют две структуры: gene и класс CDiophantine. gene используется для слежения за различными наборами решений. Создаваямая популяция — популяция ген. Эта генетическая структура отслеживает свои коэффициенты выживаемости и вероятность оказаться родителем. Также есть небольшая функция проверки на равенство.

Теперь по функциям:

Fitness function

Вычисляет коэффициент выживаемости (приспособленности — fitness) каждого гена. В нашем случае это — модуль разности между желаемым результатом и полученным значением. Этот класс использует две функции: первая вычисляет все коэффициенты, а вторая – поменьше — вычисляет коэффициент для какого-то одного гена.

```
int CDiophantine::Fitness(gene &gn) {

int total = ca * gn.alleles[0] + cb * gn.alleles[1]

+ cc * gn.alleles[2] + cd * gn.alleles[3];

return gn.fitness = abs(total — result);

}

int CDiophantine::CreateFitnesses() {

float avgfit = 0;

int fitness = 0;
```

```
for(int i=0;i<MAXPOP;i++) {

fitness = Fitness(population[i]);

avgfit += fitness;

if (fitness == 0) {

return i;

}

}

return 0;

}
```

Заметим, что если fitness = 0, то найдено решение — возврат. После вычисления приспособленности (fitness) нам нужно вычислить вероятность выбора этого гена в качестве родительского.

Likelihood functions

Как и было объяснено, вероятность вычисляется как сумма обращенных коэффициентов, деленная на величину, обратную к коэффициенту данному значению. Вероятности кумулятивны (складываются), что делает очень легким вычисления с родителями. Например:

Хромосома	**Вероятность**
1	(1/84)/0.135266 = 8.80%
2	(1/24)/0.135266 = 30.8%
3	(1/26)/0.135266 = 28.4%
4	(1/133)/0.135266 = 5.56%
5	(1/28)/0.135266 = 26.4%

В программе, при одинаковых начальных значениях, вероятности сложатся: представьте их в виде кусков пирога. Первый ген — от 0 до 8.80%, следующий идет до 39.6% (так как он начинает 8.8). Таблица вероятностей будет выглядеть приблизительно так:

Хромосома	**Вероятность (smi = 0.135266)**
1	(1/84)/smi = 8.80%
2	(1/24)/smi = 39.6% (30.6+8.8)
3	(1/26)/smi = 68% (28.4+39.6)
4	(1/133)/smi = 73.56% (5.56+68)
5	(1/28)/smi = 99.96% (26.4+73.56)

Последнее значение всегда будет 100. Имея в нашем арсенале теорию, посмотрим на код. Он очень прост: преобразование к float необходимо для того, чтобы избегать целочисленного деления. Есть две функции: одна вычисляет smi, а другая генерирует вероятности оказаться родителем.

```
float CDiophantine::MultInv() {

float sum = 0;

for(int i=0;i<MAXPOP;i++) {

sum += 1/((float)population[i].fitness);

}

return sum;

}

void CDiophantine::GenerateLikelihoods() {

float multinv = MultInv();
```

```
float last = 0;

for(int i=0;i<MAXPOP;i++) {

population[i].likelihood = last

= last + ((1/((float)population[i].fitness) / multinv) * 100);

}

}
```

Итак, у нас есть и коэффициенты выживаемости (fitness) и необходимые вероятности (likelihood). Можно переходить к размножению (breeding).

Breeding Functions

Функции размножения состоят из трех: получить индекс гена, отвечающего случайному числу от 1 до 100, непосредственно вычислить кроссовер двух генов и главной функции генерации нового поколения. Рассмотрим все эти функции одновременно и то, как они друг друга вызывают. Вот главная функция размножения:

```
void CDiophantine::CreateNewPopulation() {

gene temppop[MAXPOP];

for(int i=0;i<MAXPOP;i++) {

int parent1 = 0, parent2 = 0, iterations = 0;

while(parent1 == parent2 || population[parent1]

== population[parent2]) {

parent1 = GetIndex((float)(rand() % 101));
```

```
parent2 = GetIndex((float)(rand() % 101));

if (++iterations > (MAXPOP * MAXPOP)) break;

}

temppop[i] = Breed(parent1, parent2); // Create a child.

}

for(i=0;i<MAXPOP;i++) population[i] = temppop[i];

}
```

Итак, первым делом мы создаем случайную популяцию генов. Затем делаем цикл по всем генам. Выбирая гены, мы не хотим, чтобы они оказались одинаковы (ни к чему скрещиваться с самим собой, и вообще — нам не нужны одинаковые гены (operator = в gene). При выборе родителя, генерируем случайное число, а затем вызываем GetIndex. GetIndex использует идею кумулятивности вероятностей (likelihoods), она просто делает итерации по всем генам, пока не найден ген, содержащий число:

```
int CDiophantine::GetIndex(float val) {

float last = 0;

for(int i=0;i<MAXPOP;i++) {

if (last <= val && val <= population[i].likelihood) return i;

else last = population[i].likelihood;

}

return 4;
```

```
}
```

Возвращаясь к функции CreateNewPopulation(): если число итераций превосходит MAXPOP2, она выберет любых родителей. После того, как родители выбраны, они скрещиваются: их индексы передаются вверх на функцию размножения (Breed). Breed function возвращает ген, который помещается во временную популяцию. Вот код:

```
gene CDiophantine::Breed(int p1, int p2) {

int crossover = rand() % 3+1;

int first = rand() % 100;

gene child = population[p1];

int initial = 0, final = 3;

if (first < 50) initial = crossover;

else final = crossover+1;

for(int i=initial;i<final;i++) {

child.alleles[i] = population[p2].alleles[i];

if (rand() % 101 < 5) child.alleles[i] = rand() % (result + 1);

}

return child;

}
```

В конце концов мы определим точку кроссовера. Заметим, что мы не хотим, чтобы кроссовер состоял из копирования только одного родителя. Сгенерируем случайное число, которое определит наш

кроссовер. Остальное понятно и очевидно. Добавлена маленькая мутация, влияющая на скрещивание. 5% — вероятность появления нового числа.

Теперь уже можно взглянуть на функцию Solve(), которая возвратит аллель, содержащую решение. Она всего лишь итеративно вызывает вышеописанные функции. Заметим, что мы присутствует проверка: удалось ли функции получить результат, используя начальную популяцию. Это маловероятно, однако лучше проверить.

```
int CDiophantine::Solve() {

int fitness = -1;

// Generate initial population.

srand((unsigned)time(NULL));

for(int i=0;i<MAXPOP;i++) {

for (int j=0;j<4;j++) {

population[i].alleles[j] = rand() % (result + 1);

}

}

if (fitness = CreateFitnesses()) {

return fitness;

}

int iterations = 0;

while (fitness != 0 || iterations < 50) {
```

```
GenerateLikelihoods();
CreateNewPopulation();
if (fitness = CreateFitnesses()) {
return fitness;
}
iterations++;
}
return -1;
}
```

Описаниезавершено.

2. Листинг программы

```
#include <stdlib.h>
#include <time.h>
#include <iostream.h>
#define MAXPOP 25
struct gene {
int alleles[4];
int fitness;
```

```
float likelihood;

// Test for equality.

operator==(gene gn) {

for (int i=0;i<4;i++) {

if (gn.alleles[i] != alleles[i] }

return false;

}

return true;

}

};

class CDiophantine {

public:

CDiophantine(int, int, int, int, int); // Constructor with coefficients for
a,b,c,d.

int Solve(); // Solve the equation.

// Returns a given gene.

gene GetGene(int i) { return population[i];}

protected:

int ca,cb,cc,cd; // The coefficients.
```

```
int result;

gene population[MAXPOP]; // Population.

int Fitness(gene &); // Fitness function.

void GenerateLikelihoods(); // Generate likelihoods.

float MultInv(); // Creates the multiplicative inverse.

int CreateFitnesses();

void CreateNewPopulation();

int GetIndex(float val);

gene Breed(int p1, int p2);

};

CDiophantine::CDiophantine(int a, int b, int c, int d, int res): ca(a), cb(b),
cc(c), cd(d), result(res) {}

int CDiophantine::Solve() {

int fitness = -1;

// Generate initial population.

srand((unsigned)time(NULL));

for(int i=0;i<MAXPOP;i++) { // Fill the population with numbers between

for (int j=0;j<4;j++) { // 0 and the result.

population[i].alleles[j] = rand() % (result + 1);
```

```
}

}

if (fitness = CreateFitnesses()) {

return fitness;

}

int iterations = 0; // Keep record of the iterations.

while (fitness != 0 || iterations < 50) {// Repeat until solution found, or over
50 iterations.

GenerateLikelihoods(); // Create the likelihoods.

CreateNewPopulation();

if (fitness = CreateFitnesses()) {

return fitness;

}

iterations++;

}

return -1;

}

int CDiophantine::Fitness(gene &gn) {
```

```
int total = ca * gn.alleles[0] + cb * gn.alleles[1] + cc * gn.alleles[2] + cd *
gn.alleles[3];

return gn.fitness = abs(total — result);

}

int CDiophantine::CreateFitnesses() {

float avgfit = 0;

int fitness = 0;

for(int i=0;i<MAXPOP;i++) {

fitness = Fitness(population[i]);

avgfit += fitness;

if (fitness == 0) {

return i;

}

}

return 0;

}

float CDiophantine::MultInv() {

float sum = 0;

for(int i=0;i<MAXPOP;i++) {
```

```
sum += 1/((float)population[i].fitness);

}

return sum;

}

void CDiophantine::GenerateLikelihoods() {

float multinv = MultInv();

float last = 0;

for(int i=0;i<MAXPOP;i++) {

population[i].likelihood = last = last + ((1/((float)population[i].fitness) /
multinv) * 100);

}

}

int CDiophantine::GetIndex(float val) {

float last = 0;

for(int i=0;i<MAXPOP;i++) {

if (last <= val && val <= population[i].likelihood) return i;

else last = population[i].likelihood;

}

return 4;
```

```
}
gene CDiophantine::Breed(int p1, int p2) {
int crossover = rand() % 3+1; // Create the crossover point (not first).
int first = rand() % 100; // Which parent comes first?
gene child = population[p1]; // Child is all first parent initially.
int initial = 0, final = 3; // The crossover boundaries.
if (first < 50) initial = crossover; // If first parent first. start from crossover.
else final = crossover+1; // Else end at crossover.
for(int i=initial;i<final;i++) { // Crossover!
child.alleles[i] = population[p2].alleles[i];
if (rand() % 101 < 5) child.alleles[i] = rand() % (result + 1);
}
return child; // Return the kid...
}
void CDiophantine::CreateNewPopulation() {
gene temppop[MAXPOP];
for(int i=0;i<MAXPOP;i++) {
int parent1 = 0, parent2 = 0, iterations = 0;
```

```
while(parent1 == parent2 || population[parent1] == population[parent2]) {

parent1 = GetIndex((float)(rand() % 101));

parent2 = GetIndex((float)(rand() % 101));

if (++iterations > 25) break;

}

temppop[i] = Breed(parent1, parent2); // Create a child.

}

for(i=0;i<MAXPOP;i++) population[i] = temppop[i];

}

void main() {

CDiophantine dp(1,2,3,4,30);

int ans;

ans = dp.Solve();

if (ans == -1) {

cout << «No solution found.» << endl;

} else {

gene gn = dp.GetGene(ans);

cout << «The solution set to a+2b+3c+4d=30 is:\n»;
```

```
cout << «a = » << gn.alleles[0] << "." << endl;

cout << «b = » << gn.alleles[1] << "." << endl;

cout << «c = » << gn.alleles[2] << "." << endl;

cout << «d = » << gn.alleles[3] << "." << endl;

}

}
```

Примеры применения ГА

По следующим ссылкам вы можете найти интересные примеры, демонстрирующие работу генетического алгоритма и его применение.

- Демонстрация работы классического ГА на многоэкстремальной функции (http://ai.bpa.arizona.edu/~mramsey/ga.html).
- Решение задачи коммивояжера (TSP) при помощи ГА (http://lib.training.ru/Lib/ArticleDetail.aspx?ar=803&l=&mi=93&mic=112).
- Обучение модели человека ходьбе при помощи ГА (http://www.naturalmotion.com/pages/technology.htm).
- Еще одна демонстрация работы ГА (http://www.rennard.org/alife/english/gavgb.html).

Литература

1. Darrel Whitley, A Genetic Algorithm Tutorial, Statistics and Computing (4): 65–85, 1994.
2. Darrel Whitley, An Overview of Evolutionary Algorithms: Practical Issues and Common Pitfalls, Journal of Information and Software Technology 43: 817–831, 2001.
3. K. Deb, S. Agrawal, Understanding Interactions Among Genetic Algorithm Parameters, 1998.
4. Авторский сайт Ю. Цоя (http://www.qai.narod.ru/).
5. Исаев С.А. Популярно о генетических алгоритмах (http://algolist.manual.ru/ai/ga/ga1.php).
6. Авторский сайт Ю. Цоя (http://www.qai.narod.ru/).
7. Исаев С.А. Популярно о генетических алгоритмах (http://algolist.manual.ru/ai/ga/ga1.php).
8. http://www.gotai.net/ - сайт по ИИ.
9. http://neuronet.alo.ru/
10. http://www.neuroproject.ru/ – сайт компании, которая занимается разработкой программного обеспечения с использованием генетических алгоритмов и нейронных сетей.
11. Вороновский Г.К., Махотило К.В., Петрашев С.Н., Сергеев С.А., Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности, Харьков, ОСНОВА, 1997. – 112с.
12. Holland J. H. Adaptation in natural and artificial systems. An introductory analysis with application to biology, control, and artificial intelligence.— London: Bradford book edition, 1994 — 211 p.
13. De Jong K.A. An analysis of the behavior of a class of genetic adaptive systems. Unpublished PhD thesis. University of Michigan, Ann Arbor, 1975. (Also University Microfilms No. 76-9381).
14. De Jong K.A., Spears W.M. An Analysis of the Interacting Roles of Population Size and Crossover // Proceedings of the International Workshop «Parallel Problems Solving from Nature» (PPSN'90), 1990.
15. De Jong K.A., Spears W.M. A formal analysis of the role of multi-point crossover in genetic algorithms. // Annals of Mathematics and Artificial Intelligence, no. 5(1), 1992.
16. Darrel Whitley "A Genetic Algorithm Tutorial", 1993.

17. Darrel Whitley, A Genetic Algorithm Tutorial, Statistics and Computing (4), 1994.
18. Darrel Whitley, An Overview of Evolutionary Algorithms: Practical Issues and Common Pitfalls, Journal of Information and Software Technology, 2001.
19. Mitchell M. An Introduction to Genetic Algorithms. Cambridge, MA: The MIT Press, 1996.
20. K. Deb, S. Agrawal, Understanding Interactions Among Genetic Algorithm Parameters, 1998.
21. Robin Biesbroek "Genetic Algorithm Tutorial. 4.1 Mathematical foundations", 1999.
22. Soraya Rana "Examining the Role of Local Optima and Schema Processing in Genetic Search", 1999.
23. David E. Goldberg, Kumara Sastry "A Practical Schema Theorem for Genetic Algorithm Design and Tuning", 2001.
24. Koza, John R. Genetic programming: on the programming of computers by means of natural selection, A Bradford book, The MIT Press, London, 1992.
25. Кононюк А. Е Дискретно-непрерывная математика. (Алгоритмы). —В 12-и кн. Кн. 10,Ч.1— К.: 2017. — 608 с.
26. Кононюк А. Е Дискретно-непрерывная математика. (Алгоритмы). —В 12-и кн. Кн. 10,Ч.2— К.: 2017. —544 с.

- Статья «Моделируя жизнь», автор Андрей Тепляков
- *Metropolis, N.*, *Ulam, S.* The Monte Carlo Method, — Journal of the American Statistical Association 1949 **44** № 247 335—341.
- *Alex F Bielajew*. Fundamentals of the Monte Carlo method for neutral and charged particle transport. 2001
- *W. M. C. Foulkes, L. Mitas, R. J. Needs and G. Rajagopal* Quantum Monte Carlo simulations of solids, — Reviews of Modern Physics 73 (2001) 33.
- Статья «Metropolis, Monte Carlo and the MANIAC»

- *Fishman, George S.* Monte Carlo : concepts, algorithms, and applications. — Springer, 1996. — ISBN 0-387-94527-X.
- Vitter's original paper: J. S. Vitter, «Проектирование и анализ динамических кодов Хаффмана», журнал ACM, 34(4), октябрь 1987 г., стр. 825—845.
- J. S. Vitter, «ALGORITHM 673 Dynamic Huffman Coding», ACM Transactions on Mathematical Software, 15(2), June 1989, pp 158–167. Also appears in Collected Algorithms of ACM.
- Donald E. Knuth, «Dynamic Huffman Coding», Journal of Algorithm, 6(2), 1985, pp 163–180.