

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ

ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Ю.А.СКОБЦОВ

«ОСНОВЫ ЭВОЛЮЦИОННЫХ ВЫЧИСЛЕНИЙ»

Учебное пособие

Донецк 2008

УДК 621.3+681.3

ББК 32.813

С__

Рекомендовано Міністерством освіти і науки України
як навчальний посібник для студентів вищих учбових закладів
(лист № 14/181-531 від 26.02. 2008 р.)

Рецензенти:

Бідюк П.І., доктор техн. наук, професор
(Національний технічний університет України “КПІ”);

Каргин А.О., доктор техн. наук, професор
(Донецький Національний університет);

Ходаков В.Є., доктор техн. наук, професор
(Херсонський Національний технічний університет)

Ю.О.Скобцов.

С__ **Основи еволюційних обчислень.**- Навчальний посібник.-
Донецьк: ДонНТУ, 2008.-326с.

ISBN_____

У посібнику викладені основні теоретичні і методологічні матеріали з еволюційних обчислень, які є самостійним напрямом в теорії інтелектуальних систем. Послідовно вводяться базисні поняття класичних генетичних алгоритмів (ГА), теорія схем і рішення задач чисельної і комбінаторної оптимізації за допомогою генетичних алгоритмів. Розглянуті узагальнення і модифікації ГА, паралельні і імовірнісні ГА. Викладені основи генетичного програмування і машинного навчання, еволюційні стратегії і програмування. Розглянуті питання програмної і апаратної реалізації ГА.

Посібник розраховано, насамперед на студентів і аспірантів ВНЗ, які навчаються за напрямками “Комп’ютерна інженерія”, “Комп’ютерні науки”, “Програмна інженерія” а також широкому колу фахівців, що займаються інтелектуальними системами обробки інформації.

УДК 621.3+681.3

ББК 32.813

ISBN_____

©Ю.А.Скобцов

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ	9
ЧАСТЬ 1. ВВЕДЕНИЕ В ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ	
1. ОСНОВЫ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ	16
1.1 Простой генетический алгоритм.....	16
1.2 Генетические операторы.....	20
1.3 Представление вещественных решений в двоичной форме.....	26
1.4 Использование кода Грея в ГА.....	30
1.5 Логарифмическое кодирование.....	33
1.6 Основная терминология в генетических алгоритмах.....	33
1.7 Контрольные вопросы к разделу 1.....	35
2. ТЕОРИЯ СХЕМ И МОДЕЛИ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ.....	37
2.1 Теория схем.....	37
2.2 Фундаментальная теорема ГА.....	39
2.3 Модель динамической системы для простого генетического алгоритма	46
2.4 Применение марковских цепей для моделирования конечных популяций.....	58
2.5 Применение методов статистической механики для моделирования ГА.....	59
2.6 Контрольные вопросы к разделу 2.....	69
3. ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ ДЛЯ ЗАДАЧ КОМБИНАТОРНОЙ ОПТИМИЗАЦИИ.....	71
3.1 Задача об укладке рюкзака.....	71
3.2 Задача о покрытии.....	77
3.3 Задача коммивояжера.....	77
3.3.1 Упорядоченное представление.....	80
3.3.2 Представление соседства.....	82
3.3.3 Представление путей.....	84

3.3.4	Матричное представление.....	87
3.3.4.1	Матрица смежности.....	87
3.3.4.2	Матрица предшествования.....	90
3.4	Контрольные вопросы к разделу 3.....	96

ЧАСТЬ 2. МОДИФИКАЦИИ И ОБОБЩЕНИЯ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ

4.	МОДИФИКАЦИИ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ.....	99
4.1	Создание исходной популяции.....	99
4.2	Отбор родителей (селекция).....	100
4.2.1	Пропорциональный отбор (метод "рулетки").....	101
4.2.2	Ранжирование.....	103
4.2.3	Равномерное ранжирование.....	105
4.2.4	Локальный отбор.....	105
4.2.5	Отбор на основе усечения.....	108
4.2.6	Турнирный отбор.....	108
4.2.7	Метод Больцмана.....	110
4.2.8	Методы выбора пар для скрещивания.....	110
4.2.9	Неявные методы отбора, основанные на масштабировании фитнес- функции.....	112
4.3	Операторы рекомбинации (скрещивания, кроссинговера).....	113
4.3.1	Двоичная рекомбинация.....	113
4.3.1.1	Одноточечный кроссинговер.....	113
4.3.1.2	Многоточечный кроссинговер.....	114
4.3.1.3	Однородный кроссинговер.....	116
4.3.1.4	Ограниченный кроссинговер.....	117
4.3.2	Рекомбинация действительных значений.....	117
4.3.2.1	Дискретная рекомбинация.....	117
4.3.2.2	Промежуточная рекомбинация.....	118
4.3.2.3	Линейная рекомбинация.....	120

4.4	Оператор мутации.....	120
4.4.1	Двоичная мутация.....	121
4.4.1.1	Классическая мутация.....	121
4.4.1.2	Оператор инверсии.....	121
4.4.2	Мутация над вещественными числами.....	122
4.5	Сокращение промежуточной популяции.....	124
4.5.1	Глобальная редукция.....	124
4.5.1.1	Чистая замена.....	125
4.5.1.2	Элитарная схема.....	125
4.5.1.3	Равномерная случайная замена.....	125
4.5.1.4	Пропорциональная редукция.....	126
4.5.1.5	Селекционная схема.....	126
4.5.2	Локальная замена.....	126
4.6	Асинхронные генетические алгоритмы.....	127
4.7	ГА с изменяемой мощностью популяций.....	128
4.8	Адаптивные ГА.....	132
4.9	Ниши в генетических алгоритмах	136
4.10	Многокритериальная оптимизация.....	138
4.11	Генетические микро алгоритмы.....	140
4.12	Контрольные вопросы к разделу 4.....	141
5.	ПАРАЛЛЕЛЬНЫЕ ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ.....	143
5.1	Структуризация ГА.....	143
5.2	Параллельный генетический алгоритм на основе модели «рабочий хозяин».....	145
5.3	Параллельные генетические алгоритмы на основе «модели островов».....	149
5.4	Виды распределенных ГА.....	152
5.5	Клеточные ГА.....	156
5.6	Козволюционные ГА.....	158

5.7	Параллельная реализация ГА.....	159
5.8	Инструментарий распараллеливания	162
5.9	Иерархические (многоуровневые) ГА.....	163
5.10	Контрольные вопросы к разделу 5.....	164
6.	ВЕРОЯТНОСТНЫЕ И КОМПАКТНЫЕ ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ.....	166
6.1	Вероятностные генетические алгоритмы.....	166
6.2	Пошаговое обучение на основе виртуальной популяции.....	168
6.3	Компактный генетический алгоритм.....	170
6.4	Генетический алгоритм SELFISH	172
6.5	Сравнение простых и вероятностных генетических алгоритмов.....	174
6.6	Контрольные вопросы к разделу 6.....	177
	ЧАСТЬ 3. ГЕНЕТИЧЕСКОЕ ПРОГРАММИРОВАНИЕ И МАШИННОЕ ОБУЧЕНИЕ	
7.	ГЕНЕТИЧЕСКОЕ ПРОГРАММИРОВАНИЕ.....	179
7.1	Функциональное и терминальное множество.....	180
7.1.1	Терминальное множество.....	180
7.1.2	Функциональное множество.....	181
7.2	Структуры для представления программ.....	183
7.2.1	Древовидное представление.....	183
7.2.2	Линейные структуры.....	185
7.2.3	Графоподобные структуры.....	187
7.2.4	Другие формы представления программ.....	192
7.3	Инициализация начальной популяции.....	195
7.3.1	Инициализация древовидных структур.....	196
7.3.2	Инициализация линейных структур.....	197
7.4	Кроссинговер в генетическом программировании.....	198
7.4.1	Выполнение кроссинговера на древовидных	198

	структурах.....	
7.4.2	Кроссинговер на линейных структурах.....	202
7.4.3	Выполнение кроссинговера для графоподобных структур.....	203
7.5	Мутация в генетическом программировании.....	205
7.5.1	Выполнение мутации на древовидных структурах.....	205
7.5.2	Выполнение мутации на линейных структурах.....	208
7.5.3	Выполнение мутации на графоподобных структурах...	209
7.6	Фитнесс-функция в генетическом программировании.....	209
7.7	Интроны.....	213
7.8	Общий алгоритм генетического программирования.....	213
7.9	Символьная регрессия.....	214
7.10	Диагностическое дерево	223
7.11	Модульное построение программ в генетическом программировании.....	238
7.12	Контрольные вопросы к разделу 7.....	242
8.	МАШИННОЕ ОБУЧЕНИЕ.....	244
8.1	Питсбургский подход	245
8.2	Мичиганский подход.....	250
8.3	Применение ГА в задачах прогнозирования.....	257
8.4	Контрольные вопросы к разделу 8.....	265
ЧАСТЬ 4. ЭВОЛЮЦИОННЫЕ СТРАТЕГИИ И ПРОГРАММИРОВАНИЕ, РЕАЛИЗАЦИЯ		
9.	ЭВОЛЮЦИОННЫЕ СТРАТЕГИИ.....	267
9.1	Двукратная эволюционная стратегия.....	269
9.2	Многократная эволюционная стратегия.....	273
9.3	Сравнение эволюционной стратегии и генетических алгоритмов.....	277
9.4	Контрольные вопросы к разделу 9.....	279

10. ЭВОЛЮЦИОННОЕ ПРОГРАММИРОВАНИЕ.....	280
10.1 Конечный автомат в качестве генома.....	281
10.2 Применение эволюционного программирования в прогнозировании.....	287
10.3 Применение эволюционного программирования в задачах управления.....	290
10.4 Современные направления эволюционного программирования.....	291
10.5 Контрольные вопросы к разделу 10.....	292
11. РЕАЛИЗАЦИЯ ЭВОЛЮЦИОННЫХ АЛГОРИТМОВ.....	294
11.1 Программная реализация.....	294
11.1.1 Пакет Evolver фирмы Palisade Corp.....	295
11.1.2 Пакет GeneHunter фирмы Ward System Group.....	299
11.1.3 Пакет Genetic Training Option (GTO) фирмы California Scientific Software	301
11.1.4 Программа FlexTool	301
11.1.5 MATLAB 7 - Genetic algorithm toolbox.....	302
11.1.6 Некоммерческое программное обеспечение	304
11.2 Аппаратная реализация генетических алгоритмов.....	305
11.2.1 Аппаратная реализация классического генетического алгоритма.....	306
11.2.2 Аппаратная реализация компактного ГА.....	316
11.2.3 Контрольные вопросы к разделу 11.....	319
ЛИТЕРАТУРА	321

ПРЕДИСЛОВИЕ

В настоящее время быстро развивается новое направление в теории и практике искусственного интеллекта – эволюционные вычисления (ЭВ). Этот термин обычно используется для общего описания алгоритмов поиска, оптимизации или обучения основанных на некоторых формализованных принципах естественного эволюционного отбора. Особенности идей эволюции и самоорганизации заключаются в том, что они находят подтверждение не только для биологических систем развивающихся много миллиардов лет. Эти идеи в настоящее время с успехом используются при разработке многих технических и, в особенности, программных систем.

История эволюционных вычислений началась в 60-е годы XX века, когда различные группы ученых в области кибернетики независимо друг от друга исследовали возможности применения принципов эволюции биологических систем при решении различных технических проблем, как правило, требующих решения задач оптимизации. Таким образом, было основано новое научное направление, которое в настоящее время принято называть "эволюционными вычислениями".

Генетические алгоритмы (ГА) были основаны в Мичиганском университете американским исследователем Холландом [1] и первоначально разработаны для задач оптимизации в качестве достаточно эффективного механизма комбинаторного перебора вариантов решения. В отличие от многих других работ, целью Холланда было не только решение конкретных задач, но исследование явления адаптации в биологических системах и применение его в вычислительных системах. При этом потенциальное решение – особь представляется хромосомой – двоичным кодом. Популяция содержит множество особей. В процессе эволюции

используются три основных генетических оператора: репродукция, кроссинговер и мутация. Голдберг [2] (ученик Холланда) успешно разработал ГА и расширил области его применения.

Также в 60-х годах в Германии Рехенберг [3] заложил основы "эволюционных стратегий" (ЭС) при решении задачи оптимизации вещественных параметров в расчете линий электропередачи. Это направление развивалось долгие годы независимо и здесь были получены важные фундаментальные результаты. В ЭС потенциальное решение – особь является вектором вещественных чисел, популяция состоит из двух особей и основным генетическим оператором является мутация.

Фогель [4] независимо от других исследователей основал эволюционное программирование (ЭП), где потенциальное решение представляется конечным автоматом. Основным генетическим оператором здесь также является мутация, которая случайным образом изменяет таблицу переходов-выходов автомата.

Немного позднее Коза в Массачусетском технологическом институте США заложил основы генетического программирования (ГП) [5]. Здесь в качестве особи выступала программа на LISP, которая представлялась древовидной структурой. На этих структурах были разработаны генетические операторы кроссинговера и мутации.

В настоящее время указанные направления объединены в "эволюционные вычисления" (ЭВ), которые успешно применяются при решении многих проблем. При этом отдельные направления успешно взаимодействуют за счет заимствования лучших черт друг у друга.

Значительный вклад в исследование механизмов адаптации внесли также и отечественные ученые – Ивахненко А.Г.[6] и Цыпкин Я.З.[7] по самообучающимся системам и Расстригин Л.А. [8] по стохастической оптимизации. В настоящее время исследования по ЭВ активно развиваются в России Букатовой И.Л. [9,10], Курейчиком В.М.[11,12],

Батищевым Д.И.[13] и в Украине Верланем А.Ф.[14], Скурихиным [15], Вороновским Г.К.[16] и другими исследователями.

В науке и технике эволюционные вычисления используются в качестве адаптивных алгоритмов для решения практических задач и как вычислительная модель эволюции естественных систем. ЭВ успешно применяются при решении сложных задач в технических разработках и в бизнесе. С помощью ЭВ было разработано много промышленных проектных решений, которые позволили сэкономить миллионы долларов. ЭВ широко используются для прогнозирования развития финансовых рынков, инвестиций и т.п. Методы ЭВ обычно используются для оценки и выбора (суб)оптимальных непрерывных параметров моделей большой размерности, для решения различных NP-полных комбинаторных задач, в системах извлечения знаний из больших баз данных (Data mining) и многих других областях науки и техники. Следует отметить, что когда задача не может быть решена другими, более простыми методами, ЭВ часто могут найти оптимальные или близкие к ним решения. При этом объем вычислений может оказаться большим, но скорость, с которой он растет при увеличении размерности задачи обычно меньше, чем у остальных известных методов. После того, как компьютерные системы стали достаточно быстродействующими и недорогими, ЭВ превратились в важнейший инструмент поиска субоптимальных решений задач, которые до этого считались неразрешимыми.

Предлагаемое учебное пособие содержит основные положения эволюционных вычислений и их приложения при решении различных технических проблем. При его написании использовались современные зарубежные и отечественные монографии и статьи, которые приведены в списке литературы. В книге используются материалы курса лекций "Эволюционные вычисления в технических задачах", который автор читает в Донецком Национальном техническом университете.

В первом разделе изложены основы простого ГА, описаны генетические операторы репродукции, кроссинговера и мутации. Рассмотрены вопросы представления потенциальных решений для целых и вещественных чисел.

Во втором разделе изложены математические основы ГА – теория схем, которая показывает влияние генетических операторов на "выживаемость" отдельных групп особей, которые обладают (или наоборот не обладают) некоторыми свойствами.

Третий раздел посвящен задачам комбинаторной оптимизации. Детально рассмотрена задача коммивояжера в качестве базовой проблемы для NP-полных задач. Изложены основные виды представления потенциальных решений этой задачи и проблемно-ориентированные генетические операторы.

В четвертом разделе представлены современные модификации и обобщения ГА. Изложены различные способы, и стратегии реализации всех функциональных блоков основной блок-схемы простого ГА первого раздела. Рассмотрены различные стратегии отбора родителей, двоичные и арифметические операторы кроссинговера и мутации, методы сокращения промежуточной популяции. Представлены ГА с изменяющейся мощностью популяции с различными методами определения срока жизни; адаптивные ГА, в которых в процессе эволюции подстраиваются вероятности кроссинговера и мутации.

В пятом разделе изложены параллельные генетические алгоритмы, рассмотрены основные виды параллелизации и способы их реализации. Рассмотрены основные виды параллельных ГА: глобальные (модель «рабочий-хозяин»), распределенные («модель островов») и клеточные. Описан параллельный глобальный генетический алгоритм на основе модели «рабочий хозяин». Представлен параллельный генетический алгоритм, который реализуется согласно «модели островов» и его

основные параметры. Изложен клеточный генетический алгоритм на основе модели “диффузии”.

Шестой раздел посвящен вероятностным и компактным генетическим алгоритмам, где популяция представляется вектором вероятностей генов.

Седьмой раздел посвящен генетическому программированию. Изложены основы классического ГП на древовидных структурах, определены операторы кроссинговера и мутации на этих структурах. Представлено новое направление ГП, основанное на модели линейных графов, которое позволяет более эффективно решать многие задачи. Рассмотрены особенности реализации основных этапов ГП. Подробно изложена символьная регрессия.

Восьмой раздел представляет машинное обучение и системы классификации на основе ЭВ, где в качестве особей выступают продукции (или системы правил). Описаны Мичиганский и Питсбургский подходы к решению этой проблемы.

В девятом разделе изложены основы эволюционных стратегий, проведено сравнение ЭС и ГА.

В десятом разделе рассмотрены основы эволюционного программирования, которое основано на эволюции конечных автоматов.

Одиннадцатый раздел посвящен проблемам программной реализации эволюционных алгоритмов и содержит описание некоторых программных пакетов. Кроме этого рассмотрены вопросы аппаратной реализации генетических алгоритмов.

Материал разбит на четыре части: первая часть «Введение в генетические алгоритмы» включает разделы 1-3; вторая часть «Модификации и обобщения ГА» содержит 4-5 разделы; третья часть «Генетическое программирование и машинное обучение»

соответственно разделы 7-8; четвертая часть «Эволюционные стратегии и программирование, реализация» - разделы 9-11.

Основной целью учебного пособия является систематическое изложение нового перспективного направления в области теории и практики искусственного интеллекта. Материал книги следует изучать после знакомства с основами дискретной математики и методов оптимизации. Для лучшего усвоения материала пособие содержит контрольные вопросы и упражнения.

СПИСОК УСЛОВНЫХ ОБОЗНАЧЕНИЙ

ЭВ – эволюционные вычисления;
ГА – генетический алгоритм;
ГП – генетическое программирование;
ЭС – эволюционные стратегии;
ЭП – эволюционное программирование;
ОК – оператор кроссинговера;
ОМ – оператор мутации;
ОИ – оператор инверсии;
ВГА – вероятностный генетический алгоритм;
КГА – компактный генетический алгоритм;
ЗК – задача коммивояжера;
РГА – распределенный генетический алгоритм;
ПГА – параллельный генетический алгоритм;
КГА – клеточный генетический алгоритм;
КЭГА – коэволюционные генетические алгоритмы;
СРГА – синхронные распределенные ГА;
АРГА – асинхронные распределенные ГА;
ИИ – искусственный интеллект;
ЦФ – целевая функция;
ЭМ – эволюционное моделирование;
PVM – parallel virtual machine;
MPI – message passing interface;
PMX – partially-mapped crossover;
OX – order partially-mapped crossover;
CX – cycle partially-mapped crossover;
PBIL – Population-Based Incremental Learning.

ЧАСТЬ 1

«ВВЕДЕНИЕ В ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ»

1. ОСНОВЫ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ

1.1. Простой генетический алгоритм

Основы теории генетических алгоритмов сформулированы Холландом в основополагающей работе [1] и в дальнейшем были развиты рядом исследователей. Наиболее известной и цитируемой в настоящее время является монография Д.Голдберга [2], где систематически изложены основные результаты и области практического применения ГА.

ГА используют принципы и терминологию, заимствованные у биологической науки – генетики. В ГА каждая особь представляет потенциальное решение некоторой проблемы. В классическом ГА особь кодируется строкой двоичных символов – хромосомой, каждый бит которой называется геном. Множество особей – потенциальных решений составляет популяцию. Поиск (суб)оптимального решения проблемы выполняется в процессе эволюции популяции - последовательного преобразования одного конечного множества решений в другое с помощью генетических операторов репродукции, кроссинговера и мутации.

ЭВ используют следующие механизмы естественной эволюции:

1) Первый принцип основан на концепции выживания сильнейших и естественного отбора по Дарвину, который был сформулирован им в 1859

году в книге «Происхождение видов путем естественного отбора». Согласно Дарвину особи, которые лучше способны решать задачи в своей среде, выживают и больше размножаются (репродуцируют). В генетических алгоритмах каждая особь представляет собой решение некоторой проблемы. По аналогии с этим принципом особи с лучшими значениями целевой (фитнесс) функции имеют большие шансы выжить и репродуцировать. Формализация этого принципа, как мы увидим далее, дает оператор репродукции.

2) Второй принцип обусловлен тем фактом, что хромосома потомка состоит из частей полученных из хромосом родителей. Этот принцип был открыт в 1865 году Менделем. Его формализация дает основу для оператора скрещивания (кроссинговера).

3) Третий принцип основан на концепции мутации, открытой в 1900 году де Вре. Первоначально этот термин использовался для описания существенных (резких) изменений свойств потомков и приобретение ими свойств, отсутствующих у родителей. По аналогии с этим принципом генетические алгоритмы используют подобный механизм для резкого изменения свойств потомков и тем самым, повышают разнообразие (изменчивость) особей в популяции (множестве решений).

Эти три принципа составляют ядро ЭВ. Используя их, популяция (множество решений данной проблемы) эволюционирует от поколения к поколению.

Эволюцию искусственной популяции – поиска множества решений некоторой проблемы формально можно описать алгоритмом, который представлен на рис.1.1.



Рис.1.1. Простой генетический алгоритм

ГА берет множество параметров оптимизационной проблемы и кодирует их последовательностями конечной длины в некотором конечном алфавите (в простейшем случае двоичный алфавит «0» и «1»).

Предварительно простой ГА случайным образом генерирует начальную популяцию стрингов (хромосом). Затем алгоритм генерирует

следующее поколение (популяцию), с помощью трех основных генетических операторов:

- 1) Оператор репродукции (ОР);
- 2) Оператор скрещивания (кроссинговера, ОК);
- 3) Оператор мутации (ОМ).

Генетические операторы являются математической формализацией приведенных выше трех основополагающих принципов Дарвина, Менделя и де Вре естественной эволюции.

ГА работает до тех пор, пока не будет выполнено заданное количество поколений (итераций) процесса эволюции или на некоторой генерации будет получено заданное качество или вследствие преждевременной сходимости при попадании в некоторый локальный оптимум.

В каждом поколении множество искусственных особей создается с использованием старых и добавлением новых с хорошими свойствами. Генетические алгоритмы - не просто случайный поиск, они эффективно используют информацию накопленную в процессе эволюции.

В процессе поиска решения необходимо соблюдать баланс между "эксплуатацией" полученных на текущий момент лучших решений и расширением пространства поиска. Различные методы поиска решают эту проблему по-разному.

Например, градиентные методы практически основаны только на использовании лучших текущих решений, что повышает скорость сходимости с одной стороны, но порождает проблему локальных экстремумов с другой стороны. В полярном подходе случайные методы поиска используют все пространство поиска, но имеют низкую скорость сходимости. В ГА предпринята попытка объединить преимущества этих двух противоположных подходов. При этом операторы репродукции и кроссинговера делают поиск направленным. Широту поиска обеспечивает

то, что процесс ведется на множестве решений – популяции и используется оператор мутации.

В отличие от других методов оптимизации ГА оптимизируют различные области пространства решений одновременно и более приспособлены к нахождению новых областей с лучшими значениями целевой функции за счет объединения квазиоптимальных решений из разных популяций.

Каждый из функциональных блоков ГА рис.1.1 , как показано в разделе 3 (модификации и обобщения ГА), может быть реализован различными способами. Но сначала мы рассмотрим на простом примере основные моменты классического ГА,

1.2. Генетические операторы

Рассмотрим работу простого ГА на следующем примере из популярной монографии Голдберга [2]. Надо найти (для простоты) целочисленное значение x на отрезке от $[0,31]$, при котором функции $y = x^2$ принимает максимальное значение.

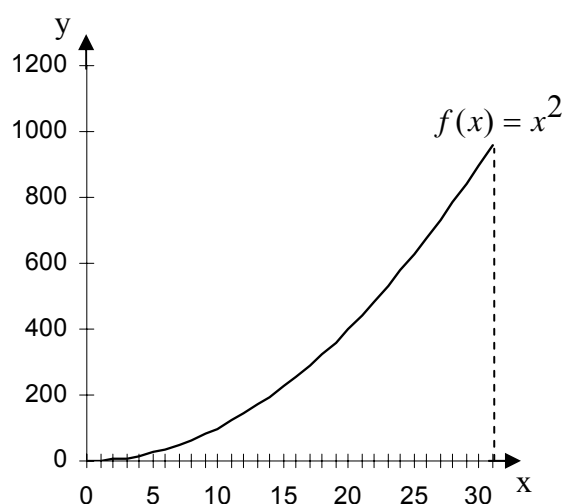


Рис.1.2. Пример функции

Начальный этап работы ГА для данного примера приведен в верхней таблице (репродукция) рис.1.3.

Репродукция

№ хромо- сомы	Начальная популяция особей	Десятичное значение x	Значение f(x)	$\frac{f(x_i)}{\sum f(x_j)}$	Среднее значение $\overline{f(x)}$	Максимальное значение $f_{\max}(x)$
1	01101	13	169	0,14	293	576
2	11000	24	576	0,49		
3	01000	8	64	0,06		
4	10011	19	361	0,31		



Кроссинговер

№ хромо- сомы	Популяция после репродукции	Пары хромосом для кроссинговера	Популяция после кроссинговера	Значение f(x)	Среднее значение $\overline{f(x)}$	Максимальное значение $f_{\max}(x)$
1	0 1 1 0 1	1-2	01100	144	439	729
2	1 1 0 0 0	1-2	11001	625		
3	1 1 0 0 0	3-4	11011	729		
4	1 0 0 1 1	3-4	10000	256		



Мутация

№ хромо- сомы	Популяция после кроссинговера	Новая популяция после мутации	Десятичное значение x	Значение f(x)	Среднее значение $\overline{f(x)}$	Максимальное значение $f_{\max}(x)$
1	01100	01100	12	144	496.5	961
2	11001	11001	25	625		
3	11011	11111	31	961		
4	10000	10000	16	256		

Рис.1.3. Эволюция популяции

Здесь особи начальной популяции (двоичные коды значений переменных x - столбец 2) сгенерированы случайным образом. Двоичный код значения x называется хромосомой (она представляет генотип). Популяция образует множество потенциальных решений данной проблемы. В третьем столбце представлены их десятичные значения (фенотип). Далее на этом примере проиллюстрируем работу трех основных генетических операторов.

Репродукция

Репродукция – это процесс, в котором хромосомы копируются в промежуточную популяцию для дальнейшего "размножения" согласно их значениям целевой (фитнес-) функции. При этом хромосомы с лучшими значениями целевой функции имеют большую вероятность попадания одного или более потомков в следующее поколение.

Очевидно, оператор репродукции (ОР) является искусственной версией естественной селекции – выживания сильнейших по Дарвину. Этот оператор представляется в алгоритмической форме различными способами (подробнее различные варианты ОР будут рассмотрены в разделе 4). Самый простой (и популярный) метод реализации ОР – построение колеса рулетки, в которой каждая хромосома имеет сектор, пропорциональный ее значению ЦФ. Для нашего примера "колесо рулетки" имеет следующий вид, представленный на рис.1.4.

Для селекции хромосом используется случайный поиск на основе колеса рулетки. При этом колесо рулетки вращается и после останова ее указатель определяет хромосому для селекции в промежуточную популяцию (родительский пул). Очевидно, что хромосома, которой соответствует больший сектор рулетки, имеет большую вероятность попасть в следующее поколение. В результате выполнения оператора репродукции формируется промежуточная популяция, хромосомы которой

будут использованы для построения поколения с помощью операторов скрещивания.

В нашем примере выбираем хромосомы для промежуточной популяции, вращая колесо рулетки 4 раза, что соответствует мощности начальной популяции. Величину $\frac{f(x_i)}{\sum f(x_j)}$ обозначим, как $P(x_i)$, тогда ожидаемое количество копий i -ой хромосомы определяется $M = P(x_i) * N$, N -мощность популяции. Число копий хромосомы, переходящих в следующее поколение, иногда определяется и так:

$$\tilde{M} = \frac{f(x_i)}{\bar{f}(x)},$$

где $\bar{f}(x)$ - среднее значение хромосомы в популяции.

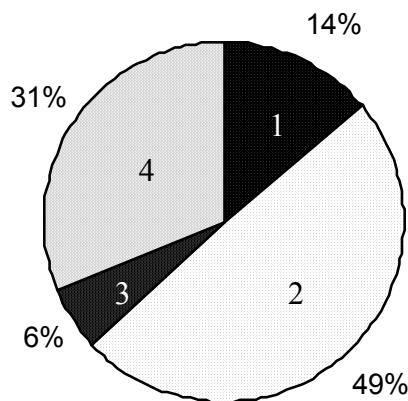


Рис.1.4. Колесо рулетки

Расчетные числа копий хромосом по приведенной формуле следующие: хромосома 1 - 0,56; хромосома 2 - 1,97; хромосома 3 - 0,22; хромосома 4 - 1,23. В результате, в промежуточную популяцию 1-я хромосома попадает в одном экземпляре, 2-я - в двух, 3-я - совсем не

попадает, 4-я – в одном экземпляре. Полученная промежуточная популяция является исходной для дальнейшего выполнения операторов кроссинговера и мутации.

Оператор кроссинговера(скрещивания)

Одноточечный или простой оператор кроссинговера (ОК) с заданной вероятностью P_c выполняется в 3 этапа:

1-й этап. Две хромосомы (родители)

$$\begin{array}{l} A = a_1 a_2 \dots a_k \quad a_{k+1} \dots a_L \\ B = b_1 b_2 \dots b_k \quad b_{k+1} \dots b_L \end{array}$$

Точка кроссинговера

выбираются случайно (или одним из методов, рассмотренных в разделе 4) из промежуточной популяции, сформированной при помощи оператора репродукции (ОР).

2-й этап. Случайно выбирается точка скрещивания - число k из диапазона $[1, 2 \dots n-1]$, где n – длина хромосомы (число бит в двоичном коде)

3-й этап. Две новых хромосомы A' , B' (потомки) формируются из A и B путем обмена подстрок после точки скрещивания:

$$\begin{array}{l} A' = a_1 a_2 \dots a_k \quad b_{k+1} \dots b_L \\ B' = b_1 b_2 \dots b_k \quad a_{k+1} \dots a_L \end{array}$$

Например, рассмотрим выполнение кроссинговера для хромосом 1 и 2 из промежуточной популяции:

$$A = 0 \ 1 \ 1 \ 0 \ 1$$

$$B = 1 \ 1 \ 0 \ 0 \ 0$$

$$1 \leq k \leq 4, \quad k=4$$

$$A' = 0 \ 1 \ 1 \ 0 \ 0$$

$$B' = 1 \ 1 \ 0 \ 0 \ 1$$

Следует отметить, что ОК выполняется с заданной вероятностью P_c (отобранные два родителя не обязательно производят потомков). Обычно величина $P_c \approx 0.5$.

Таким образом, операторы репродукции и скрещивания очень просты – они выполняют копирование особей и частичный обмен частей хромосом. Продолжение нашего примера представлено на рис.1.3 во второй таблице (кроссинговер).

Сравнение с предыдущей таблицей показывает, что в промежуточной популяции после скрещивания улучшились все показатели популяции (среднее и максимальное значения ЦФ).

Мутация

Далее согласно схеме классического ГА с заданной вероятностью P_m выполняется оператор мутации. Иногда этот оператор играет вторичную роль. Обычно вероятность мутации мала - $P_m \approx 0,001$.

Оператор мутации (ОМ) выполняется в 2 этапа:

1-й этап. В хромосоме $A = a_1 a_2 \dots a_L$ случайно выбирается k -ая позиция (бит) мутации ($1 \leq k \leq n$).

2-й этап. Производится инверсия значения гена в k -й позиции.

$$a'_k = \bar{a}_k.$$

Например, для хромосомы 11011 выбирается $k=3$ и после инверсии значения третьего бита получается новая хромосома – 11111. Продолжение нашего примера представлено в третьей таблице (мутация) рис.1.3. Таким образом, в результате применения генетических операторов найдено оптимальное решение $x=31$.

В данном случае, поскольку пример искусственно подобран, мы нашли оптимальное решение за одну итерацию. В общем случае ГА работает до тех пор, пока не будет достигнут критерий окончания процесса поиска и в последнем поколении определяется лучшая особь.

1.3. Представление вещественных решений в двоичной форме

В предыдущем примере мы рассматривали только целочисленные решения. Обобщим ГА на случай вещественных чисел на следующем примере, в котором для функции

$$f(x) = (1,85 - x) \cdot \cos(3,5x - 0,5),$$

представленной на рис.1.5 необходимо найти вещественное $x \in [-10, +10]$, которое максимизирует f , т.е. такое x_0 , для которого $f(x_0) \geq f(x)$ для всех $x \in [-10, +10]$.

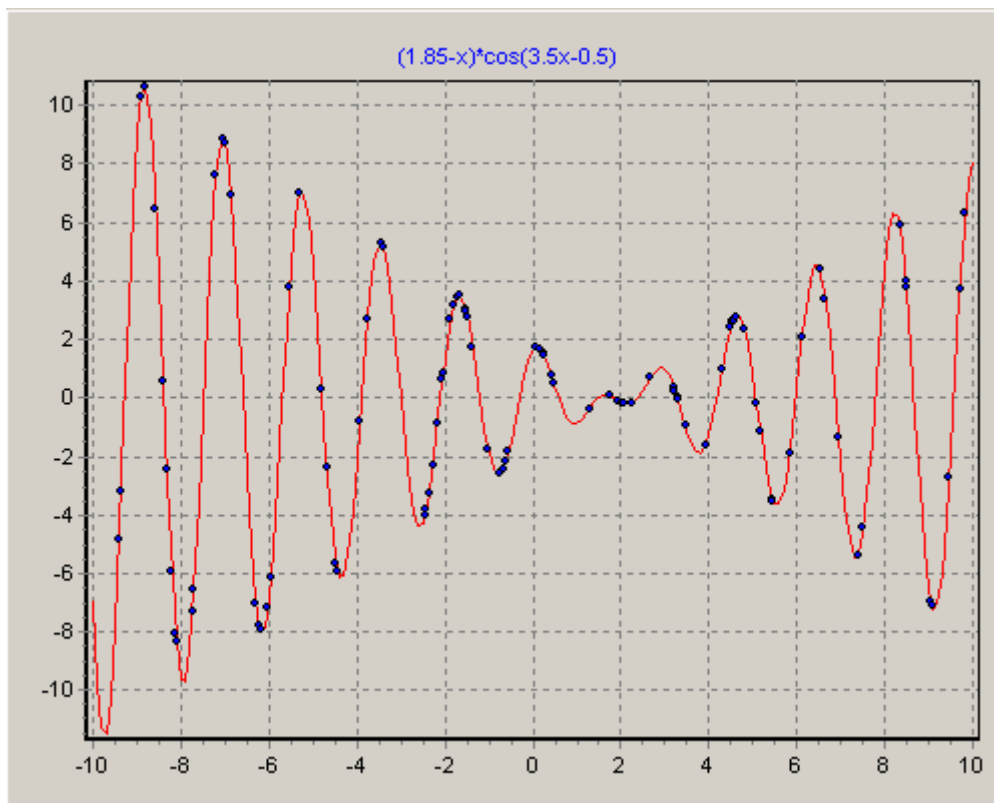


Рис.1.5. Пример функции с популяцией особей в начале эволюции

Нам необходимо построить ГА для решения этой задачи. Для представления вещественного решения (хромосомы) x будем использовать двоичный вектор [17], который применяется в классическом

простом ГА. Его длина зависит от требуемой точности решения, которую в данном случае положим 3 знака после запятой.

Поскольку отрезок области решения имеет длину 20, для достижения заданной точности отрезок $[a,c] = [-10,+10]$ должен быть разбит на равные части (маленькие отрезки), число которых должно быть не менее $20 \cdot 1000$. В качестве двоичного представления используем двоичный код номера (маленького) отрезка. Этот код позволяет определить соответствующее ему вещественное число, если известны границы области решения. Отсюда следует, что двоичный вектор для кодирования вещественного решения должен иметь 15 бит, поскольку

$$16384 = 2^{14} < 20000 \leq 2^{15} = 32768$$

Это позволяет разбить отрезок $[-10,+10]$ на 32768 частей и обеспечить необходимую точность. Отображение из двоичного представления $(b_{14}b_{13}...b_0)$ ($b_i \in \{0,1\}$) в вещественное число из отрезка $[a,c] = [-10,+10]$ выполняется в два шага.

1) Перевод двоичного числа в десятичное:

$$(<b_{14}b_{13}...b_0>)_2 = \left(\sum_{i=0}^{14} b_i 2^i \right)_{10} = X'$$

2) Вычисление соответствующего вещественного числа x :

$$x = a + x' \cdot \frac{(c-a)}{2^{15}-1} = -10 + x' \cdot \frac{20}{2^{15}-1}, \text{ где } (-10) \text{ левая граница области}$$

решения.

Естественно хромосомы

$$(0000000000000000) \text{ и } (1111111111111111)$$

представляют границы отрезка -10 и $+10$ соответственно.

Очевидно, при данном двоичном представлении вещественных чисел можно использовать классический простой ГА. На рис.1.5–рис.1.8 представлено расположение особей - потенциальных решений на различных этапах ГА в процессе поиска решения. На рисунке 1.5

показана начальная популяция потенциальных решений, которая равномерно покрывает область поиска решения. Далее явно видно, как постепенно с увеличением номера поколения особи "конденсируются" в окрестностях экстремумов и в конечном счете находится лучшее решение.

В заключение отметим, что ГА отличаются от других оптимизирующих и поисковых процедур следующими особенностями:

- 1) Работают не с параметрами, а с закодированным множеством параметров.
- 2) Осуществляет поиск из популяции точек, а не из единственной точки.
- 3) Использует целевую функцию непосредственно, а не ее приращение.
- 4) Использует не детерминированные, а вероятностные правила поиска решений.
- 5) Каждая новая популяция состоит из жизнеспособных особей (хромосом).
- 6) Каждая новая популяция лучше (в смысле целевой функции) предыдущей.
- 7) В процессе эволюции последующая популяция зависит только от предыдущей.

Цель ГА двоякая:

- 1) Абстрактно и формально объяснить адаптацию процессов в естественных системах.
- 2) Спроектировать искусственные программные и технические системы на основе механизмов, использующихся в естественных системах.

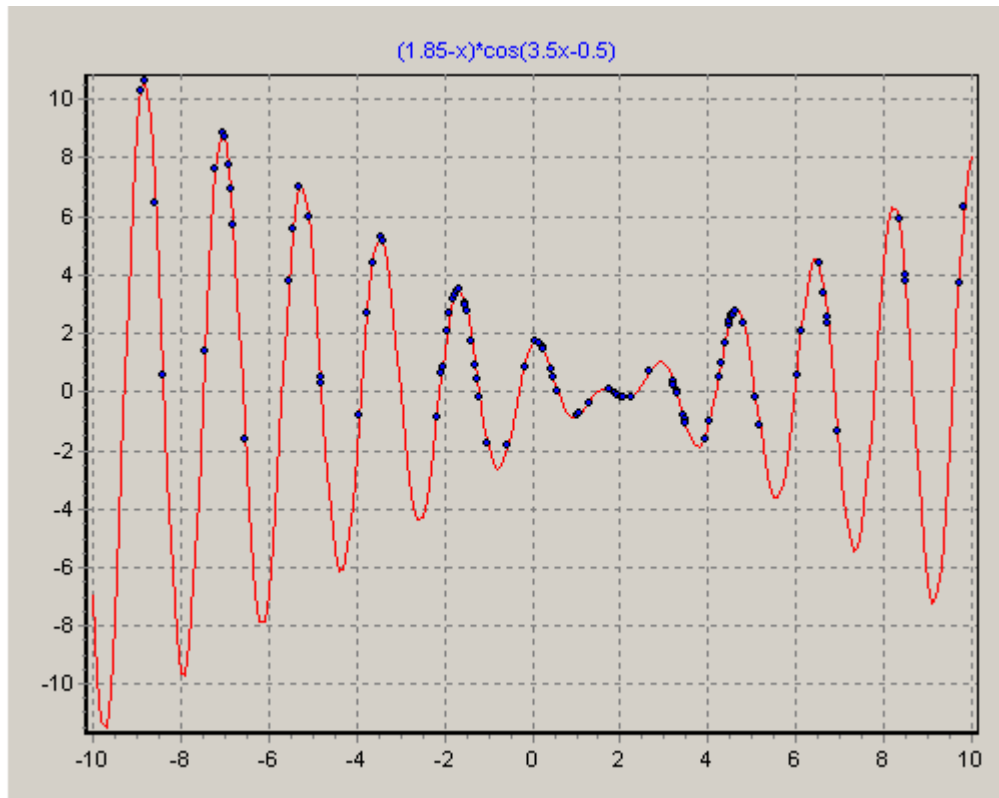


Рис.1.6. Начальная «конденсация» особей популяции в окрестностях экстремумов

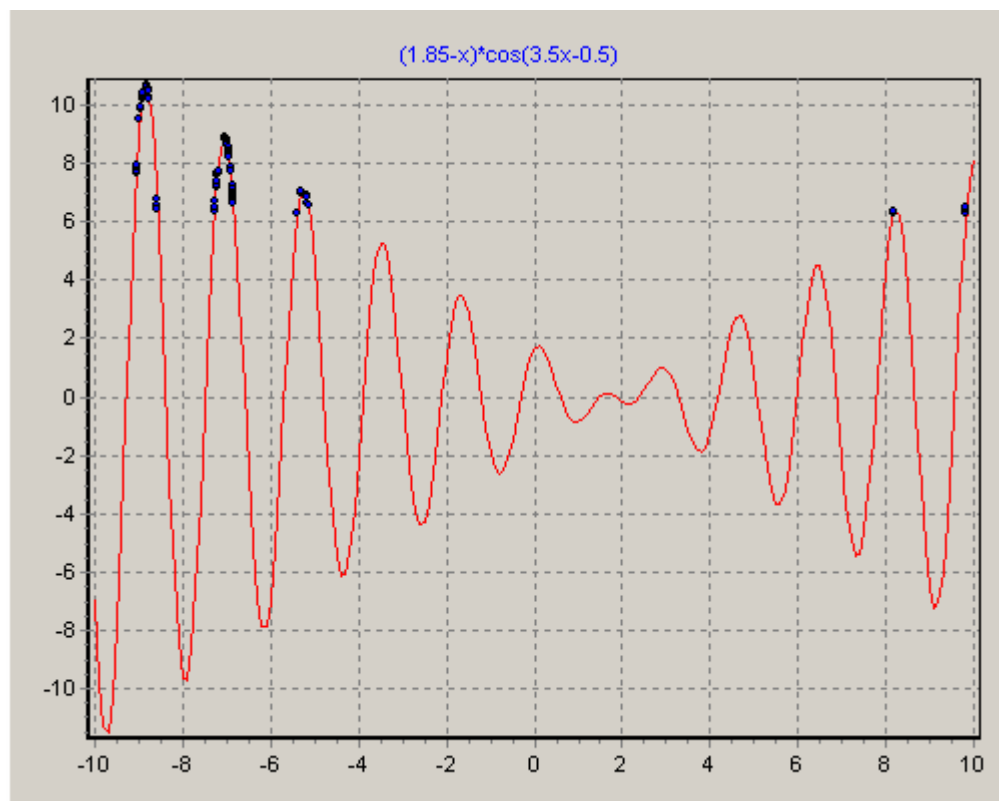


Рис.1.7. «Конденсация» особей в окрестностях экстремумов

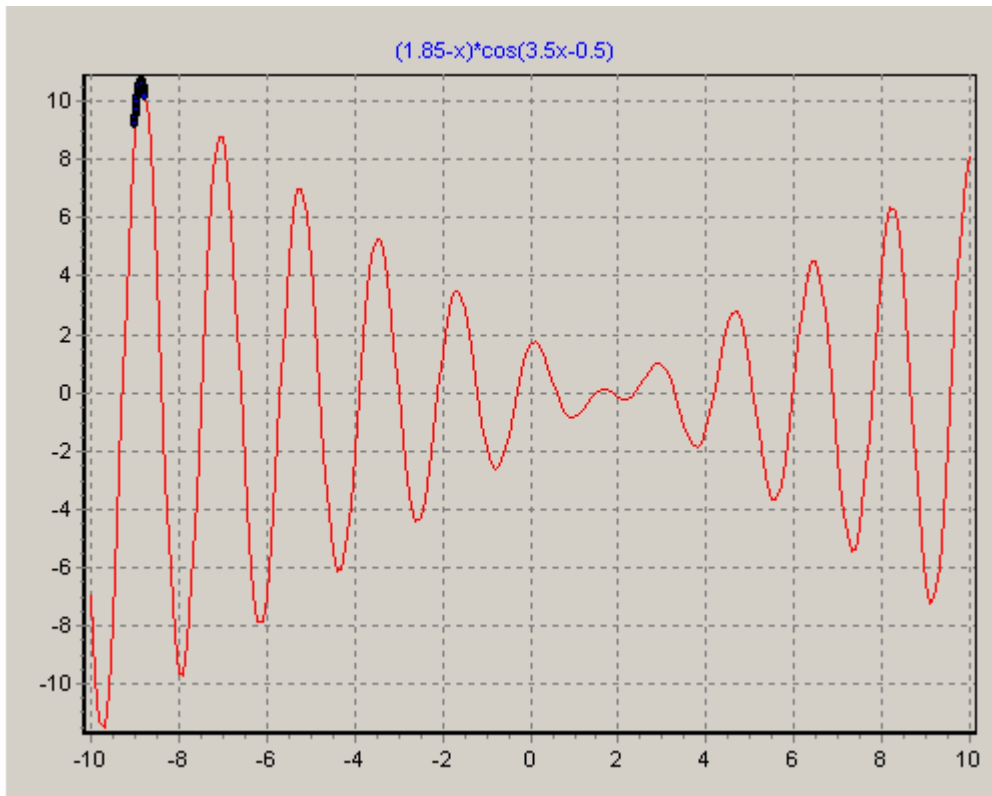


Рис.1.8. Положение особей популяции в конце эволюции

1.4. Использование кода Грея в ГА

Рассмотренное двоичное представление реального числа, имеет существенный недостаток: расстояние между реальными числами (на числовой оси) часто не соответствует расстоянию (по Хеммингу) между их двоичными представлениями. Поэтому желательно получить двоичное представление, где близкие расстояния между хромосомами (двоичными представлениями) соответствовали близким расстояниям в проблемной области (в данном случае расстоянию на числовой оси). Это можно сделать, например, с помощью кода Грея. В таблице 1.4 приведен для примера код Грея, для 4-х битовых слов.

Таблица 1.4

Двоичный код	Код Грея
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

Заметим, что в коде Грея соседние двоичные слова отличаются на 1 бит (расстояние по Хеммингу равно 1).

Рассмотрим алгоритмы преобразования двоичного числа $\bar{b} = \langle b_1, \dots, b_m \rangle$ в код Грея $\bar{g} = \langle g_1, \dots, g_m \rangle$ и наоборот.

Procedure Binary-to-Gray()

```
{
    g1=b1
    for k=2 to m do
        gk=bk-1 xor bk
    }
```

Procedure Gray-to-Binary

```
{
    value=g1
    b1=value
    for k=2 to m do
        begin
            if gk=1 then value=NOTvalue
            bk=value
        }
    end
```

Рис.1.9 Преобразование в код Грея

Здесь параметр m определяет разрядность двоичного числа. Существует и другая матричная (эквивалентная) процедура преобразования в код Грея. Например, для $m=4$ матрица

$$A = \begin{bmatrix} 1000 \\ 1100 \\ 0110 \\ 0011 \end{bmatrix} \text{ и } A^{-1} = \begin{bmatrix} 1000 \\ 1100 \\ 1110 \\ 1111 \end{bmatrix}$$

позволяют выполнять следующие преобразования:

$\bar{g} = A \cdot \bar{b}$ и $\bar{b} = A^{-1} \bar{g}$, где умножение матриц выполняются по mod 2.

Отметим, что код Грея прежде всего оправдывается при использовании операторов мутации.

1..5. Логарифмическое кодирование

Данный вид кодирования применяется для сокращения длины хромосом. При этом первый бит (a) кодовой последовательности используется для знака показательной функции, второй бит (b) – для знака степени этой функции, и остальные биты (str) представляют значение самой степени. Таким образом, двоичный код <a b str> представляет вещественное число $(-1)^b e^{(-1)^a [str]_{10}}$. Здесь $[str]_{10}$ означает десятичное число, представленное двоичным кодом str. Например, двоичный код <10110> представляет вещественное число $r = (-1)^0 e^{(-1)^1 [110]_{10}} = -e^6 = 0,002478752$. Следует отметить, что при этом кодировании пять битов позволяет кодировать вещественные числа из интервала $[-e^7, e^7]$, что значительно больше, чем позволяет, например, метод кодирования, представленный в разделе 1.3.

1.6. Основная терминология в генетических алгоритмах

Поскольку, как отмечалось выше, в ГА используются некоторые термины, заимствованные из генетики, то полезно привести их и дать техническую интерпретацию.

Ген – элементарный код в хромосоме s_i , называемый также знаком, или детектором (в классическом ГА $s_i=0,1$).

Хромосома – упорядоченная последовательность генов в виде закодированной структуры данных $S=(s_1, s_2, \dots, s_n)$, определяющая решение (в простейшем случае двоичная последовательность –стринг, где $s_i=0,1$).

Локус – местоположение (позиция, номер бита) данного гена в хромосоме.

Аллель – значение, которое принимает данный ген (например, $\{0,1\}$).

Особь – одно потенциальное решение проблемы (представляемое хромосомой).

Популяция – множество особей – потенциальных решений, которые представляются хромосомами.

Поклоение – текущая популяция ГА (для текущей итерации алгоритма).

Генотип – набор хромосом данной особи (особями популяции могут быть генотипы либо отдельные хромосомы).

Генофонд – множество всех возможных генотипов.

Фенотип – набор значений, соответствующий данному генотипу, - декодированное множество параметров или структура данной задачи (например, фенотип - десятичное значение x его двоичного кода - генотипа);

Размер (мощность) популяции N – число особей (решений) в популяции.

Число поколений (генераций) – количество итераций, в течение которых производится генетический поиск.

Селекция – комплекс правил, моделирующих выживание особей на основе их значений ЦФ.

Эволюция популяции – это чередование поколений, в которых хромосомы изменяют свои признаки так, чтобы каждая новая популяция наилучшим образом приспособлялась к внешней среде.

Фитнесс-функция (полезности) – важнейшее понятие, определяющее меру приспособленности данной особи в популяции. В задачах оптимизации часто представляется целевой функцией или определяет меру близости к оптимальному решению. В обучении может принимать вид функции погрешности (ошибки). На каждой итерации ГА приспособленность каждой особи популяции оценивается с помощью фитнес-функции.

1.7 Контрольные вопросы к разделу 1

1. Какие “источники” ГА?
2. Какие генетические операторы используются в ГА?
3. Какую роль в ГА играет оператор репродукции (ОР)?
4. Опишите реализацию ОР в виде колеса рулетки и приведите пример его работы.
5. Придумайте другую реализацию ОР.
6. Опишите 1-точечный оператор кроссинговера (ОК) и приведите пример его работы.
7. Придумайте другую реализацию ОК.
8. Какую роль играет оператор мутации (ОМ) ?
9. Опишите ОМ и приведите пример его работы.
10. Придумайте другую реализацию ОМ.
11. Выполните программную реализацию простого ГА на одном из языков программирования для поиска максимума функции $f(x) = 3x^3 + 2$ на отрезке $[-5, 5]$.

12. Какие основные параметры ГА ?
13. Исследуйте зависимость работы (скорость сходимости) ПГА от мощности популяции N (варьируя ее значение).
14. Исследуйте зависимость работы ПГА от значения вероятности ОК P_c .
15. Исследуйте зависимость работы ПГА от значения вероятности ОМ P_m .

2. ТЕОРИЯ СХЕМ И МОДЕЛИ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ

2.1. Теория схем

Теоретические основы ГА представляют двоичное стринговое представление решений (хромосом) и понятие схемы (schema) или шаблона. Это понятие было введено Холландом для определения множества хромосом, которые обладают некоторыми общими свойствами, то есть в каком то смысле подобны друг другу. Термин «схема» согласно Холланду [1] есть шаблон, описывающий подмножество стрингов, имеющих одинаковые значения в некоторых позициях. Для этого вводится новый троичный алфавит $\{0, 1, *\}$, где $*$ означает неопределенное значение (0 или 1, то неизвестно что именно). Например, схема $(0*0001)$ соответствует двум стрингам $\{000001, 010001\}$, а $(*0110*)$ описывает подмножество из 4-х стрингов $\{00100, 101100, 001101, 101101\}$. Очевидно, схема с m неопределенными позициями “*” представляет 2^m стрингов. Для стрингов длины n , существует 3^n схем (возможны 3 символа $\{0, 1, *\}$ в каждой из n позиций).

В простом ГА основная идея заключается в объединении хромосом со значениями ЦФ выше среднего. Например, пусть 1 в хромосоме соответствует наличию признака, способствующего выживанию (значение ЦФ больше среднего). Допустим, что имеются подстринги вида $11***$ и $**111$. Тогда, применяя к ним ОК можно получить хромосому 11111 с признаками, способствующим наилучшим значениям фитнес функции. Рассмотрим также полезное понятие строительного блока. Например, в шаблоне $****1$ строительным блоком является элемент 1. В шаблоне $10***$ строительным блоком будет составной элемент 10. Очевидно, вид строительных блоков должен выбираться из знаний о решаемой задаче и

отражать полезные свойства, чтобы далее из строительных блоков как из “кирпичиков” собрать “здание”, т.е. решение с лучшей ЦФ. Желательно, чтобы в ГА выражались условия, которые разрешают разрыв строительных блоков только в крайних случаях, указанных пользователем.

Не все схемы являются одинаковыми. Некоторые более специфичны, чем другие. Для количественной оценки вводятся 2 характеристики:

- 1) порядок схемы $O(H)$, который определяется числом фиксированных позиций (не равных *). Например, для $H_1 = 0*1***$ ее порядок $O(H_1) = 2$.
- 2) Определенная длина $L(H)$ – расстояние между первой и последней определенной (не равной *) позицией. Например для шаблона $H_2 = 0111*1**$
 $L(H_2) = 6 - 1 = 5$. А для $H_3 = 0****$ $L(H_3) = 1 - 1 = 0$.

Обозначим через $m(H,t)$ – число стрингов, содержащихся в популяции $A(t)$ (t – шаг итерации или время), которые отображаются (покрываются) схемой H .

Пусть $f(H)$ означает среднее значение фитнес-функции для хромосом, покрываемых данной схемой H , а

$$\overline{f(x)} = \frac{\sum_{j=1}^N f(x_j)}{N} \quad \text{– среднее значение фитнес-функции для всей}$$

популяции (N – размер популяции). Очевидно, что для схемы, которая представляет хорошее решение, было бы желательным, чтобы количество хромосом, соответствующих этой схеме, возрастало в процессе эволюции с ростом номера поколения. Далее мы рассмотрим, как ведет себя $m(H,t)$ при выполнении основных операторов ГА.

2.2. Фундаментальная теорема ГА

Влияние репродукции

Напомним, что в процессе репродукции хромосомы копируются в промежуточную популяцию согласно их значениям фитнес-функции – хромосома x_i со значением $f(x_i)$ выбирается с вероятностью $P(x_i) = \frac{f(x_i)}{\sum f(x_j)}$. После репродукции мы ожидаем на следующем шаге получить $m(H, t+1)$ двоичных стрингов, отображаемой схемой H . Известно, что

$$m(H, t + 1) = m(H, t) * N * f(H) / \sum f(x_j) \quad (2.1)$$

Это обусловлено тем, что:

- (1) в среднем вероятность выбора стрингов, покрываемых схемой H , определяется $f(H) / \sum f(x_j)$,
- (2) число стрингов, представляемых H , равно $m(H, t)$,
- 3) число стрингов в популяции равно N .

Мы можем переписать эту формулу с учетом обозначения

$$\overline{f(x)} = \frac{\sum_{j=1}^N f(x_j)}{N}$$

и получим следующее выражение для числа особей, покрываемых схемой в промежуточной популяции

$$m(H, t + 1) = m(H, t) * f(H) / \overline{f(x)}. \quad (2.2)$$

Другими словами схемы “растут” как отношение среднего значения фитнес-функции схемы к среднему значению фитнес-функции популяции. Схема со значением фитнес-функции выше средней в

популяции имеет больше возможностей для копирования и наоборот. Правило репродукции Холланда гласит: “схема со значением фитнес-функции выше среднего живёт и размножается, а со значением фитнес-функции ниже среднего умирает”.

Предположим, что схема H имеет значение выше среднего фитнес-функции на величину $c * \bar{f}$, где c – константа. Тогда последнее выражение (2.2) можно модифицировать:

$$m(H, t + 1) = m(H, t) * (\bar{f} + c * \bar{f}) / \bar{f} = (1 + c) m(H, t).$$

Начиная с $t = 0$ и предполагая, что c – величина постоянная, получаем следующее выражение числа особей промежуточной популяции, покрываемых схемой,

$$m(H, t) = m(H, 0) * (1 + c)^t. \quad (2.3)$$

Это равенство описывает геометрическую прогрессию. Очевидно, что при $c > 0$ схема “размножается”, а при $c < 0$ схема умирает. Далее рассмотрим влияние оператора кроссинговера на число особей в популяции, покрываемых схемой.

Влияние кроссинговера

Рассмотрим конкретный стринг длины $n = 7$ $A = 011|1000$ и две схемы, представляющие этот стринг:

$$H_1 = *1*|***0, H_2 = ***|10**.$$

Здесь символ ‘|’ – , как обычно, обозначает точку кроссинговера ($k=3$).

Очевидно, что схема H_1 после кроссинговера с точкой $k=3$, скорее всего, будет уничтожена потому, что ‘1’ в позиции 2 и ‘0’ в позиции 7 попадут в разные новые стринги после кроссинговера. С другой стороны, ясно, что схема H_2 будет выживать, так как “10” в позициях 4,5 будут содержаться вместе в одном новом стринге. Хотя мы взяли точку скрещивания ОК случайно, ясно, что схема H_1 менее приспособлена к выживанию, чем схема H_2 . Если точка скрещивания ОК выбирается

случайно среди $n-1 = 7-1 = 6$ возможных позиций, то ясно, что схема H_1 разрушается с вероятностью

$$P(d) = L(H_1) / (n-1) = 5/6.$$

Очевидно, что эта же схема выживает с вероятностью

$$P(S) = 1 - P(d) = 1/6.$$

Аналогично, схема H_2 имеет $L(H_2) = 1$ и вероятность её уничтожения $P(d) = 1/6$, а вероятность её выживания схемы после применения ОК $P(S) = 5/6$. Очевидно, что нижняя граница вероятности выживания схемы после применения ОК может быть вычислена для любой схемы. Так как схема выживает, когда точка ОК попадает вне “определенной длины”, вероятность выживания для простого ОК определяется по формуле

$$P_s(OK) = 1 - L(H)/(n-1).$$

Если ОК выполняется посредством случайного выбора, например, с вероятностью P_c , то вероятность выживания схемы определяется так:

$$P(S) \geq 1 - P_c * L(H)/(n-1).$$

Очевидно, что это выражение уменьшается при $P_c \rightarrow 1$. Теперь мы можем асимптотически оценить совместный эффект операторов репродукции и кроссинговера. При независимости выполнения ОР и ОК можно получить следующее выражение:

$$m(H, t+1) \geq m(H, t) \frac{f(H)}{\bar{f}} [1 - P_c \frac{L(H)}{n-1}]. \quad (2.4)$$

Таким образом, число схем N в новой популяции зависит от двух факторов:

- 1) значение фитнес-функции схемы выше или ниже ЦФ популяции;
- 2) схема имеет “длинную” или “короткую” $L(H)$ (определенную длину).

Видно, что схемы со значением ЦФ выше средней и короткой длиной $L(H)$ имеют возможность экспоненциального роста в новой популяции. Далее рассмотрим влияние оператора мутации на число особей в популяции, покрываемых схемой.

Влияние мутации

Напомним, что оператор мутации ОМ есть случайное изменение элемента в строке с вероятностью P_m . Очевидно, что для того, чтобы схема H выжила, все определенные позиции должны выжить. Поскольку один ген выживает с вероятностью $(1-P_m)$, то данная схема выживает, когда каждая из $O(H)$ фиксированных позиций схемы выживает. Умножая вероятность выживания $(1-P_m)^{O(H)}$ раз, получим, что вероятность выживания при ОМ равна $(1-p_m)^{O(H)}$. Для малых величин $P_m \ll 1$ это выражение может быть аппроксимировано

$$P_s(OM) = 1 - O(H) \cdot P_m.$$

Из этого следует, что схема H дает ожидаемое число особей в следующей популяции после выполнения всех генетических операторов ОР, ОК и ОМ согласно следующей формуле

$$m(H, t+1) > m(H, t) \cdot \frac{f(H)}{\bar{f}} \left[1 - P_m \frac{L(H)}{1-n} - O(H) \cdot P_m \right] \quad (2.5)$$

Этот важный результат известен как **фундаментальная теорема ГА**.

Учитывая (2.3) ее можно также сформулировать следующим образом.

Теорема 2.1. Схемы малого порядка, малой определенной длины и со значением фитнес-функции выше средней формируют показательное растущее число своих представителей в последующих поколениях генетического алгоритма.

На основании приведенных результатов была выдвинута **гипотеза о строительных блоках**.

Гипотеза 2.1. Генетический алгоритм стремится достичь близкого к оптимальному результату за счет комбинирования хороших схем (со значением фитнес-функции выше средней и малого порядка и определенной длины). Такие схемы называются строительными блоками. В соответствии с этим оптимальное решение строится путем объединения наилучших из полученных на текущий момент частичных решений. Оператор скрещивания на двоичных строингах не слишком часто уничтожает схемы малой определенной длины, однако ликвидирует схемы большой длины. Однако не смотря на губительность операторов скрещивания и мутации для схем большого порядка и определенной длины, число обрабатываемых схем настолько велико, что даже относительно малом числе особей в популяции генетический алгоритм дает неплохие результаты.

Несмотря на то, что для доказательства этой гипотезы были предприняты значительные усилия, строгого доказательства получено не было, и в большинстве нетривиальных приложений опираются на эмпирические результаты.

Далее проиллюстрируем полученные результаты. Вернемся к примеру Голдберга определения $\max f(x)=x^2$. В дополнение к имеющимся таблицам рис 1.3 пусть имеется 3 конкретные схемы $H_1=1*****$, $H_2=*10**$ и $H_3=1***0$, описанные в табл.2.1.

Рассмотрим сначала схему H_1 . При репродукции строинги копируются с вероятностью, определенной согласно величине их фитнес-функции. Из табл. 2.1 видно, что строинги 2,4 покрываются схемой H_1 . После выполнения репродукции мы видим в табл. 2.2 (строка H_1), что получены три копии и строинг 3 также вошел в популяцию.

Таблица 2.1

Схема	Перед репродукцией	Представители стрингов	Средняя ЦФ схемы $f(H)$
H1	1****	2,4	469
H2	*10**	2,3	320
H3	1***0	2	576

Таблица 2.2

Схема	После репродукции			После всех операторов		
	Ожидаемое число стрингов	Действительное число стрингов	Представители стрингов	Ожидаемое число стрингов	Действительное число стрингов	Представители стрингов
H1	3,20	3	2, 3, 4	3,20	3	2, 3, 4
H2	2,18	2	2, 3	1,64	2	2, 3
H3	1,97	2	2, 3	0,0	1	4

Проверим, соответствует ли это число фундаментальной теореме схем. Согласно теории мы ожидаем $m \cdot f(H) / \bar{f}$ копий. Вычисляя среднее значение ЦФ $f(H_1)$ получаем $\bar{f}(H_1) = (576 + 361) / 2 = 468,5$. Разделив это число на среднее значение ЦФ популяции $\bar{f} = 293$ и умножив его на число стрингов, покрываемых H_1 на шаге t $m(H_1, t) = 2$, получаем ожидаемое число стрингов, покрываемых H_1 на шаге $t+1$, т.е. $m(H_1, t+1) = 2 \cdot 468,5 / 293 = 3,20$.

Сравнивая это число с реальным числом копий 3, видим, что округление рассчитанного значения копий 3,2 дает их реальное число 3. Дальнейший анализ показывает, что в данном случае ОК не оказывает влияние на число стрингов, покрываемых схемой, так как определенная длина $L(H_1)=0$ предотвращает разрыв схемы. Далее при мутации с вероятностью $P_{OM}=0,001$ мы должны ожидать уменьшение числа стрингов на $m \cdot O(H_1) \cdot P(OM) = 3 \cdot 1 \cdot 0,001 = 0,003$, что практически не оказывает влияния на окончательный результат. В итоге для схемы H_1 получаем ожидаемое увеличение числа особей до 3, что соответствует формуле 2.2.

Рассмотрим теперь схемы H_2 и H_3 с двумя фиксированными позициями. Схема H_2 имеет также два представителя в начальной популяции (2,3) и две копии в следующей промежуточной популяции. Вычисляем $m(H_2) = 2 \cdot 320 / 293 = 2,18$. Здесь $f(H_2) = 320$ – среднее значение ЦФ схемы, а $\bar{f} = 293$ – среднее значение ЦФ популяции. Для H_3 получаем только одно стринговое представление и $m(H_3) = 1 \cdot 576 / 293 = 1,97$. Здесь 576 – среднее значение ЦФ схемы.

Заметим, что для конкретной схемы H_2 две копии – это хороший результат и он может случиться только 1 раз из 4-х возможных случаев ($n-1=5-1$)=4. Схема $H_2 = *|1|0|*|*$ - имеет реальную возможность дать новую копию. В результате, схема H_2 выживает с высокой вероятностью. Действительно, число стрингов, покрываемых H_2 , равно $m(H_2, t+1) = 2,18 \cdot 0,75 = 1,64 \approx 2$.

Для схемы H_3 при высокой определенной длине $L(H_3)=4$ оператор ОК обычно разрушает эту схему. Поэтому ожидаемое число $m(H_3, t+1)=0$ в соответствии с формулой (2.4).

В соответствие с полученными результатами важнейшим аспектом является кодирование особей, которое должно обеспечить построение схем малого порядка малой определенной длины и со значениями

фитнесс-функции выше среднего. Следующий простой пример показывает важность построения генома с учетом теорем схем.

Рассмотрим поиск максимума (для простоты при целочисленных значениях x и y) функции $f(x, y) = x^2 - y + 17$. Можно показать, что максимум $f=66$ достигается при значениях $x=111_2=7_{10}$ и $y=0000_2=0_{10}$. Пусть $x=x_2x_1x_0$ и $y=y_3y_2y_1y_0$, где $x_i, y_i \in \{0,1\}$. Рассмотрим различные строение геномов. В первом случае пусть генотип представляет $y_3x_2y_2x_1y_1x_0y_0$, во втором генотип определим как $y_3y_2y_1y_0x_2x_1x_0$. Отметим, что в первом случае старшие разряды x и y близко расположены в генотипе, а во втором варианте наоборот – достаточно далеко. Поскольку желательна короткая определенная длина, генетический алгоритм с геномом, в котором старшие разряды расположены близко, должен быть лучше по сравнению с геномом, где эти разряды стоят далеко друг от друга. Для первого случая $\frac{L(H)}{1-n} = \frac{1}{6}$ для схемы $H_1=01*****$, где содержатся значения, наиболее важные для искомого решения. Для второго случая схема при тех же значениях имеет вид $H_2=0*****1$, для которой $\frac{L(H)}{1-n} = 1$ и поэтому необходимые значения старших битов x и y , скорее всего, при выполнении оператора кроссинговера будут «разорваны». Поэтому первый вариант генома предпочтительней.

2.3. Модель динамической системы для простого генетического алгоритма

Теория схем позволяет прогнозировать ожидаемые изменения в числе особей, покрываемых схемами, которые происходят при смене поколений ГА, но при этом ничего не известно о составе популяции, скорости сходимости популяции, распределении значений фитнес-

функции в популяции и т.п. Поэтому были разработаны более точные модели ГА, некоторые из которых изложены ниже в интерпретации [18]. Первой рассмотрим модель M.Vose, G.Liepins [19], которая предложена для следующей модификации простого ГА.

1. Вычислить значение фитнес-функции для каждой особи x в популяции.
2. Выбрать (с заменой) два родителя из текущей популяции с вероятностью, пропорциональной относительным значениям фитнес-функции в популяции.
3. Выполнить односточечный кроссинговер для двух родителей (со случайно выбранной точкой) с вероятностью p_c и образованием двух потомков. (Если кроссинговер не выполняется, то потомки являются точными копиями родителей). Выбрать случайным образом одного из потомков и отбросить второго.
4. Выполнить мутацию в потомке каждого бита с вероятностью p_m и поместить его в новую популяцию.
5. Переход на шаг 2, если новая популяция полна (достигнуто заданное число особей).
6. Переход на шаг 1.

Данная модификация отличается от рассмотренных ранее, прежде всего тем, что на шаге 3 «выживает» только один потомок. Таким образом, для популяции мощностью N выполняется N рекомбинаций. Такая модификация облегчает формализацию при разработке модели.

В рассматриваемой модели каждая n -разрядная хромосома в пространстве решений представляется целым числом i ($0 \leq i \leq 2^n - 1$). Например, для $l=8$ хромосома 00000111 представляется целым числом 7. Популяция поколения t представляется двумя вещественными векторами $\bar{p}(t)$ и $\bar{s}(t)$, каждый из которых имеет длину 2^n . Компонента

$p_i(t)$ вектора $\bar{p}(t)$ определяется как часть (доля) популяции поколения t , состоящая из строки i . Компонента $s_i(t)$ вектора $\bar{s}(t)$ равна вероятности выбора строки i в качестве родителя на шаге 2 простого ГА. Например, при разрядности хромосомы $l=2$ и популяции, которая состоит из строк 01, 10 и двух копий строки 11 имеем:

$\bar{p}(t) = (0; 0,25; 0,25; 0,5)$. Если мы определим фитнес-функцию равной числу единиц в строке, то получим $\bar{s}(t) = (0; 0,1667; 0,1667; 0,6667)$. Таким образом, вектор $\bar{p}(t)$ полностью определяет состав популяции поколения t , а $\bar{s}(t)$ отражает вероятность выбора особей при заданной фитнес-функции. Очевидно, что эти два вектора связаны фитнес-функцией. Пусть F – двумерная матрица, такая что $F_{ij} = 0$ для $i \neq j$ и $F_{ij} = f(i)$. Таким образом F является диагональной матрицей, у которой элемент (i,i) равен значению фитнес-функции строки i . Отсюда следует, что при пропорциональном отборе родителей имеем

$$\bar{s}(t) = \frac{F\bar{p}(t)}{\sum_{j=0}^{2^n-1} F_{ij} p_j(t)} \quad (2.6)$$

Таким образом, при заданных $\bar{p}(t)$ и F мы можем легко найти $\bar{s}(t)$ и наоборот. В дальнейшем мы будем, в основном, оперировать с $\bar{s}(t)$. Далее необходимо найти генетический оператор G , такой, что применение G к $\bar{s}(t)$ позволяет моделировать эффекты запуска ГА поколения t и формировать популяцию поколения $t+1$, т.е.

$$\bar{s}(t+1) = G\bar{s}(t). \quad (2.7)$$

Тогда, начиная с нулевого поколения $\bar{s}(0)$, итеративно применяя оператор G , мы можем получить точное описание ожидаемого поведения ГА.

Сначала для простоты предположим, что в ГА используется только оператор отбора родителей (кроссинговер и мутация отсутствуют). Обозначим $E(x)$ математическое ожидание от x . Тогда, поскольку $s_i(t)$ равна вероятности выбора строки i , $E(\bar{p}(t)) = \bar{s}(t)$.

Пусть $\bar{x} \sim \bar{y}$ означает, что \bar{x} и \bar{y} отличаются только скалярным множителем. Тогда из (2.5) следует, что $\bar{s}(t+1) \sim F\bar{s}(t)$, что означает $E(\bar{s}(t+1)) \sim F\bar{s}(t)$.

Таким образом, мы получили соотношение типа (2.7) (которое хотели), где оператор $G=F$ в случае использования только генетического оператора отбора. Полученные результаты дают только математическое ожидание значений. В популяции с конечным числом особей ошибки выборочного обследования будут вызывать отклонения от ожидаемых значений. В пределе при бесконечной популяции ожидаемые значения являются точными.

Далее учтем влияние генетических операторов кроссинговера и мутации. Для этого определим оператор G в виде композиции фитнес-матрицы F и оператора рекомбинации M , который моделирует эффекты кроссинговера и мутации (отметим, что здесь рекомбинация включает в себя как кроссинговер, так и мутацию). Одним из возможных способов определения рекомбинации M является построение матрицы $r_{ik}(k)$ – вероятности того, что строка k производится в результате рекомбинации между строками i и j . Если матрица $r_{ik}(k)$ известна, то можно вычислить

$$E(p_k(t+1)) = \sum_{i,j} s_i(t) s_j(t) r_{ij}(k).$$

Определение матриц $r_{ik}(k)$ и M является нетривиальной задачей. В [19] авторы сначала ввели упрощенную матрицу M , элементы которой M_{ij} определяют вероятность $r_{ik}(0)$ – получения строки из всех нулей в результате рекомбинации строк i и j (при выбранных i и j).

Если вероятность $r_{ik}(0)$ известна, то она может быть использована и для определения общего случая $r_{ik}(k)$.

Выражение для $r_{ik}(0)$ равно сумме двух слагаемых: 1) вероятности того, что кроссинговер не выполняется между строками i и j и отобранный потомок (i и j) мутирует в нулевую строку; 2) вероятности того, что кроссинговер выполняется и отобранный потомок мутирует в нулевую строку.

Если строки i и j отобраны в качестве родителей, вероятность выполнения кроссинговера между ними равна p_c , соответственно вероятность невыполнения кроссинговера – $(1 - p_c)$. Аналогично, вероятность мутации каждого бита отобранного потомка равна p_m и вероятность того, что бит не мутирует – $(1 - p_m)$. Обозначим через $|i|$ число единиц в строке длины l . Тогда вероятность мутации строки i в нулевую строку равна вероятности того, что все $|i|$ единиц мутируют, а оставшиеся $(n - |i|)$ бит останутся неизменными – $p_m^{|i|}(1 - p_m)^{n - |i|}$.

Таким образом, первое слагаемое в выражении $r_{ik}(0)$ определяется следующим образом

$$\frac{1}{2}(1 - p_c)[p_m^{|i|}(1 - p_m)^{n - |i|} + p_m^{|j|}(1 - p_m)^{n - |j|}].$$

Напомним, что в данной модели после рекомбинации только один потомок отбирается в следующее поколение. Множитель $\frac{1}{2}$ показывает, что каждый из двух потомков имеет одинаковую вероятность выбора в следующее поколение.

Рассмотрим второе слагаемое. Пусть h и k обозначают потомков, которые образовались в результате кроссинговера в точке скрещивания s . Отметим, что всего возможно $(n-1)$ точек кроссинговера – поэтому вероятность выбора точки скрещивания s равна $1/(n-1)$. В результате второе слагаемое можно представить таким образом

$$\frac{1}{2} \frac{p_c}{n-1} \sum_{c=1}^{n-1} [p_m^{|h|} (1-p_m)^{n-|h|} + p_m^{|k|} (1-p_m)^{n-|k|}] .$$

Как и ранее, множитель $\frac{1}{2}$ показывает, что в следующее поколение отбирается только один потомок. В полученном соотношении осталось найти выражения для $|h|$ и $|k|$. Пусть i_1 обозначает подстроку i , состоящую из $(1-c)$ бит левее точки кроссинговера - c . Аналогично, через i_2 обозначим подстроку i , которая состоит из битов правее точки скрещивания c . Таким же образом для строки j получаем j_1 и j_2 , как показано на рис. 2.1.

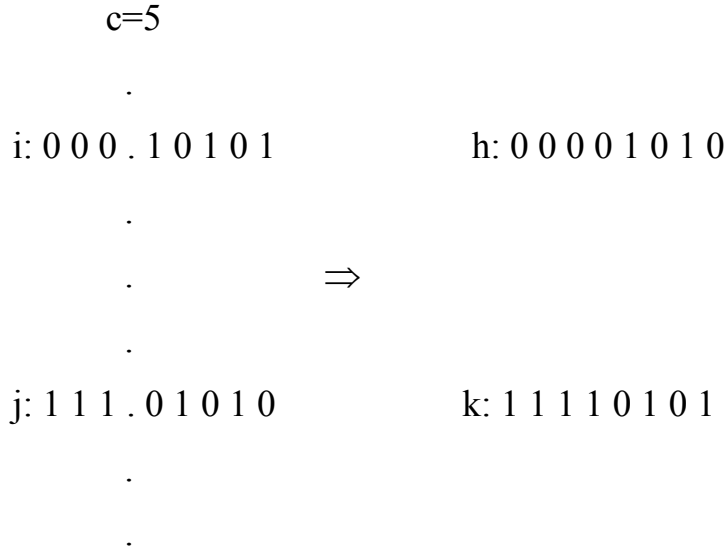


Рис.2.1. Иллюстрация построения c , i_1 , i_2 , j_1 и j_2 .

Можно найти выражения для $|h| = |i| - |i_2| + |j_2|$ и $|k| = |j| - |j_2| + |i_2|$. Заметим, что $|i_2| = \left| (2^c - 1) \wedge i \right|$, где \wedge означает поразрядное логическое И. Так как $(2^c - 1)$ представляет строку с $(n-c)$ нулями, за которыми следуют с единичных бит, выражение $\left| (2^c - 1) \wedge i \right|$ дает число единиц в правой подстроке i после точки кроссинговера. Аналогично $|j_2| = \left| (2^c - 1) \wedge j \right|$.

$$\text{Обозначим } \Delta_{i,j,c} = |i_2| - |j_2| = \left| (2^c - 1) \wedge i \right| - \left| (2^c - 1) \wedge j \right|,$$

$$\text{тогда } |h| = |i| - \Delta_{i,j,c} \text{ и } |k| = |j| + \Delta_{i,j,c} .$$

Теперь можно записать полное выражение для $r_{ij}(0)$. Для простоты пусть $\eta = p_m / (1 - p_m)$. Тогда после некоторых преобразований получаем следующее выражение

$$r_{ij}(0) = \frac{(1-p_m)^n}{2} \left[\eta^{|i|} \left(1 - p_c + \frac{p_c}{l-1} \sum_{c=1}^{n-1} \eta^{-\Delta_{i,j,c}} \right) + \eta^{|j|} \left(1 - p_c + \frac{p_c}{l-1} \sum_{c=1}^{n-1} \eta^{\Delta_{i,j,c}} \right) \right].$$

Полученные результаты демонстрируют методы анализа проблем поведения ГА на данном уровне.

Пусть $G(\bar{x}) = F \circ M(\bar{x})$ для вектора \bar{x} , где \circ обозначает оператор композиции. Тогда в пределе для s бесконечным числом особей имеет место $G(\bar{s}(t)) \sim \bar{s}(t+1)$. Определим $G_p(\bar{x}) = M(F\bar{x}/|F\bar{x}|)$, где $|\bar{v}|$ обозначает сумму компонентов вектора \bar{v} . Тогда в пределе для бесконечной популяции получаем $G_p(\bar{p}(t)) = \bar{p}(t+1)$. Отметим, что G и G_p действуют на различные формы представления популяции, но могут быть преобразованы друг в друга несложным преобразованием.

Подобная формализация позволяет лучше понять и предсказать поведение ГА. В [19] G рассматривается как динамическая система, где поведение ГА имеет геометрическую интерпретацию, что позволило доказать некоторые поведенческие свойства ГА. «Геометрической картиной» при этом называется множество всевозможных векторов \bar{s} , образующих пространство S , на котором действует оператор G , что ведет к движению точки по некоторой траектории в этом пространстве, которая начинается в начальной точке $\bar{s}(0)$. При анализе динамики G прежде всего необходимо определить неподвижные точки G на S , т.е. множество точек $\bar{s}(t)$, для которых выполняется $G(\bar{s}(t)) = \bar{s}(t)$. Другими словами мы хотим найти такие точки $\bar{s}(t)$ в пространстве S , которые обладают свойством, что попадая в них ГА, не покидает их.

Эта общая проблема не была решена в работе [19], но в ней рассмотрены задачи поиска неподвижных точек F и M и

проанализированы некоторые свойства. Можно показать, что неподвижные точки F (при использовании только оператора отбора) представляют те популяции, которые сходятся к строкам с равными значениями фитнес-функции. В [19] показано, что только один класс таких неподвижных точек стабилен – множество неподвижных точек, соответствующих максимальному значению фитнес-функции в пространстве поиска. Другими словами, если популяция сходится в состояние, которое состоит не только из строк с максимальным значением фитнес-функции, малое изменение в распределении фитнеса в популяции может привести к уходу из неподвижной точки. Однако, если популяция состоит из особей с максимальными значениями фитнес-функции, то при небольшом изменении распределения фитнеса ГА всегда возвращается в неподвижную точку.

В [19] показано также, что в случае использования одного оператора M пространство \bar{s} имеет только одну неподвижную точку, где вектор \bar{s} состоит из равных вероятностей для всех строк в пространстве поиска. Аналогично при использовании M на \bar{p} существует одна неподвижная точка – все строки присутствуют в равной пропорции. Это означает, что в пределе при бесконечном числе особей в популяции кроссинговер и мутация при отсутствии отбора дают максимально смешанные популяции с равной вероятностью всех строк.

Приведенная формализация является только первым шагом в построении более точных математических моделей ГА. Основным ее недостатком является предположение о бесконечном числе особей в популяции, что ведет к тому, что результаты формулируются в терминах математического ожидания. Эта идеализация упрощает математический анализ, но поведение конечных популяций может существенно отличаться от бесконечных.

2.4. Применение марковских цепей для моделирования конечных популяций

В работе [20] для моделирования поведения ГА использовался математический аппарат марковских цепей. Напомним, что марковские цепи представляют стохастический процесс в котором вероятность того, что процесс в момент времени t будет в состоянии j , зависит только от состояния i в момент $(t-1)$.

В нашем случае для ГА состоянием является данная конечная популяция. Множество всех состояний есть множество всевозможных популяций размера n , которые могут быть пронумерованы некоторым каноническим способом. В [20] i – ая популяция представляется вектором $\bar{\theta}_i$ длины 2^n . Тогда компонента ‘ y ’ (с номером y) вектора $\bar{\theta}_i$ есть число строк y в популяции P_i . Очевидно, что в простом ГА текущая популяция j зависит (стохастически) только от популяции предыдущего поколения. Поэтому ГА аппарат марковских цепей может быть использован для моделирования ГА.

Таким образом, для построения модели мы должны определить вероятность перехода в ГА из данной популяции в любую другую. Множество всевозможных популяций мощности n может быть представлено матрицей Z большой размерности, в которой столбцы являются всевозможными векторами популяций $\bar{\theta}_i$. Показано [20], что Z может представить $T = \left(\frac{N + 2^n - 1}{2^n - 1} \right)$ всех возможных популяций размера мощности N . Тогда элемент матрицы Z_{yi} равен числу строк y в популяции i .

Рассмотрим небольшой пример построения матрицы Z для $n=2$ и $N=2$. В этом случае множество всевозможных популяций следующее:

$$p_0 = \begin{Bmatrix} 00 \\ 00 \end{Bmatrix}, p_1 = \begin{Bmatrix} 00 \\ 01 \end{Bmatrix}, p_2 = \begin{Bmatrix} 00 \\ 10 \end{Bmatrix}, p_3 = \begin{Bmatrix} 00 \\ 11 \end{Bmatrix}, p_4 = \begin{Bmatrix} 01 \\ 01 \end{Bmatrix},$$

$$p_5 = \begin{Bmatrix} 01 \\ 10 \end{Bmatrix}, p_6 = \begin{Bmatrix} 01 \\ 11 \end{Bmatrix}, p_7 = \begin{Bmatrix} 10 \\ 10 \end{Bmatrix}, p_8 = \begin{Bmatrix} 10 \\ 11 \end{Bmatrix}, p_9 = \begin{Bmatrix} 11 \\ 11 \end{Bmatrix}.$$

Для данного примера получаем матрицу

$$Z = \begin{pmatrix} 2111000000 \\ 0100211000 \\ 0010010210 \\ 0001001012 \end{pmatrix},$$

где строки матрицы соответствуют строкам – хромосомам (00, 01, 10, 11), а столбцы – возможным популяциям $p_0 - p_9$. Например, элемент $Z_{11} = 2$ показывает, что строка (00) в популяции p_0 встречается 2 раза. Аналогично $Z_{12} = 1$ говорит о том, что строка (00) в популяции p_1 встречается 1 раз. В такой нотации состояние марковской цепи соответствует столбцу матрицы Z . Далее необходимо определить матрицу переходов Q размера $N \times N$, где элемент Q_{ij} равен вероятности перехода из популяции i в популяцию j в процессе эволюции ГА. Матрица Q может быть использована при описании некоторых поведенческих свойств ГА.

Пусть $p_i(y)$ обозначает вероятность генерации строки y вследствие действия операторов отбора и рекомбинации на популяции p_i . Число вхождений строки y в популяцию p_j равно Z_{yj} . Поэтому вероятность генерации некоторого числа строк из популяции p_i определяется тем, что Z_{yj} строк y в популяции генерируется из популяции p_i . Оно равно вероятности генерации строки y в результате применения Z_{yj} различных операторов отбора и рекомбинации, умноженной на число способов, в результате которых эти Z_{yj} операторов могут случиться в процессе выполнения общего числа шагов n отбора и репродукции. Следуя [20], можно оценить это для каждой строки.

Число способов выбора Z_{0j} вхождений строки 0 для Z_{0j} позиций (слотов) в популяции j равно $\binom{N}{Z_{0j}}$. Выбор строки 0 Z_{0j} раз оставляет $(N - Z_{0j})$ позиций для заполнения в новой популяции. Число способов размещения Z_{1j} вхождений строки 1 в $(N - Z_{0j})$ позиций равно

$$\binom{N - Z_{0j}}{Z_{1j}}.$$

Продолжая этот процесс, получаем выражение для всевозможных способов формирования популяции p_j в результате выполнения N шагов отбора и рекомбинации:

$$\binom{N}{Z_{0j}} \binom{N - Z_{0j}}{Z_{1j}} \dots \binom{N - Z_{0j} - Z_{1j} - \dots - Z_{2^{l-2},j}}{Z_{2^{l-1},j}} = \frac{N!}{Z_{0,j}! Z_{1,j}! \dots Z_{2^{l-1},j}!}.$$

При формировании этого выражения мы нумеровали строки от 0 до $(2^n - 1)$. Вероятность того, что правильное число вхождений каждой строки y в популяции p_j генерируется из популяции p_i равна

$$\prod_{y=0}^{2^n-1} [p_i(y)]^{Z_{y,j}}.$$

Вероятность того, что популяция p_j генерируется из популяции p_i определяется произведением двух полученных выражений:

$$Q_{ij} = \frac{N!}{Z_{0,j}! Z_{1,j}! \dots Z_{2^{l-1},j}!} \prod_{y=0}^{2^n-1} [p_i(y)]^{Z_{y,j}} = N! \prod_{y=0}^{2^n-1} \frac{[p_i(y)]^{Z_{y,j}}}{Z_{y,j}!}.$$

В полученном выражении осталось пока неопределенным $p_i(y)$ - вероятность того, что строка y будет генерирована в результате одного шага отбора-рекомбинации на популяции p_i . Для определения $p_i(y)$ используем определенные выше матрицы F и M . Отметим, что $p_i(y)$ есть математическое ожидание доли строки y в популяции, генерируемой из p_i

в процессе ГА. Доля строки y в p_i определяется $\frac{\bar{\phi}_i}{|\bar{\phi}_i|_y}$, где $|\bar{v}|$ обозначает сумму компонент вектора \bar{v} и $(v)_y$ обозначает y -ю компоненту этого вектора. Тогда вероятность выбора строки y на каждом шаге отбора равна $\left(\frac{F\bar{\phi}_i}{|F\bar{\phi}_i|}\right)_y$, и математическое ожидание доли строки y в следующей популяции – $\left[M\left(\frac{F\bar{\phi}_i}{|F\bar{\phi}_i|}\right)\right]_y$.

Так как $p_i(y)$ эквивалентно математическому ожиданию доли строки y в следующей популяции, мы можем записать окончательное выражение для Q_{ij} :

$$Q_{ij} = N! \prod_{y=0}^{2^l-1} \frac{\left[M(F\bar{\phi}_i/|F\bar{\phi}_i|)_y\right]^{Z_{yj}}}{Z_{yj}!}.$$

Таким образом, полученная матрица Q_{ij} дает точную математическую модель простого ГА для конечной популяции. Подобные модели являются наиболее детальными и в принципе они могут быть использованы для прогнозирования любого аспекта поведения ГА. Однако на практике использование таких моделей проблематично вследствие высокой размерности модели. Например, для умеренной сложности ГА с $n=8$ и $N=8$ матрица переходов Q_{ij} имеет более 10^{29} элементов. Это не означает, что подобные модели бесполезны. Они позволяют исследовать некоторые фундаментальные свойства ГА, такие как наличие и условия неподвижных точек, асимптотическое поведение ГА и т.п.

2.5. Применение методов статистической механики для моделирования ГА

В отличие от рассмотренного выше микроописания ГА, где фактически исследуется каждое возможное решение, входящее в популяцию, разработан альтернативный макроскопический подход [21, 18], где ГА описывается в терминах статистических свойств популяции. Преимущество макроскопического подхода состоит, прежде всего, в использовании модели с небольшим числом параметров. При этом исследование динамики системы с огромным числом степеней свободы сводится к изучению системы с относительно малым числом параметров. Моделирование нелинейной системы с небольшим числом степеней свободы, конечно, представляет меньшую проблему. При этом количество параметров макромоделей не должно зависеть ни от размерности задачи, ни от мощности популяции.

С другой стороны макроскопический уровень имеет также определенные недостатки. Очевидно, что малое число обобщенных параметров содержит гораздо меньше информации о популяции, чем ее полное описание. Поэтому этот подход является приближенным и часто трудно оценить насколько приближение будет грубым для данной проблемы. При этом очень важным является выбор макропараметров. Основная сложность заключается в оценке влияния генетических операторов на эти параметры. Известны работы [21], в которых получены оценки воздействия генетических операторов на математическое ожидание и дисперсию, определенные в контексте ГА. В данном разделе представлено описание поведения ГА с помощью методов статистической механики, которое опубликовано в работах [21,18]. При этом методы статистической механики применяются двояко. Во-первых, генетические операторы (отбора-репродукции, кроссинговера и мутации) имеют

вероятностный характер. Поэтому методы статистической механики можно использовать для расчета ожидаемых результатов применения данных операторов и отклонения от ожидаемых значений. Во вторых, там где требуется информация, отсутствующая в макроописании, она может быть получена из макропараметров методом максимума энтропии.

В настоящем разделе мы будем рассматривать следующую модификацию ГА, который работает с популяцией мощности N битовых строк (особей) длины n . Каждый элемент строки принимает значение из множества $\{-1, 1\}$. Строки (хромосомы) будем обозначать x_i^α , где α - номер особи в популяции и нижний индекс определяет i -й элемент (ген) строки. Целевую функцию (ЦФ) обозначим через E^a . Будем рассматривать задачу минимизации этой функции. Следует отметить, что понятия «целевая функция» и «фитнесс-функция» в данном разделе имеют разный смысл. Целевая функция является средством измерения качества отдельной особи популяции при решении поставленной задачи. В отличие от нее фитнес-функция используется при отборе особей популяции. Таким образом, ЦФ является свойством данной особи, которое не зависит от ГА. Фитнесс-функция является свойством используемой стратегии отбора, которая используется в данном ГА.

При отборе используются два метода: 1) пропорциональный отбор (метод рулетки); 2) правило выбора Больцмана. Конкретный метод отбора определяется зависимостью между ЦФ и фитнес-функцией $F^a = (E^a)$.

Напомним, что в правиле выбора по Больцману $F(E) = e^{-\beta E}$ и вероятность выбора особи x^a определяется

$$p^a = \frac{e^{-\beta E}}{\sum_{a=1}^n e^{-\beta E}},$$

где β выступает в роли параметра (аналога) температуры в распределении Больцмана, управляющего величиной селекционного давления. Этот вид отбора имеет ряд преимуществ, которые приведены в разделе 4.2. Могут использоваться и другие методы отбора родителей, представленные в разделе 4.2.

В ГА используется оператор мутации, где знак каждой из компонент хромосомы x_i^a изменяется с вероятностью p_m . Используется однородный (равномерный) оператор кроссинговера, который производит двух потомков с вероятностью $p_c=1$. Известны работы, где данный подход использует и традиционный одноточечный кроссинговер [21].

Определим величину h^a , которую будем называть «полем» (в некоторых работах – «энергией»), следующим образом $h^a = \sum_{i=1}^n w_i x_i^a$, где w_i - весовые коэффициенты, которые определяют относительную важность каждого бита для данной задачи. Мы будем рассматривать задачи, для которых ЦФ является некоторой функцией от величины h^a - $E^a = E(h^a)$. Для различных задач и даже для различных вариантов одной и той же задачи определяются свои значения w^i и своя функция $E(h)$. Следует отметить, что для некоторых задач (например, spin-glass chain [18,21]) использовались функции другого вида.

Далее рассмотрим распределение величины поля на популяции. Следуя статистическому подходу, для описания популяции будем использовать моменты K_1 (иногда называемые семиинвариантами) n -го порядка. Нас интересуют, прежде всего, четыре первых семиинварианта - K_1, K_2, K_3, K_4 . Первый момент представляет выборочное математическое ожидание популяции:

$$K_1 = \frac{1}{N} \sum_{a=1}^N h^a ;$$

второй момент – выборочную дисперсию:

$$K_2 = \frac{1}{N} \sum_{a=1}^N (h^a - K_1)^2 .$$

Третий момент K_3 связан с асимметрией, а четвертый K_4 - с эксцессом. В общем виде моменты определяются через производящие функции. Следует отметить, что все статистики усредняются только по популяции, а не по всем допустимым особям. Напомним, что для плотности распределения $\rho(x)$ производящая функция определяется следующим образом

$$Z(\gamma) = \int \rho(x) e^{\gamma x} dx .$$

Тогда момент l -го порядка определяется следующим выражением

$$m_l = (-1)^l \frac{\partial^l}{\partial \gamma^l} Z \Big|_{\gamma=1} .$$

Таким образом, приведенные моменты (семиинварианты) характеризуют популяцию, являются указанными выше ее макропараметрами, которые изменяются в процессе эволюции.

Кроме приведенных моментов введем также в качестве макропараметра популяции меру сходства между особями популяции. Корреляция q^{ab} служит для определения меры сходствами между двумя хромосомами a и b , взвешенной в соответствии с относительной важностью каждой компоненты (гена):

$$q^{ab} = \frac{1}{N} \sum_i x_i^a x_i^b .$$

Тогда средняя по популяции корреляция равна:

$$q = \frac{1}{N(N-1)} \sum_a \sum_{b \neq a} q^{ab} .$$

Таким образом, цель состоит в исследовании динамики введенных статистических параметров популяции – моментов и корреляции в процессе эволюции. При этом главным является исследование изменения параметров при применении генетических операторов отбора,

кроссинговера и мутации. При заданном распределении поля $\rho_t(h)$ поколения t схематически это можно представить следующим образом:

$$\rho_t(h) \xrightarrow{\text{отбор}} \rho_t^s(h) \xrightarrow{\text{кроссинговер}} \rho_t^{sc}(h) \xrightarrow{\text{мутация}} \rho_t^{scs}(h) = \rho_t(h+1)$$

Далее последовательно рассмотрим влияние генетических операторов отбора, кроссинговера и мутации на выбранные статистические параметры – моменты и корреляцию.

Сначала исследуем влияние оператора отбора (репродукции). Если $\rho(h)$ - распределение поля на популяции до применения оператора отбора, то можно предположить, что распределение поля после его применения будет пропорционально $\rho(h)F(E(h))$ - произведению вероятности того, что некоторое значение поля присутствует в популяции, на вероятность его выбора. В общем случае это предположение некорректно для популяций конечной мощности, поскольку в нем не учитываются выборочные отклонения. Для более точного учета воздействия оператора отбора рассмотрим производящую функцию

$$Z_s(\gamma) = \sum_{a=1}^N F[E(h^a)e^{\gamma h^a}],$$

где γ - формальный параметр, а дифференцирование $Z_s(\gamma)$ по γ дает моменты поля h . Логарифм от этой величины является производящей функцией для распределения поля после применения оператора отбора:

$$K_m^s = \frac{\partial^m}{\partial \gamma^m} \log Z_s \Big|_{\gamma=0}.$$

Процесс отбора включает в себя независимый выбор с возвращением n значений h в соответствии с распределением поля. Поэтому производящая функция после применения оператора отбора через математическое ожидание по распределению $\rho(h)$ следующим образом:

$$M(\log Z)_{\rho(h)} = \left(\prod_{a=1}^N \int_{-\infty}^{\infty} \rho(h^a) dh^a \right) \log Z_s.$$

Данное выражение представляет фундаментальный результат для вычисления оператора отбора на распределение произвольной случайной величины, которая функционально связана с ЦФ. Оно позволяет предсказать значения моментов этого распределения после действия отбора путем дифференцирования по переменной γ . Данное выражение мало пригодно для практического использования поскольку содержит P -кратный интеграл. Можно показать, что оно может быть сведено с помощью специальных преобразований к двойному интегралу, что дает возможность точного расчета с использованием численных методов. Кроме этого, для вычисления интеграла может быть применен асимптотический анализ. Например, при популяции большого объема справедливо приближенное соотношение

$$M(\log Z_s)_{\rho(h)} = \log NM(F(h)e^{\gamma h})_{\rho(h)} - \frac{1}{2N} \left(\frac{M(F(h)^2 e^{2\gamma h})_{\rho(h)}}{M(F(h)e^{\gamma h})^2} - 1 \right) + O(N^{-2}).$$

Здесь $F(h)$ означает $F(E(h))$. Первое слагаемое представляет производящую функцию для бесконечной мощности популяции, которая определяется производящей функцией для распределения $\rho(h)F(h)$. Второе слагаемое является главным членом корректирующего ряда для конечной популяции. Разложение в ряд Тэйлора левой части выражения по переменной γ дает все моменты $\rho(h)$ после применения оператора отбора в терминах средних значений $F(h)$ по $\rho(h)$ до применения отбора. Эти средние значения можно выразить через моменты до применения оператора кроссинговера или путем усреднения по распределению $\rho(h)$, для которого известны точные значения моментов или, если $F(h)$ допускает разложение в ряд Тэйлора, путем непосредственного

выражения этих средних величин через моменты до применения оператора отбора.

Далее рассмотрим влияние оператора отбора на корреляцию q , которая непосредственно не связана с ЦФ. На корреляцию влияют два аспекта. Прежде всего, при отборе корреляция изменяется вследствие дублирования некоторых хромосом. В этом случае хромосомы полностью коррелированы сами с собой, поэтому очевидно, что дублирование хромосом увеличивает корреляцию популяции q . Кроме этого, вследствие изменения математического ожидания и других моментов ЦФ изменяется и ожидаемое значение корреляции. Например, если особи популяции имеют большие значения фитнес-функции, то, скорее всего, популяция сильно коррелирована, так как существует не очень много особей с высокими значениями фитнес-функции и поэтому много хромосом наверняка очень похожи. Это следует из принципа максимума энтропии. Это можно показать, рассматривая оператор отбора в качестве генератора полиномиального распределения. Пусть m^a – количество реализаций, при которых выбрана хромосома a . Тогда после применения оператора отбора корреляция имеет вид:

$$q^s = \frac{1}{N(N-1)} \sum_{a=1}^N m^a (m^a - 1) + \frac{1}{N(N-1)} \sum_{a=1}^n \sum_{b \neq a}^n m^a m^b q^{ab}.$$

Здесь первое слагаемое соответствует компоненте дублирования особей и обусловлено корреляцией между повторяющимися хромосомами (которые имеют единичную корреляцию). После усреднения по популяции получаем:

$$q^s = \sum_{a=1}^N (p^a)^2 + \sum_{a=1}^N \sum_{b \neq a}^N p^a p^b q^{ab},$$

где p^a – вероятность отбора строки. Отметим, что первое слагаемое учитывает воздействие повторяющихся строк на значение корреляции, а второе слагаемое соответствует максимуму энтропии. Первое слагаемое

отражает влияние конечности мощности популяции и вносит основной вклад в общее значение корреляции для всей популяции, что уменьшает эффективность оператора кроссинговера.

Для класса задач, в которых ЦФ является функцией поля, наиболее естественным является использование равномерного (однородного) оператора кроссинговера (описан в разделе 4.2). Для случая, когда сохраняются оба потомка, полученные при каждом кроссинговере, распределение элементов (генов) сохраняется и после применения оператора. Поэтому оператор кроссинговера никак не влияет на математическое ожидание h , K_1 или корреляцию q . Однако оператор кроссинговера влияет на моменты высоких порядков:

$$K_2^c = K_2 - \frac{N}{2(n-1)} \left[K_2 - \frac{n(n-1/N)}{4} (1-q) \right].$$

Способность оператора повышать разнообразие популяции (генетического материала) связана с тем, что близость родительских хромосом измеряется корреляцией q . Можно вывести также выражения и для моментов более высоких порядков. Оператор кроссинговера возвращает их к «естественным значениям», которые соответствуют популяции с максимальной энтропией.

Далее исследуем влияние оператора мутации на статистические параметры. В данной модификации ГА используется следующий оператор мутации. С вероятностью p_m каждый бит произвольно взятой хромосомы изменяет свое значение x_i^a на противоположное: $-x_i^a$. Можно показать, что влияние такого оператора мутации на статистические параметры следующее:

$$K_1^m = K_1 + 2p^m(K_1^{random} - K_1),$$

$$K_2^m = K_2 + 4p^m(p^m - 1)(K_2^{random} - K_2),$$

$$q^m = (1 - 2p^m)^2 q$$

и т.д. Здесь верхний индекс “random” означает моменты случайной популяции, то есть популяции, которая получена повторяющимся применением оператора мутации.

Если в качестве макропараметров взять указанные моменты и корреляцию, то интересно наблюдать за их изменением и значением ЦФ в процессе эволюции на примере некоторых «учебных» задач. В целом, характерна следующая картина. Оператор отбора может увеличить среднее значение решения s , если оно не является оптимальным, и уменьшить дисперсию. Кроме этого. Оператор отбора может вызвать увеличение значений моментов высоких порядков корреляцию q . Оператор кроссинговера не влияет на математическое ожидание h или среднее значение ЦФ, так как оно является функцией от h . Зато он воздействует на моменты старших порядков. Этот оператор увеличивает дисперсию, усредненную по q , и уменьшает значение старших моментов. Мутация же сдвигает значения всех моментов к их значениям для случайной популяции.

В качестве примера рассмотрим задачу нахождения подмножества s заданной суммой [21]. Для данной задачи определяются значения весовых коэффициентов w_i , каждый из которых является положительным числом. Необходимо найти такое подмножество весов, сумма элементов которого максимально приближается к некоторому наперед заданному значению G . В этом случае ЦФ определяется следующим образом:

$$E^a = (h^a - G)^2.$$

В общем случае данная задача является NP-полной и является частным случаем задачи о рюкзаке, которая будет рассмотрена в разделе 3.1. Интересной особенностью этой задачи является то, что в ней проявляются как направленное действие оператора отбора, при котором математическое ожидание поля смещается по направлению к оптимальному значению G , так и стабилизирующее воздействие, при

котором математическое ожидание сохраняет значение, близкое к G , но при этом уменьшается дисперсия, так как особи популяции группируются в окрестности экстремального решения. Это хорошо видно на рис.2.2. Здесь результаты усреднены по 500 экспериментам. Распределение полей в популяции показано для поколений 0, 10, 20, 30, 50, 80, 110 и 140. Оптимальное значение $\frac{G}{N} = 0,25$. Мощность популяции $n=80$ и разрядность хромосомы $l=150$. На рисунке ясно видны два этапа поиска: 1) направленный, где происходит смещение среднего значения в сторону оптимума и 2) стабилизирующий, когда распределение сужается вокруг экстремального значения.

Макропараметрами для данной задачи являются два первых момента h и корреляция q . Динамика изменения этих параметров показана на рис. 2.3. Здесь пунктирными линиями показан теоретический прогноз, а сплошными линиями - данные компьютерного моделирования. Следует отметить высокую точность прогноза – различие между прогнозируемыми и экспериментальными данными составляет несколько процентов.

Таким образом в данном разделе представлены различные подходы к моделированию простых ГА, которые позволяют с различных сторон исследовать их свойства на микроскопическом и макроскопическом уровне.

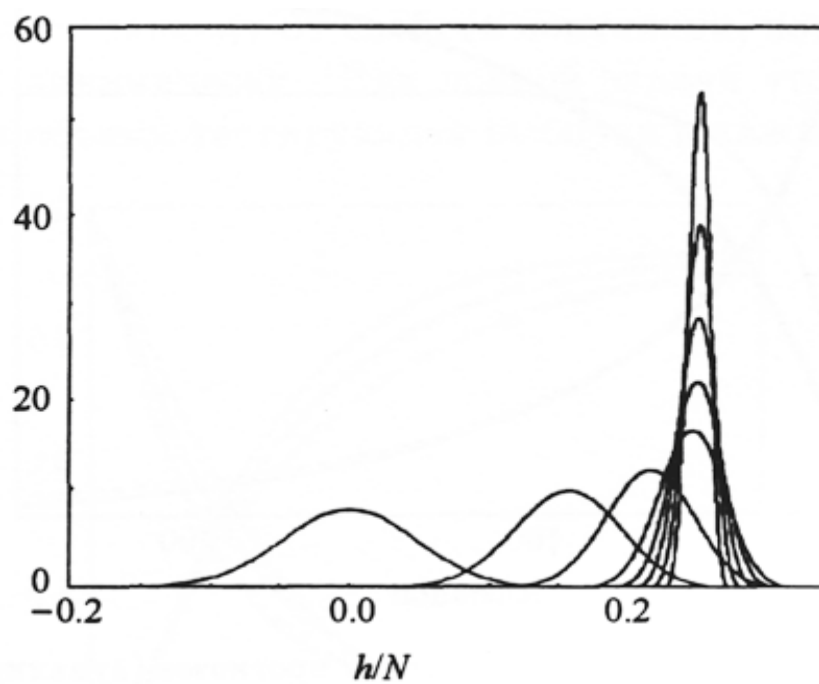


Рис. 2.2. Изменение распределения в процессе эволюции

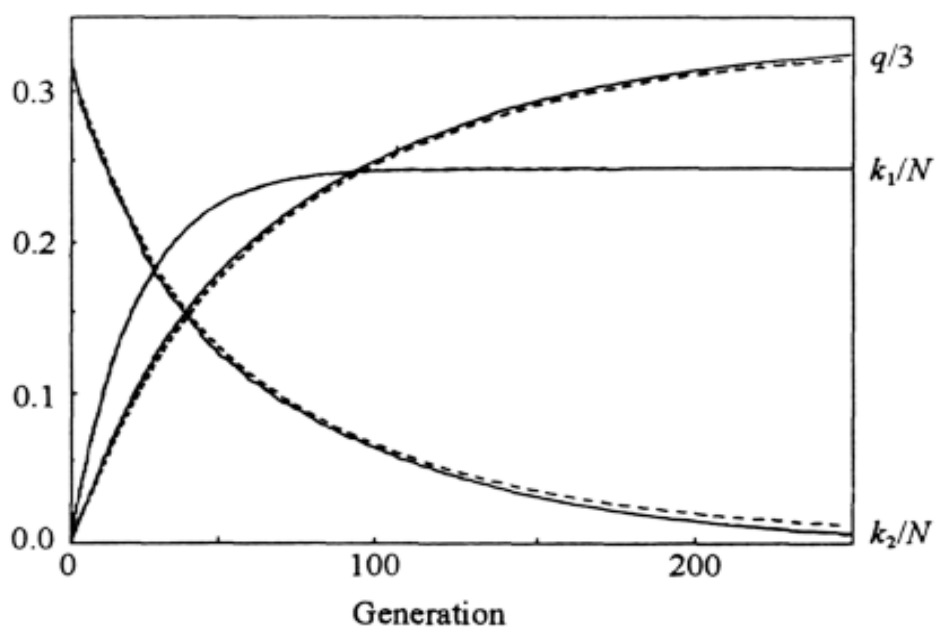


Рис.2.3. Сравнение результатов теоретических прогнозов и компьютерного моделирования.

2.6 Контрольные вопросы к разделу 2

1. Что отражает понятие схема в ГА?
2. Дано 3 стринга $A_1 = 11101111$, $A_2 = 000101000$, $A_3 = 01000011$ и 6 схем – $H_1 = 1*****$, $H_2 = 0*****$, $H_3 = *****11$, $H_4 = ***0*00*$, $H_5 = 1*****1*$, $H_6 = 1110**1*$. Какие схемы покрывают приведенные стринги? Какой порядок и определенную длину имеют эти схемы? Оцените вероятность выживания каждой схемы при мутации с вероятностью $P_m = 0,002$. Оцените вероятность выживания каждой схемы при кроссинговере с вероятностью $P_c = 0,80$.
3. Популяция содержит следующие стринги с соответствующими значениями фитнес-функции для нулевого поколения

Стринг	Значение фитнес-функции
10001	20
11100	10
00011	5
01110	15

Вероятности мутации $P_m = 0,01$ и кроссинговера $P_c = 1,0$ соответственно.

Вычислите ожидаемое число схемы вида $H_1 = ****$ в поколении 1.

Аналогично сделайте это для схемы $H_2 = 0**1*$.

4. Сформулируйте фундаментальную теорему ГА.
5. Какую часть в нее вносит оператор репродукции?
6. Какую часть в нее вносит оператор кроссинговера?
7. Какую часть в нее вносит оператор мутации?
8. Приведите пример вычисления $m(H, t + 1)$ для простого ГА.

9. Приведите пример вычисления $m(H, t + 1)$ для репродукции и кроссинговера.
10. Приведите пример вычисления $m(H, t + 1)$ для репродукции и мутации.
11. Приведите пример вычисления выживания $P(s)$ нескольких схем.
12. Приведите пример вычисления разрушения $P(d)$ нескольких схем.
13. Что определяет состав популяции в модели динамической системы ?
14. Как связаны $\bar{p}(t)$ и F в модели динамической системы?
15. Дайте геометрическую интерпретацию поведенческих свойств ГА в модели динамической системы.
16. Как представляется популяции в модели ГА «марковской цепи»?
17. Опишите матрицу вероятностей переходов для «марковской цепи» ГА.
18. Опишите макроскопический подход к моделированию ГА.
19. Какие макропараметры используются в статистической модели?
20. Как влияют на макропараметры генетические операторы?

3. ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ ДЛЯ ЗАДАЧ КОМБИНАТОРНОЙ ОПТИМИЗАЦИИ

До сих пор рассматривалось применение ГА, в основном, при решении задачи численной оптимизации при поиске экстремумов функции. Но в настоящее время ГА используются больше для решения задач комбинаторной оптимизации, где публикуется подавляющая часть работ.

Базовой задачей комбинаторной оптимизации является задача коммивояжера, для которой отрабатываются новые методы решения задач данного типа. Однако, сначала мы рассмотрим задачи об укладке рюкзака и о покрытии множества, которые допускают использование простого (классического ГА [2,11]).

3.1. Задача об укладке рюкзака

Эта задача имеет следующую неформальную простую постановку. Имеется рюкзак объемом C и n различных предметов. Каждый предмет i имеет известный объем W_i и стоимость P_i ($i = 1 \dots n$). В рюкзак можно положить целое число различных предметов. Нужно упаковать рюкзак так, чтобы полная стоимость уложенных предметов была максимальной, а их общий объем не превышал заданный объем C – емкость рюкзака. Форма предметов здесь не учитывается.

Формальная постановка задачи: для данного множества весов W_i , стоимостей P_i и объема C надо найти двоичный вектор

$X = (x_1, \dots, x_n)$, где:

$x_i = 1$, если предмет укладывается в рюкзак;

$x_i = 0$, если предмет не укладывается;

при этом должно выполняться:

$$V = \sum_{i=1}^n W_i \leq C, \text{ и } \sum P_i = \max.$$

Так как решение задачи можно представить двоичным вектором $X = (x_1, \dots, x_n)$, то очевидно при его поиске можно применить простой ГА со стандартными операторами скрещивания и мутации. Но при этом, что на каждом шаге надо следить за тем, чтобы новые решения, полученные в результате скрещивания или мутации, удовлетворяли требуемому ограничению $V \leq C$. В случае невыполнения ограничения «неправильное» потенциальное решение должно быть уничтожено, что ведет к сокращению популяции.

В качестве фитнес-функции в простейшем случае можно взять

$$P(X) = \sum_{i=1}^n x_i \cdot P_i, \text{ но в этом случае, как указано выше, есть проблемы с}$$

неправильными решениями.

Данная задача относится к классу задач с ограничениями, при решении которых применяются следующие подходы [17]: 1) введение в фитнес-функцию дополнительного штрафа; 2) использование алгоритмов “восстановления” некорректных решений.

1) В первом случае в фитнес-функцию вводится дополнительная штрафная функция, которая для неправильных решений дает большие отрицательные значения ЦФ. При этом задача с ограничениями трансформируется в задачу без ограничений путем назначения штрафа для некорректных решений. Фитнес-функция для каждой особи может быть определена следующим образом

$$f(X) = \sum_{i=1}^n x_i \cdot P_i - Pen(X).$$

Разработано множество методов назначения штрафных значений, из которых ниже рассмотрено только три вида, в которых рост значения

штрафной функции относительно степени нарушения ограничения имеет логарифмический, линейный и квадратичный характер:

$$\text{а) } Pen(X) = \log_2(1 + \rho \cdot (\sum_{i=1}^n x_i \cdot W_i - C)),$$

$$\text{б) } Pen(X) = \rho \cdot (\sum_{i=1}^n x_i \cdot W_i - C),$$

$$\text{в) } Pen(X) = (\rho \cdot (\sum_{i=1}^n x_i \cdot W_i - C))^2,$$

где для всех трех случаев $\rho = \max_{i=1 \dots n} \{P_i / w_i\}$.

2) Второй подход к решению задач с ограничениями основан на специальных алгоритмах “восстановления” некорректных решений. Следует отметить, что многие алгоритмы восстановления требуют значительных вычислительных ресурсов, и полученные решения иногда требуют адаптации к конкретным практическим приложениям.

В этом случае в качестве фитнес-функции используется

$$f(X') = \sum_{i=1}^n x_i' P_i, \text{ где вектор } X' \text{ – восстановленная версия исходного вектора } X.$$

Здесь следует отметить, по крайней мере, два аспекта. Во-первых, можно использовать различные алгоритмы восстановления. Во-вторых, восстановленные особи могут замещать только некоторую часть исходных особей в популяции. Процент замещаемых особей может варьироваться от 0% до 100% и его значение является важнейшим параметром метода восстановления. В некоторых работах отмечается, что наилучшие результаты получаются при 5%, во всяком случае, лучше, чем в двух крайних случаях – 0% (без замещения) и 100% (любая восстановленная особь заменяет исходную). Ниже приведен простой алгоритм восстановления.

Восстановление(X)

```
{
  переполнение_рюкзака=false;
  X'=X;
  if ( $f(X') = \sum_{i=1}^n x'_i \cdot W_i > C$ )
    then переполнение_рюкзака=true;
  while(переполнение_рюкзака)
    {
      i=выбор предмета из рюкзака;
      // удаление выбранного предмета из рюкзака;
       $x'_i = 0$ ;
      if ( $f(X') = \sum_{i=1}^n x'_i \cdot W_i \leq C$ )
        then переполнение_рюкзака=false;
    }
}
```

При этом используются два основных способа выбора объекта:

- а) случайный выбор объекта из рюкзака;
- б) “жадное восстановление”, при котором сначала предметы сортируются в порядке убывания стоимости P_i и на каждом шаге для удаления выбирается предмет минимальной стоимости (из имеющихся в рюкзаке).

3) Третий подход к решению задач с ограничениями использует специальное отображение (декодирование) особей, которое гарантирует генерацию допустимого решения (с учетом ограничений), или используют проблемно-ориентированные генетические операторы, сохраняющие корректность решения.

Рассмотрим один из возможных вариантов алгоритма декодирования, который основан на кодировании решения вектором целых чисел, так называемом «упорядоченном представлении» (ordinal representation), подробно описанном в разделе 3.3.2. Здесь каждая хромосома кодируется вектором целых чисел, где i -ая компонента вектора – целое число в диапазоне от 1 до $n-i+1$. “Упорядоченное представление” использует для ссылок (базовый) список предметов L . Вектор e фактически содержит указатели на базовый список L . Декодирование вектора осуществляется путем выбора соответствующего предмета из текущего списка и удаления его из базового списка. Например, при базовом списке предметов $L=(1,2,3,4,5,6)$ текущий вектор $e=(4,3,4,1,1,1)$ декодируется в следующую последовательность предметов: 4, 3, 6, 1, 2, 5. Первым в тур включается четвертый город списка L - 4, который устраняется из L . Далее в тур включается третий город из текущего списка L . Затем включается 6 - четвертый город текущего L и т.д. Подробнее описание этого представления и выполнение кроссинговера на нем описано в разделе 3.3.2. В данном методе хромосома может интерпретироваться как стратегия (порядок) включения предметов в решение. Отметим, что основным достоинством данного кодирования хромосомы является то, что на нем работает одноточечный кроссинговер. То есть для двух допустимых решений – родителей кроссинговер порождает также допустимое решение – потомок (подробнее см. в разделе 3.3.2). Оператор мутации при данном представлении выполняется путем замены i -го гена (целочисленной компоненты вектора) на случайное целое число из диапазона $[1, \dots, n-i+1]$. Алгоритм декодирования представлен ниже.

Декодирование(X)

```

{
    генерация списка предметов L;
    i=1;
    Сумма_весов=0;
    Суммарная_стоимость=0;
    While( $i \leq n$ )
    {
        j=  $x_i$ ;
        удаление j-го предмета из списка L;
        if(  $\text{сумма\_весов} + W_j \leq C$  ) then
        {
            Сумма_весов= Сумма_весов +  $W_j$ ;
            Суммарная_стоимость= Суммарная_стоимость +  $P_j$ ;
        }
        i=i+1;
    }
}

```

Очевидно, что представленный алгоритм зависит от способа генерации списка предметов L. Обычно используются два метода генерации этого списка:

а) список предметов L генерируется в том порядке, в котором предметы расположены во входном файле (как правило, случайно);

б) список предметов L генерируется в порядке убывания их стоимостей (жадный алгоритм). Декодирование вектора X выполняется на

основе отсортированного вектора. Например, x_{25} интерпретируется как 25-й предмет текущего списка L.

Задача об укладке рюкзака в различных вариантах имеет многочисленные практические приложения. Например, к ней можно свести оптимизацию загрузки транспорта и т.п.

3.2. Задача о покрытии

Есть множество элементов S и множество подмножеств F_i , состоящих из элементов S . Необходимо найти минимальное число подмножеств из F таких, чтобы объединение этих подмножеств содержало все элементы множества S . Задача имеет простую экономическую интерпретацию: пусть, например, имеется некоторое количество клиентов и для их обслуживания необходимо выбрать некоторое количество сервисных центров

Очевидно, здесь решение можно также представить двоичным вектором

$X = (x_1, \dots, x_n)$, где:

$x_i = 1$, подмножество $F(i)$ входит в покрытие;

$x_i = 0$, если $F(i)$ не входит в покрытие;

При этом:

$$\bigcup X_i F_i = S$$

$$\sum X_i = \min$$

Поскольку решение задачи, как было показано выше, представляется двоичным вектором, то при его поиске можно использовать простой ГА со стандартными операторами кроссинговера и мутации.

3.3. Задача коммивояжера

Напомним неформальную постановку этой классической задачи. Коммивояжер (бродячий торговец) должен выйти из первого города, посетить по одному разу в некотором порядке все города и вернуться в исходный город. На рис.3.1 приведен пример задачи для четырех городов.

Расстояния между городами известны. В каком порядке следует обходить города, чтобы замкнутый путь (тур) коммивояжера был кратчайшим?

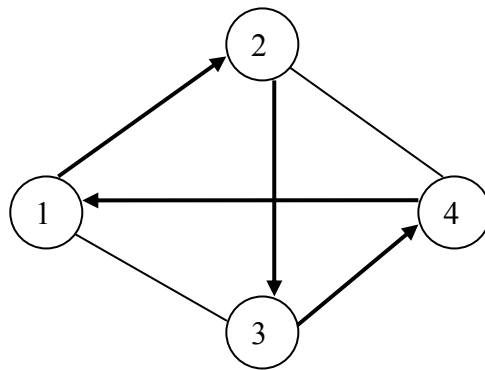


Рис.3.1. Задача коммивояжера

В формальной постановке задачи коммивояжера (ЗК): имеется полный взвешенный ориентированный граф без петель G с множеством вершин $N=\{1,2,\dots,n\}$; веса всех дуг неотрицательны; в этом графе требуется найти гамильтонов цикл с минимальной стоимостью. Исходная информация по ЗК представляется в виде $n \times n$ матрицы $S = \{s_{ij}\}$ - вес дуги (i,j) графа G , $i = \overline{1,n}$, $j = \overline{1,n}$, $i \neq j$; все элементы главной диагонали нулевые $s_{ii} = 0$ (но в некоторых постановках полагаются $s_{ii} = \infty$). Обычно s_{ij} интерпретируется как расстояние между городами i и j . С учетом других возможных интерпретаций на матрицу S требование

симметричности не налагается. Например, в случае интерпретации s_{ij} как стоимости проезда в общем случае может быть $s_{ij} \neq s_{ji}$. В общем случае не считается обязательным и выполнение неравенства треугольника $s_{ij} + s_{jk} \geq s_{ik}$.

Тур коммивояжера может быть описан циклической перестановкой $t=(j_1, j_2, \dots, j_n, j_1)$, причём все j_1, j_2, \dots, j_n – разные; повторяющийся в начале и в конце номер города j_1 , показывает, что перестановка циклическая. Пространством поиска решений этой задачи является множество перестановок n городов. Любая простая (одиночная) перестановка n городов даёт решение, являющееся полным туром из n городов. Оптимальным решением является перестановка, которая даёт \min стоимость тура. Очевидно, размерность пространства поиска для несимметричной задачи равна $(n-1)!$. Известно, что эта задача является NP – полной, т.е. переборной. Она имеет многочисленные практические приложения, в которых число “городов” может быть достаточно большим. Например, при производстве сложных деталей задача сверления отверстий может иметь сотни и тысячи “городов”. При производстве СБИС возникают задачи (например, по внесению примесей в полупроводник) с числом “городов” около миллиона. За последние десятилетия разработано достаточно много алгоритмов решения этой задачи, дающих субоптимальное решение. В последнее десятилетие эта задача является базовой для исследования ГА в области комбинаторной оптимизации.

Очевидно, что двоичное представление тура при решении ЗК нецелесообразно. Действительно если мы интересуемся оптимальной перестановкой городов, т.е. (i_1, i_2, \dots, i_n) и используем двоичное представление в виде одного бинарного вектора, то изменение даже в одном бите двоичного кода перестановки может дать двоичный вектор, не

принадлежащий к области решения, т.е. не являющейся перестановкой n городов.

Для решения 3К с помощью генетических алгоритмов разработаны специальные методы представления (кодирования) решений и соответствующие проблемно-ориентированные генетические операторы [11,17]. В основном, используются три способа представления тура при решении 3К с использованием ГА: 1) представление порядка; 2) представление соседства; 3) представление путей. Для каждого из этих представлений разработаны свои “генетические” операторы. Оператор мутации относительно легко определить на этих представлениях в виде одиночной перестановки соседних городов в туре. Поэтому в дальнейшем мы, в основном, рассмотрим операторы кроссинговера.

3.3.1. Упорядоченное представление.

В этом случае тур представляется списком из n – городов, где i -й элемент списка имеет номер от 1 до $n-i+1$. При этом используется базовый упорядоченный список городов L , который служит для ссылок упорядоченного представления. Фактически мы рассматривали этот метод кодирования решения в предыдущем разделе при решении задачи об укладке рюкзака.

Рассмотрим на конкретном примере тура $T = (1-2-4-3-8-5-9-6-7)$, для которого упорядоченный список $L = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$. Тогда данный тур при этом упорядоченном списке представляется следующим списком ссылок $e = (1\ 1\ 2\ 1\ 4\ 1\ 3\ 1\ 1)$, который интерпретируется следующим образом. Здесь жирным курсивом выделен текущий указатель в списке e .

Первый номер из списка e равен 1, поэтому берём в тур 1-й город из базового списка L , удаляем его из L и сдвигаем указатель по e . Тогда:

Тур $T_1 = (1)$, базовый $L_1 = (2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$, указатель $e = (1\ 1\ 2\ 1\ 4\ 1\ 3\ 1\ 1)$.

Следующий номер по указателю e также равен 1, поэтому снова берём в тур 1-й город из базового списка L , удаляем его из L и сдвигаем указатель по e . Тогда:

текущий тур $T_2=(1-2)$, $L_1 = (3\ 4\ 5\ 6\ 7\ 8\ 9)$ и указатель в e сдвигается на третью позицию $e = (1\ 1\ 2\ 1\ 4\ 1\ 3\ 1\ 1)$.

Следующий номер списка e равен 2, поэтому мы берем и удаляем 2-й город из текущего базового списка L и добавляем его в тур и передвигаем указатель по e . Имеем:

текущий тур $T_3=(1-2-4)$, $L = (3\ 5\ 6\ 7\ 8\ 9)$, $e = (1\ 1\ 2\ 1\ 4\ 1\ 3\ 1\ 1)$.

Продолжая этот процесс в результате декодирования, мы по данному коду $e=(1\ 1\ 2\ 1\ 4\ 1\ 3\ 1\ 1)$, базовому списку $L = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$ построим тур $T = (1-2-4-3-8-5-9-6-7)$. Основное преимущество упорядоченного представления в том, что в этом случае работает классический кроссинговер (над векторами целых чисел). То есть для двух допустимых решений кроссинговер производит два допустимых решения – потомка.

Например, для родителей

$e_1 = (1\ 1\ 2\ 1\ | 4\ 1\ 3\ 1\ 1)$ и $e_2 = (5\ 1\ 5\ 5\ | 5\ 3\ 3\ 2\ 1)$,

которые соответствуют турам

$T_1=(1-2-4-3-8-5-9-6-7)$ и $T_2=(5-1-7-8-9-4-6-3-2)$,

имеем следующих потомков при скрещивании

$O_1 = (1\ 1\ 2\ 1\ 5\ 3\ 3\ 2\ 1)$ и $O_2 = (5\ 1\ 5\ 5\ 4\ 1\ 3\ 1\ 1)$,

которые представляют туры

$T_3=(1-2-4-3-9-7-8-6-5)$ и $T_4=(5-1-7-8-6-2-9-3-4)$.

Очевидно, что частичные туры слева от точки кроссинговера не изменяются, в тоже время частичные туры справа от нее рвутся случайным образом (сохраняя корректность решения). К сожалению, машинные эксперименты по решению ЗК на основе этого представления решений с использованием классического кроссинговера показывают посредственные результаты [17].

3.3.2. Представление соседства

В этом случае тур представляется списком соседних городов. Город j находится в позиции i если и только если в туре после города i посещается город j , что показано на рис.3.2.

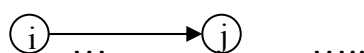


Рис.3.2. Следование городов в туре

Например, вектор $n_1=(2\ 4\ 8\ 3\ 9\ 7\ 1\ 5\ 6)$ представляет следующий тур $T_1=(1-2-4-3-8-5-9-6-7)$. При этом любой тур имеет единственное представление списком соседства. Однако некоторые «списки соседей» могут представлять «неправильные» туры. Например, список соседства $n_2=(2\ 4\ 8\ 1\ 9\ 3\ 5\ 7\ 6)$ содержит «частичный» тур $T_2=(1-2-4-1)$. Поэтому представление списком соседей не поддерживает классический оператор кроссинговера, поскольку при перестановке частей списков могут получаться неправильные, («частичные») туры. В этом случае необходим некоторый алгоритм восстановления полного тура.

Для данного способа кодирования решения были предложены и исследованы 3 основных оператора кроссинговера: 1) обмен ребер (дуг) графа; 2) обмен подтуров; 3) эвристический кроссинговер [17]. Далее мы рассмотрим их детально.

1) Обмен ребер.

Этот тип кроссинговера строит потомка (случайным) выбором ребра (пары городов $i-j$) из первого родителя, затем выбором соответствующего ребра из второго родителя и т.д. Оператор наращивает тур выбором ребер из различных родителей. Если новое ребро (взятое у одного из родителей) образует преждевременный (частичный) цикл в текущем (ещё не полном)

туре, то оператор выбирает (случайно) вместо этого ребра одно из оставшихся ребер, которое не дает преждевременного цикла.

Например, для 2-х родителей

$n_1 = (2\ 3\ 8\ 7\ 9\ 4\ 1\ 5\ 6)$, представляющего тур $T_1 = (1-2-3-8-5-9-6-4-7-1)$,

и $n_2 = (7\ 5\ 1\ 6\ 9\ 2\ 8\ 4\ 3)$ тура $T_2 = (1-7-8-4-6-2-5-9-3-1)$ получаем

следующего потомка $\sigma_1 = (2\ 5\ 8\ 7\ 9\ 4\ 3\ 1\ 6)$. Здесь произведен обмен (случайный) ребер $3 \leftrightarrow 5$ во второй позиции, далее при попытке обмена $8 \leftrightarrow 1$ в седьмой позиции возникает конфликт (два ребра входят в 8), поэтому случайно выбирается 3 в позиции 7 (из оставшихся 6, 1, 3), 1 – в позиции 8 и 6 в позиции 9. В результате порождается потомок σ_1 , который соответствует туру $T_3 = (1-2-5-9-6-4-7-3-8-1)$.

2) Обмен подтуров.

Этот оператор строит потомки путём выбора (случайной длины) подтура из первого родителя, затем выбора подтура (опять случайной длины) из второго родителя и их обмена. Как и ранее в случае конфликта оператор случайно выбирает другое ребро (город) из не вошедших в построенный тур.

3) Эвристическое скрещивание.

Данный оператор строит потомка случайным выбором города в качестве начальной точки для формирования тура. Затем сравниваются два ребра, исходящие из этого города, имеющих в двух родителях, и из них выбирается лучшее с меньшей стоимостью. Полученный город (в него входит выбранное ребро на предыдущем этапе), используется в качестве начальной точки при выборе следующего ребра и т.д. Как и ранее в случае возникновения преждевременного цикла, следующий город выбирается случайно из городов, еще не вошедших в текущий тур. Известна следующая модификация этого оператора.

Если оптимальное (с меньшей стоимостью) ребро дает преждевременный цикл в потомке, то проверяется альтернативное ребро (с

большей стоимостью) на генерацию преждевременного цикла. Если это ребро не генерирует цикл, то оно включается в тур, иначе выбирается минимальное из q ребер (где q – параметр пула выбора) случайно выбранных из оставшихся городов.

Преимущество этого кодирования в том, что в этом случае анализ схем (шаблонов) можно проводить по аналогии с двоичным случаем. Например, схема (* * * 3 * 7 * * *) представляет множество всех туров с ребрами (4-3) и (6-7). Однако основной недостаток этого представления в весьма посредственных результатах тестовых задач для всех 3-х рассмотренных операторов. Кроссинговер обмена ребер часто разрывает хорошие туры при обмене ребер родителей. Кроссинговер обмена подтуров даёт лучшие результаты, чем предыдущий так как “отношение разрыва” здесь меньше. Эвристический оператор даёт несколько лучшие результаты, за счет учета стоимости исходящих ребер и локального выбора лучшего варианта из двух возможных. Однако эксперименты показывают также “не выдающиеся” результаты [11,17].

3.3.3. Представление путей.

Это представление, возможно, самое естественное для тура. Например, тур (5-1-7-8-9-4-6-2-3) предоставляется просто упорядоченным списком (5 1 7 8 9 4 6 2 3) городов, входящих в тур.

Для данного предоставления были определены и исследованы три типа кроссинговера [2, 11, 17]:

- 1) частично соответствующий (partially-mapped - PMX);
- 2) упорядоченный ОК (order - OX);
- 3) циклический ОК (cycle - CX).

Рассмотрим их по порядку.

Частично соответствующий ОК (PMX) строит потомок путем выбора последовательности тура из одного родителя и сохранения порядка и позиции городов из другого родителя насколько это возможно. Подпоследовательность из тура выбирается случайно с помощью двух “секущих” точек, которые служат границами для операции обмена. Например, для родителей

$$P_1 = (1\ 2\ 3\ |\ 4\ 5\ 6\ 7\ |\ 8\ 9) \text{ и } P_2 = (4\ 5\ 2\ |\ 1\ 8\ 7\ 6\ |\ 9\ 3)$$

потомок строится следующим образом.

Сначала производим обмен выделенными подтурами и в результате получаем $T_1 = (X\ X\ X\ |\ 1\ 8\ 7\ 6\ |\ X\ X)$ и $T_2 = (X\ X\ X\ |\ 4\ 5\ 6\ 7\ |\ X\ X)$, где ‘X’ означает еще незаполненную позицию (допускающую произвольное значение). Этот обмен определяет также отображение

$1 \leftrightarrow 4, 8 \leftrightarrow 5, 7 \leftrightarrow 6$. Далее заполняем (вместо ‘X’) города из исходных родителей, для которых нет конфликтов (не образуются преждевременный цикл):

$$O_1 = (X\ 2\ 3\ |\ 1\ 8\ 7\ 6\ |\ X\ 9), O_2 = (X\ X\ 2\ |\ 4\ 5\ 6\ 7\ |\ 9\ 3).$$

Далее первый ‘X’ в O_1 , заменяем на ‘4’ согласно отображению $1 \leftrightarrow 4$. Аналогично второй ‘X’ в потомке O_1 заменяется ‘5’, и во втором потомке O_2 оставшиеся неопределенные позиции ‘X’ заменяются соответственно на 1 и 8. В результате получаем два потомка: $O_1 = (4\ 2\ 3\ |\ 1\ 8\ 7\ 6\ |\ 5\ 9)$ и $O_2 = (1\ 8\ 2\ |\ 4\ 5\ 6\ 7\ |\ 9\ 3)$.

Упорядоченный ОК строит потомок выбором подтура из одного родителя и сохранением относительного порядка городов из другого родителя. Например, для родителей

$$P_1 = (1\ 2\ 3\ |\ 4\ 5\ 6\ 7\ |\ 8\ 9), \quad P_2 = (4\ 5\ 2\ |\ 1\ 8\ 7\ 6\ |\ 9\ 3)$$

потомок строится следующим образом.

Сначала сегменты между двумя секущими точками копируются в потомки:

$$O_1 = (X\ X\ X\ |\ 4\ 5\ 6\ 7\ |\ X\ X), \quad O_2 = (X\ X\ X\ |\ 1\ 8\ 7\ 6\ |\ X\ X).$$

Далее, в первом родителе начиная со второй секущей точки копируются города из другого родителя, пропуская уже присутствующие в построенном подтуре. По достижению конца списка этот процесс продолжается с первой позиции и до первой точки сечения (по кольцу).

Для нашего примера после второй точки сечения во втором родителе мы имеем следующую последовательность $9 - 3 - 4 - 5 - 2 - 1 - 8 - 7 - 6$. Удаляем из нее города 4, 5, 6, 7 поскольку они уже есть в первом потомке и в результате получаем последовательность $9 - 3 - 2 - 1 - 8$. Ее мы помещаем в первый потомок, начиная со второй точки сечения (по кольцу) и получаем потомок

$O_1 = (2\ 1\ 8\ |\ 4\ 5\ 6\ 7\ |\ 9\ 3)$. Аналогично получаем второго потомка

$O_2 = (3\ 4\ 5\ |\ 1\ 8\ 7\ 6\ |\ 9\ 2)$.

Оператор кроссинговера ОХ использует то, что при представлении тура прежде всего важен порядок годов, например, два тура $(9-3-4-5-2-1-8-7-6)$ и $(4-5-2-1-8-7-6-9-3)$ идентичны.

Циклический ОК строит потомки таким образом, что каждый город вместе со своей позицией идет от одного из родителей. Например, для родителей

$P_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$ и

$P_2 = (4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5)$

сначала получаем путем выбора первого города из родителя $O_1 = (1\ X\ X\ X\ X\ X\ X\ X\ X)$. Выбор следующего города определяет текущая позиция второго родителя. В нашем примере это город 4, что дает $O_1 = (1\ X\ X\ 4\ X\ X\ X\ X\ X)$. Город 4 в свою очередь имплицитно дает город 8 (из второго родителя), что дает $O_1 = (1\ X\ X\ 4\ X\ X\ X\ 8\ X)$.

Аналогично получаем города 3, 2 в $O_1 = (1\ 2\ 3\ 4\ X\ X\ X\ 8\ X)$. Здесь мы вынуждены прервать этот процесс, так как выбор $2 \rightarrow 1$ ведет к преждевременному циклу. Поэтому оставшиеся города берутся из другого родителя P_2 (с сохранением порядка) и в результате получаем потомки: O_1

$= (1\ 2\ 3\ 4\ 7\ 6\ 9\ 8\ 5)$ и $O_2 = (4\ 1\ 2\ 8\ 5\ 6\ 7\ 3\ 9)$. Таким образом, оператор ОК СХ сохраняет абсолютные позиции потомков и родителей.

3.3.4. Матричное представление

Опубликовано достаточно много работ [2,11,12,17], где для решения задачи коммивояжера используется представление тура в виде двоичной матрицы, элементы которой $m_{ij}=0,1$. При этом применяются два основных подхода, которые используют: 1) матрицу инцидентий; 2) матрицу предшествования, которые мы рассмотрим ниже.

3.3.4.1. Матрица смежности

В матрице смежности элемент $m_{ij}=1$ в том и только случае, если в туре после города i посещается город j (в графе есть ребро от вершины i в вершину j). Например, табл.3.1 содержит матрицу смежности для тура $T_1=(1-2-4-3-8-6-5-7-9)$ и табл.3.2 – матрицу смежности для тура $T_2=(1-4-3-6-5-7-2-8-9)$.

Таблица 3.1

	1	2	3	4	5	6	7	8	9
1	0	1	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	0	0
3	0	0	0	0	0	0	0	1	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	1	0	0	0
9	1	0	0	0	0	0	0	0	0

Отметим, что в этом случае каждая строка и столбец матриц содержат одну 1. Для матрицы смежности можно использовать одно- или двух-точечный кроссинговер, где обмен производится, например, столбцами. Но в этом случае необходим дополнительный алгоритм восстановления, который позволяет восстанавливать полученные потомки до полных туров.

Таблица 3.2

	1	2	3	4	5	6	7	8	9
1	0	0	0	1	0	0	0	0	0
2	0	0	0	0	0	0	0	1	0
3	0	0	0	0	0	1	0	0	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0
7	0	1	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	1
9	1	0	0	0	0	0	0	0	0

Рассмотрим, этот подход на примере двухточечного вертикального кроссинговера с точками скрещивания 2 и 6. При этом производится обмен столбцами (3,4,5,6) матриц смежности (таблиц 3.1 и 3.2). В результате получаем промежуточный результат в виде матриц, которые представлены табл.3.3 и табл.3.4. Обе матрицы не представляют правильных решений, но заметим, что суммарное число 1 в каждой из этих промежуточных матриц правильное (9 единиц).

Таблица 3.3

	1	2	3	4	5	6	7	8	9
1	0	1	0	1	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	1	0	1	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0
7	0	0	0	0	0	0	0	0	1
8	0	0	0	0	0	0	0	0	0
9	1	0	0	0	0	0	0	0	0

Таблица 3.4

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	0	0	0
2	0	0	0	1	0	0	0	1	0
3	0	0	0	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0
6	0	0	0	0	1	0	0	0	0
7	0	1	0	0	0	0	0	0	0
8	0	0	0	0	0	1	0	0	1
9	1	0	0	0	0	0	0	0	0

На первом шаге алгоритма восстановления передвигаем 1 в матрице таким образом, чтобы каждая строка и столбец имели одну единицу. Например, в матрице табл.3.3 первая строка имеет две 1 (вместо одной). Поэтому «передвинем» $m_{14}=1$ в позицию $m_{84}=1$, а элемент $m_{24}=1$ в $m_{34}=1$, аналогично $m_{86}=1$ в $m_{16}=1$. В результате после выполнения первого этапа алгоритма восстановления получаем правильный тур для первого потомка $T_3=(1-2-8-4-3-6-5-7-9)$, в то время как второй потомок содержит два под тура $T_4=(1-6-5-7-2-8-9)$ (3-4). Поэтому на втором этапе алгоритма восстановления обрабатываем только второй потомок. При этом необходимо разорвать частичные подтуры и объединить их в единый правильный тур. Это можно сделать, например, с помощью ребра 2-4, которое присутствует у одного из родителей. В результате получаем полный тур для второго потомка $T_5=(1-6-5-7-2-4-3-8-9)$.

3.3.4.2. Матрица предшествования

Рассмотрим представление тура в виде двоичной матрицы предшествования [17]. В этом случае элемент матрицы $m_{ij}=1$ равен 1, если и только если город i предшествует (встречается в туре раньше) городу j . В противном случае $m_{ij}=0$. Отметим, что здесь имеется в виду не только непосредственное предшествование (город i связан непосредственно с городом j) но и более «глубокое». Например, тур $T_3=(3-1-2-8-7-4-6-9-5)$ представляется матрицей бинарной матрицей предшествования, которая показана в табл.3.5. Здесь элементы главной диагонали $m_{ii}=0$ и $m_{ij}=1$, если i -й город предшествует в туре j -му городу. Например, в первой строке город 1 предшествует в туре городам 2, 4, ..., 9, но не предшествует городу 3.

Таблица 3.5

	1	2	3	4	5	6	7	8	9
1	0	1	0	1	1	1	1	1	1
2	0	0	0	1	1	1	1	1	1
3	1	1	0	1	1	1	1	1	1
4	0	0	0	0	1	1	0	0	1
5	0	0	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	1
7	0	0	0	1	1	1	0	0	1
8	0	0	0	1	1	1	1	0	1
9	0	0	0	0	1	0	0	0	0

В этом представлении тура матрица M размера $n \times n$ обладает следующими свойствами:

1. число 1 в матрице равно точно $\frac{n(n-1)}{2}$;
2. $m_{ij}=0$ для всех $1 \leq i \leq n$;
3. если $m_{ij}=1$ и $m_{jk}=1$, то $m_{ik}=1$.

Если число единиц в матрице меньше чем $\frac{n(n-1)}{2}$, а два других требования выполняются, то города частично упорядочены. Это означает, что мы можем получить матрицу (по крайней мере одним способом), которая представляет правильный тур. На этой форме представления вводятся два новых генетических оператора кроссинговера: пересечение и объединение.

Оператор пересечения основан на том, что побитовое пересечение матриц дает матрицу, где:

- 1) число единиц не больше $\frac{n(n-1)}{2}$;

Таблица 3.7

	1	2	3	4	5	6	7	8	9
1	0	1	1	0	1	1	1	1	1
2	0	0	1	0	1	1	1	1	1
3	0	0	0	0	1	0	0	0	0
4	1	1	1	0	1	1	1	1	1
5	0	0	0	0	0	0	0	0	0
6	0	0	1	0	1	0	0	0	1
7	0	0	1	0	1	1	0	0	1
8	0	0	1	0	1	1	1	0	1
9	0	0	1	0	1	0	0	0	0

Таблица 3.8

	1	2	3	4	5	6	7	8	9	N1
1	0	1	1	0	1	1	1	1	1	7
2	0	0	1	0	1	1	1	1	1	6
3	0	0	0	0	1	0	0	0	0	1
4	0	0	0	0	1	1	1	1	1	5
5	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	1	1
7	0	0	0	0	0	0	0	0	1	1
8	0	0	0	0	0	0	0	0	1	1
9	0	0	0	0	0	0	0	0	0	1
N1	0	1	2	0	4	3	3	3	6	

На заключительном этапе выполнения этого оператора выбирается один из родителей и в промежуточную матрицу (полученную после пересечения) добавляются некоторые единицы (чтобы их число было равно $\frac{n(n-1)}{2}$) путем анализа сумм строк и столбцов, которые восстанавливают полный тур с учетом соответствующего родителя. В табл.8 для удобства показано число единиц по строкам и столбцам матрицы, которое, например, позволяет города на подтуры или в некотором смысле эквивалентные группы. Анализ числа 1 в строках показывает, что тур должен начинаться с (1-2-4), остальные города можно разбить на группы (3,6,7,8) и (5,9). Анализ числа 1 по столбцам позволяет «выстроить» подтур (3-5-9), а для остальных трех городов можно выбрать, например, порядок (8-7-6). В результате получаем для потомка полный тур (1-2-4-8-7-6-3-5-9), который представлен матрицей табл.3.9, которая может быть получена из предыдущей табл.3.8.

Оператор объединения основан на том, что подмножество бит из одной матрицы может быть комбинировано с подмножеством бит из другой матрицы без потери свойства быть туром, если эти подмножества имеют пустое пересечение. Оператор разбивает множество городов на две непересекающиеся группы. При этом для первой группы городов копируются биты из первой матрицы, а для второй группы копируются биты из второй матрицы.

На заключительном этапе матрица достраивается путем анализа сумм для строк и столбцов аналогично операции пересечения. Например, два родителя P_1 и P_2 , которые представлены табл.3.6 и табл.3.7, и разбиения городов $\{1, 2, 3, 4\}$ и $\{5, 6, 7, 8, 9\}$ порождают матрицу табл.3.10.

Таблица 3.9

	1	2	3	4	5	6	7	8	9
1	0	1	(1)	1	1	1	1	1	1
2	0	0	1	(1)	1	1	1	1	1
3	0	0	0	0	1	0	0	0	(1)
4	0	0	(1)	0	1	1	1	1	1
5	0	0	0	0	0	0	0	0	(1)
6	0	0	(1)	0	(1)	0	0	0	1
7	0	0	(1)	0	(1)	(1)	0	0	1
8	0	0	(1)	0	(1)	(1)	(1)	0	1
9	0	0	0	0	0	0	0	0	0

Таблица 3.10

	1	2	3	4	5	6	7	8	9
1	0	1	1	1	X	X	X	X	X
2	0	0	1	1	X	X	X	X	X
3	0	0	0	1	X	X	X	X	X
4	0	0	0	0	X	X	X	X	X
5	X	X	X	X	0	0	0	0	0
6	X	X	X	X	1	0	0	0	1
7	X	X	X	X	1	1	0	0	1
8	X	X	X	X	1	1	1	0	1
9	X	X	X	X	1	0	0	0	0

Таким образом, для решения задачи коммивояжера, в основном, используется не классическое двоичное представление, а более сложные списковые и матричные структуры, которых разработано достаточно много (каждый уважающий себя автор, как правило, предлагает свой способ кодирования потенциального решения и проблемно-ориентированные генетические операторы). Цель данного раздела, в первую очередь, заключалась в представлении некоторых нестандартных способов кодирования особей и проблемно-ориентированных генетических операторов. Подробнее данная проблема освещена в специальной литературе. В следующем разделе, где рассматриваются различные модификации генетических алгоритмов, фактически эта тема будет продолжена.

3.5. Контрольные вопросы к разделу 3

1. При решении каких задач комбинаторной оптимизации может быть использован простой ГА с двоичным кодированием хромосом?
2. Какие модификации необходимы для эффективного использования простого ГА для решения задачи укладки рюкзака?
3. Какие виды штрафных функций могут быть использованы в фитнес-функции при решении задачи укладки рюкзака?
4. Выполните программную реализацию простого ГА на одном из языков программирования для решения задачи укладки рюкзака с введением в фитнес-функцию штрафной функции. Исследуйте эффективность ГА в зависимости от вида штрафной функции.
5. В чем суть алгоритма восстановления при решении задачи укладки рюкзака?

6. Выполните программную реализацию простого ГА на одном из языков программирования для решения задачи укладки рюкзака с использованием алгоритма восстановления.
7. Выполните программную реализацию простого ГА на одном из языков программирования для решения задачи укладки рюкзака с алгоритма декодирования.
8. Как может быть использован простой ГА с двоичным кодированием хромосом для решения задачи о покрытии?
9. Почему неэффективно двоичное кодирование хромосомы при решении задачи коммивояжера?
10. Опишите основные виды недвоичного представления хромосомы для задачи коммивояжера.
11. Опишите “представление соседства” и проблемно-ориентированные операторы кроссинговера: обмен ребер, обмен туров, эвристический кроссинговер.
12. Как может быть выполнен оператор мутации на представлении соседства?
13. Опишите “упорядоченное представление” и какой тип оператора кроссинговера может на нем использоваться?
14. Опишите “представление путей” и проблемно-ориентированные операторы кроссинговера: частично соответствующей ОК (PMX), упорядоченный ОК (OX), циклический ОК (CX).
15. Выполните программную реализацию ГА для решения задачи коммивояжера для одного из приведенных недвоичных представлений.
16. Исследуйте эффективность ГА в зависимости от представления и проблемного ориентированного оператора кроссинговера.
17. Какие двоичные матрицы можно использовать для представления тура?
18. Опишите соответствующие операторы кроссинговера для матрицы смежности.

19. Чем отличается матрица предшествования от матрицы смежности и как можно реализовать операторы кроссинговера на ней?
20. Придумайте свой способ кодирования (представления) полного тура для задачи коммивояжера и соответствующие генетические операторы.

ЧАСТЬ 2

МОДИФИКАЦИИ И ОБОБЩЕНИЯ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ

4. МОДИФИКАЦИИ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ

Мы изучили в предыдущих разделах, в основном, классическую реализацию простого ГА. Далее рассмотрим различные модификации и обобщения для каждого функционального блока основной блок-схемы ГА рис. 1.1 (которые частично изложены в [22]).

4.1. Создание исходной популяции

В настоящее время наиболее известными и применяемыми на практике являются 3 способа создания исходной популяции (начального множества решений).

Стратегия "одеяла" – формирование полной популяции, содержащей все возможные решения. Например, для n -разрядной хромосомы мы имеем всего 2^n вариантов решений, которые составляют полную популяцию.

Стратегия "дробовика" - генерация достаточно большого случайного подмножества решений.

Стратегия фокусировки - генерация подмножества решений, включающих разновидности одного решения .

Первый подход (стратегия «одеяла») практически не реализуем даже для задач средней размерности, вследствие больших вычислительных затрат. Заметим, что при этом подходе начальная популяция не может развиваться, т.к. в ней уже содержатся *все* возможные решения.

Третий способ используется тогда, когда есть предположение, что некоторое решение является вариацией известного значения. В этом случае время поиска оптимального решения существенно сокращается, т.к. алгоритм начинает работу в окрестности оптимума.

Чаще всего применяют второй способ. В этом случае в результате эволюции есть возможность перейти в другие подобласти поиска и каждая из них имеет сравнительно небольшое пространство поиска.

В целом эффективность ГА, качество решения и дальнейшая эволюция в значительной степени определяются структурой и качеством начальной популяции.

На практике часто применяют комбинацию второго и третьего способов. Сначала определяются особи с высокими значениями целевой функции, а затем случайно формируются начальные решения в этих подобластях.

4.2. Отбор родителей (селекция)

Оператор отбора S порождает промежуточную популяцию \tilde{P}^t из текущей популяции P^t путем отбора и генерации новых копий особей.

$$P^t \xrightarrow{S} \tilde{P}^t$$

При отборе конкурентно-способных особей используют целевую (fitness) функцию, как единственно доступный источник информации о качестве решения. Но различные методы отбора родителей по-разному

используют эту информацию. Далее мы рассмотрим наиболее распространенные методы отбора родителей.

4.2.1. Пропорциональный отбор (метод "рулетки")

Данный вид отбора чаще всего используется на практике. При этом особи (решения) отображаются в отрезки линии (или сектора рулетки) таким образом, что их размер пропорционален значению целевой функции для данной особи. Это показано в следующем примере, представленном в табл.4.1

Таблица 4.1.

Номер особи	1	2	3	4	5	6	7	8	9	10	11
Значение ЦФ	2,0	1,8	1,6	1,4	1,2	1,0	0,8	0,6	0,4	0,2	0,0
Вероятность выбора	0,18	0,16	0,15	0,13	0,11	0,09	0,07	0,06	0,03	0,02	0,0

Здесь на отрезке $[0,1]$ для каждой особи строятся отрезки, длины которых пропорциональны вероятностям выбора особей, как это показано на рис.4.1.

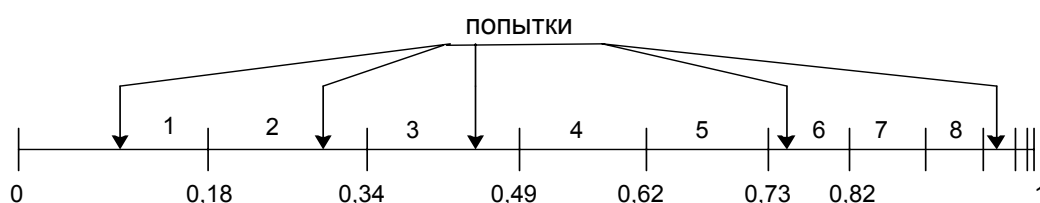


Рис.4.1. Вероятность выбора особей методом рулетки

Далее случайно генерируются числа из $[0,1]$ и в промежуточную популяцию выбираются те особи, в "чей отрезок" попадают эти случайные числа. Таким образом, каждая попытка представляет собой случайное

число из отрезка $[0,1]$, в результате чего выбирается та особь, соответствующая выбранному отрезку. В данном примере в результате пяти попыток были выбраны в промежуточную популяцию особи 1, 2, 3, 6 и 9.

Данный метод, не смотря на частое применение, имеет следующие недостатки:

- зависимость от положительных значений целевой функции (функция должна для всех особей принимать положительное значение);
- метод может использоваться (без модификации) только в задачах максимизации (но не минимизации) функции;
- простое добавление очень большой константы к целевой функции, может свести способ отбора практически к случайному выбору;
- особи с очень маленькими значениями фитнес-функции слишком быстро исключаются из популяции, что может привести к преждевременной сходимости ГА.

Для устранения этих недостатков используют масштабирование оценок значений целевой функции относительно минимального значения в популяции (см. раздел 4.2.8) или используется метод ранжирования (раздел 4.2.2).

Очевидно, что проблема минимизации может быть сведена к задаче максимизации функции и обратно. Поэтому в некоторых реализациях ГА метод рулетки применяется и для поиска минимума функции, что в практических задачах встречается чаще (минимизация затрат, расстояния, погрешности и т.д.)

Для приведенного примера можно привести следующую гистограмму вероятностей выбора особей, представленную на рис.4.2. Отметим, что здесь высота "ступенек" различна, в отличие от следующего метода выбора, для которого гистограмма представлена на рис.4.3.

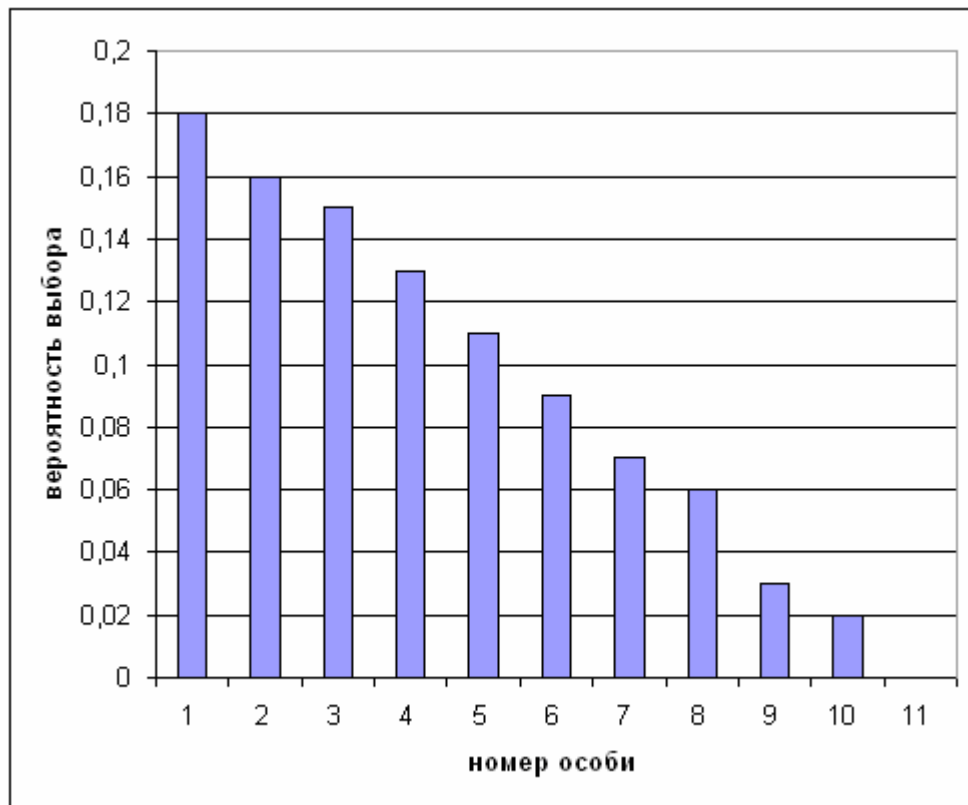


Рис.4.2. Ранжирование особей в операторе селекции методом асимметричного колеса рулетки

4.2.2. Ранжирование

При этом методе особи популяции сортируются (упорядочиваются) согласно значениям целевой функции. Отметим, что здесь вероятность отбора для каждой особи зависит только от ее позиции (номера) в этом упорядоченном множестве особей, а не от самого значения целевой функции. Этот метод отбора более устойчив, чем предыдущий. В основном, применяют линейное ранжирование, где вероятность отбора определяют в соответствии со следующим выражением:

$$P_s(a_i) = \frac{1}{N} \left(a - (a - b) \frac{i - 1}{N - 1} \right), \quad (4.1)$$

где $1 \leq a \leq 2$ выбирается случайным образом;

$$b = 2 - a;$$

N – мощность популяции;

i – номер особи в упорядоченном списке.

Для приведенного примера гистограмма вероятностей выбора особей методом ранжирования, представлена на рис.4.3. Отметим, что здесь высота "ступенек" (разность высот столбцов) одинакова, в отличие от предыдущего метода выбора.

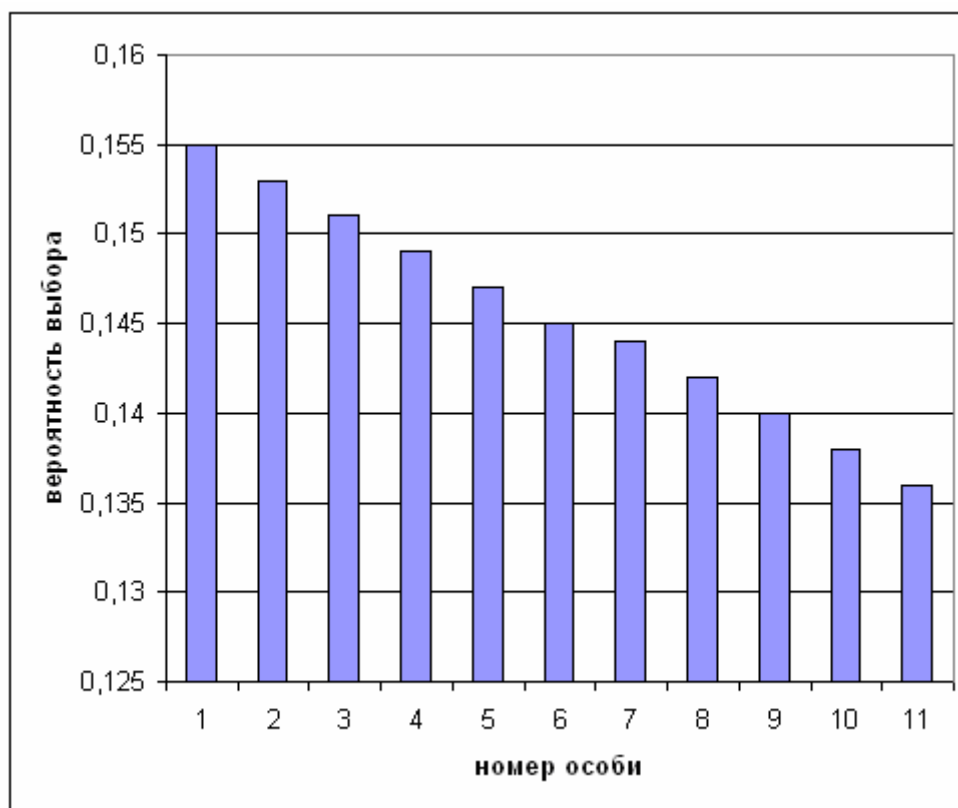


Рис.4.3. Сортировка особей в операторе селекции методом ранжирования с равным шагом

Иногда применяют нелинейное ранжирование. При этом вероятность отбора также определяется номером позиции в упорядоченном множестве особей, но вид функции может быть сложнее. Достоинством метода ранжирования является возможность его применения при поиске как максимумов, так и минимумов функций. Этот метод также не требует

масштабирования для предотвращения преждевременной сходимости в отличие от метода рулетки.

4.2.3. Равномерное ранжирование (случайный выбор)

Здесь вероятность выбора особи определяется выражением:

$$P_s(a_i) = \begin{cases} \frac{1}{\mu} & \text{при } 1 \leq i \leq \mu \\ 0 & \text{при } \mu < i \leq N \end{cases} \quad (4.2)$$

где $\mu \leq N$ – параметр метода.

4.2.4. Локальный отбор

Производится среди особей, которые находятся в некоторой ограниченной среде, где определено отношение соседства. Ранее, фактически, в качестве соседей каждой особи рассматривалась вся популяция. При этом в качестве соседей подразумевается множество партнеров для выполнения операции скрещивания.

Соседство можно определить по-разному. Далее рассмотрим типичные отношения соседства, используемые при локальном отборе.

1) *Линейное соседство:*

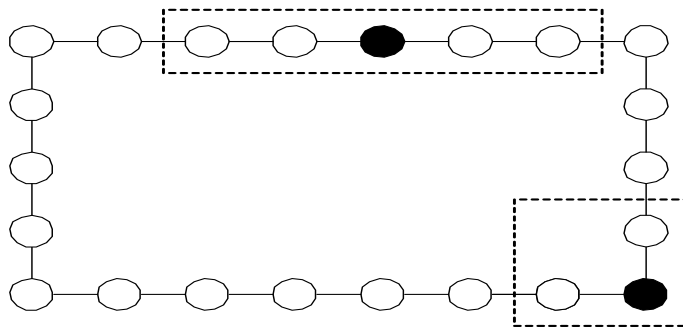


Рис.4.4. Линейное соседство

На рис.4.4 представлен пример линейного соседства. На практике рассматривают полную окрестность (на рис.4.4 вверху показана полная окрестность выделенного элемента с расстоянием $d=2$) и «полуокрестность» (в правом нижнем углу рисунка показана «полуокрестность» с расстоянием $d=1$).

2) Двумерное – четырехсвязное соседство.

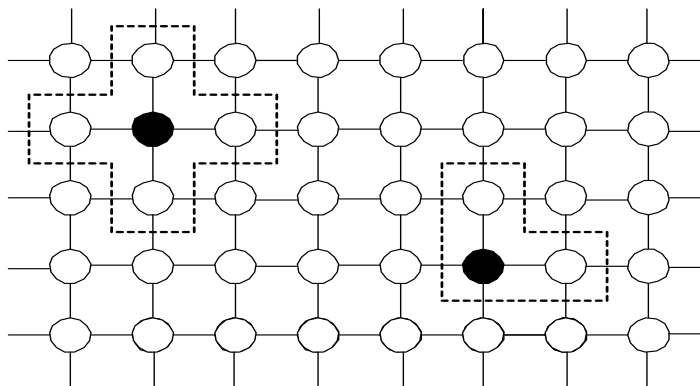


Рис.4.5. Соседство на четырехсвязной решетке

На рис.4.5 представлен пример соседства с 4-связностью (в левом верхнем углу показан полный "крест" выделенного элемента с расстоянием $d=1$, справа "полукрест" также с расстоянием $d=1$).

3) Двумерное – восьмисвязное соседство.

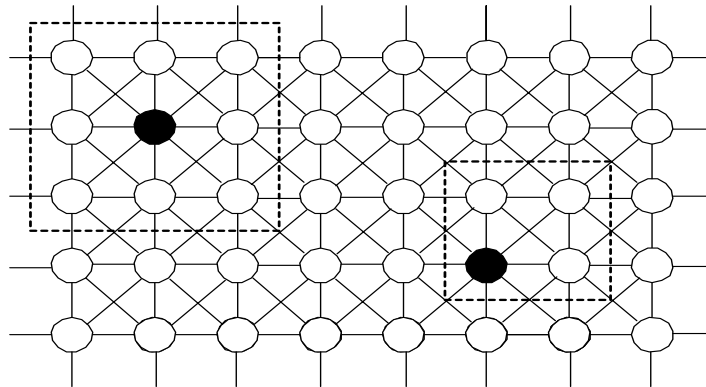


Рис.4.6. Соседство на восьмисвязной решетке

На рис.4.6 представлен пример соседства с 8-связностью (в левом верхнем углу показан полная "звезда" выделенного элемента с расстоянием $d=1$, а справа "полузвезда"). Для определения отношения можно также использовать гексагональную решетку с 6-связностью и даже 3-мерные структуры.

При отборе родителей на первом шаге производится отбор особей случайным образом, или одним из ранее рассмотренных способов. Далее, для каждой отобранной особи определяется множество локальных соседей и среди них выбирается партнер для выполнения операции скрещивания.

При наличии отношения соседства, между особями возникает эффект «изоляции расстоянием». Очевидно, что чем меньше соседство, тем больше «изоляция расстояния». Это ограничивает распространение новых решений в популяции.

Однако, из-за перекрытия соседних областей, распространение новых вариантов решений все же возможно. Мощность множества соседей определяет скорость распространения информации между особями популяции, способствуя либо быстрому распространению новых решений, либо сохранению имеющегося генофонда в популяции.

Часто при решении задачи требуется высокая изменчивость, которая поможет избежать преждевременной сходимости в районе локального оптимума. Обычно локальный отбор в малом окружении дает лучшие результаты, чем в большем окружении.

В малых и средних популяциях ($N < 100$) для локального отбора рекомендуется двумерная структура типа полужвезда с расстоянием $d = 1$.

При большом размере популяции ($N > 100$) лучше использовать большие расстояния $d > 1$ и 2-мерные структуры с соседством типа звезда.

4.2.5. Отбор на основе усечения

Предыдущие методы отбора в какой то степени заимствованы у (или имеют аналогию с) эволюции естественных популяций. Этот метод не имеет аналогов в естественной эволюции и обычно используется для больших популяций ($N > 100$). При этом сначала отбираемые особи упорядочиваются согласно их значениям целевой функции. Затем, в качестве родителей выбираются только лучшие особи. Далее, с равной вероятностью, среди них случайно выбирают пары которые производят потомков.

При этом методе используется параметр – порог отсечения T (иногда используется термин интенсивность отбора), показывающий долю (часть популяции), которая отбирается в качестве родителей. Обычно $10\% \leq T \leq 50\%$.

4.2.6. Турнирный отбор

В этом случае все особи популяции разбиваются на подгруппы размера m с последующим выбором в каждой из них особи с лучшим значением фитнес-функции. Параметром этой процедуры является размер

тура m , который принимает значения из диапазона $2 \leq m < N$. Используются два способа выбора: детерминированный и случайный. При детерминированном способе выбор выполняется с вероятностью, равной 1; в то время как при случайном методе выбор осуществляется с вероятностью меньше 1. Чаще всего популяция разбивается на подгруппы по 2-3 особи в каждой ($m=2,3$). Рис.4.7 иллюстрирует метод турнирной селекции для подгрупп, состоящих из 2-х особей.

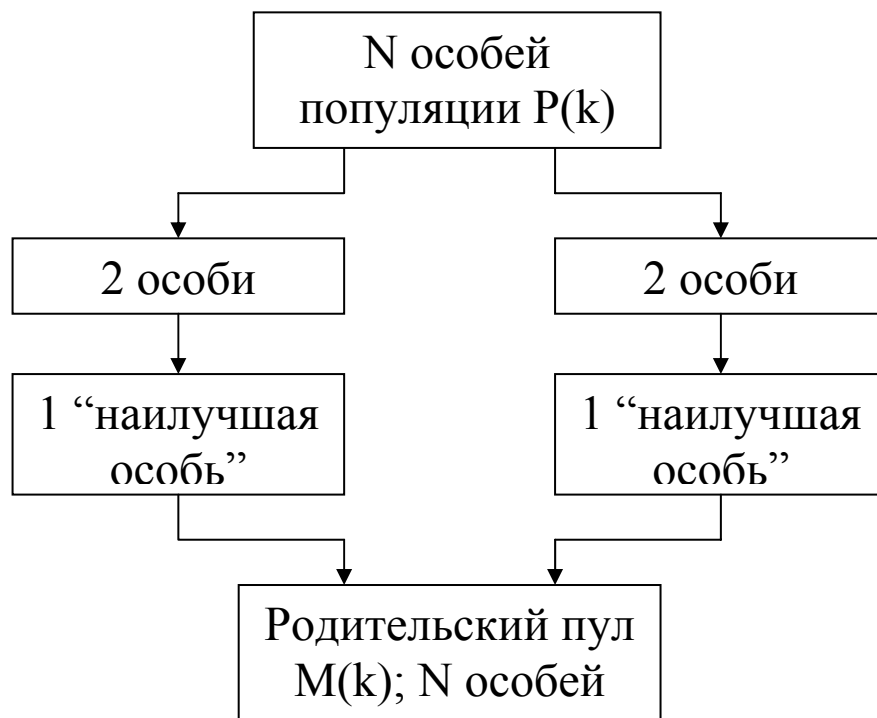


Рис. 4.7. Схема турнирной селекции

Турнирный метод может быть использован как при максимизации, так и при минимизации функции. Кроме этого, он легко распространяется на задачи многокритериальной оптимизации. Экспериментальные исследования показывают, что часто данный метод эффективней метода рулетки.

4.2.7. Метод Больцмана

В этом случае при отборе особей используется подход, который используется в известном методе оптимизации «моделирование отжига», где при управлении процессом поиска применяется «искусственная температура». Для этого вводится действительная переменная T , которая, начиная с некоторого достаточно большого значения, постепенно уменьшается (по определенному закону) и изменяет вероятность отбора особей. Вероятность отбора особи, имеющей значение фитнес-функции $f(i)$, определяется следующим образом

$$P_s(a_i) = \frac{1}{N} \left(\frac{e^{f(a_i)/T}}{\overline{e^{f(a)/T}}} \right), \quad (4.3)$$

где $\overline{e^{f(a)/T}}$ представляет среднее значение $e^{f(a)/T}$ по текущей популяции. Отметим, что с уменьшением температуры, разность значений $P_s(a_i)$ между худшими и лучшими особями увеличивается. Это позволяет на заключительном этапе сузить поиск в наиболее перспективной области пространства поиска, сохраняя при этом достаточную степень разнообразия в популяции. Показано, что для некоторых задач, этот метод отбора дает лучшие результаты, чем стандартный пропорциональный отбор типа «рулетка»[18].

4.2.8. Методы выбора пар для скрещивания

До сих пор мы, в основном, рассматривали методы отбора родителей в промежуточную популяцию. Далее из этой популяции необходимо выбрать пары особей для выполнения операции скрещивания. Основными методами выбора пар особей являются следующие.

Случайный выбор (панмиксия) родительской пары, при котором оба родителя случайным образом выбираются из всей промежуточной популяции. Следует отметить, что при этом любая особь может входить в несколько пар. Этот метод является универсальным для решения различных задач, но достаточно критичен к численности популяции, поскольку его эффективность снижается с ростом мощности популяции N .

Селективный выбор. Здесь родителями могут стать только те особи, значение целевой функции которых не меньше среднего значения по популяции при равной вероятности этих кандидатов составить брачную пару. Такой подход обеспечивает более быструю сходимость алгоритма, но неприемлем для мультимодальных задач (имеющих несколько экстремумов), для которых этот метод, как правило, быстро сходится к одному из решений. Это может привести к преждевременной сходимости к локальному экстремуму.

Часто используется также два следующих подхода: инбридинг и аутбридинг. В них формирование пары происходит на основе близкого или дальнего родства соответственно. Под родством обычно понимается расстояние между особями популяции в пространстве параметров. В связи с этим различают генотипный и фенотипный инбридинг и аутбридинг.

Инбридинг – здесь первый член пары выбирается случайно, а вторым является максимально близкая к нему особь.

Аутбридинг формирует пары из максимально далеких особей.

Комбинация этих подходов используется при решении многоэкстремальных задач.

4.2.9. Неявные методы отбора, основанные на масштабировании фитнес-функции

Кроме рассмотренных выше явных методов отбора часто применяются также и неявные методы, которые, как правило, сводятся к масштабированию фитнес-функции [23]. Масштабирование фитнес-функции производится обычно в следующих случаях: 1) для предотвращения преждевременной сходимости ГА; 2) на последнем этапе выполнения ГА когда в популяции сохраняется значительная неоднородность, однако среднее значение фитнес-функции ненамного отличается от максимального значения. Масштабирование позволяет избежать ситуаций, в которых средние и лучшие особи дают практически одинаковое количество потомков, что считается нежелательным явлением. Преждевременная сходимость соответствует ситуации, когда в популяции доминируют лучшие, но еще не оптимальные особи. Это особенно характерно для ГА, использующих при отборе родителей метод рулетки. При этом часто возникают ситуации, когда через несколько поколений популяция состоит в основном из копий лучшей особи. Поскольку в качестве исходной популяции часто используется небольшая случайная выборка из пространства возможных решений, то маловероятно, что именно эта лучшая особь соответствует оптимальному решению. Масштабирование фитнес-функции часто позволяет избежать доминирования неоптимальной особи, и тем самым предохранет ГА от преждевременной сходимости.

Масштабирование выполняется с помощью 3 основных преобразований фитнес-функций: 1) линейное, 2) сигма-отсечение и 3) степенное.

Линейное масштабирование сводится к линейному преобразованию фитнес-функции следующего вида

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

где a и b – константы, которые обычно подбираются так, чтобы среднее значение фитнес-функции после масштабирования было равно ее среднему значению до масштабирования, а максимальное значение фитнес-функции после преобразования было кратным ее среднему значению. Здесь F и F' соответствуют фитнес-функции до и после преобразования. При этом коэффициент кратности обычно выбирается в пределах от 1,2 до 2 и необходимо следить, чтобы новая фитнес-функция F' не принимала отрицательных значений.

Сигма отсечение основано на следующем преобразовании фитнес-функции

$$F' = F + (\bar{F} - c \cdot \sigma), \quad (4.4)$$

где \bar{F} означает среднее значение фитнес-функции по популяции, c – малое натуральное число (обычно от 1 до 5), а σ – стандартное отклонение по популяции. Если при этом значения F' получаются отрицательными, то они полагаются равными нулю.

При *степенном масштабировании* используется следующее преобразование

$$F' = F^k,$$

где k – число, обычно близкое к 3. В общем случае k подбирается эвристическим путем с учетом специфики задачи.

4.3. Операторы рекомбинации (скрещивания, кроссинговера)

Различают операторы двоичной и вещественной рекомбинации (скрещивания). Рассмотрим сначала более привычную (и уже знакомую нам), применяемую в простом ГА двоичную рекомбинацию.

4.3.1. Двоичная рекомбинация

4.3.1.1. Одноточечный кроссинговер

Здесь случайно выбирается точка скрещивания с вероятностью $P_c \approx 0.5$ и производится обмен фрагментами хромосом после точки скрещивания. Пример показан на рис.4.8.

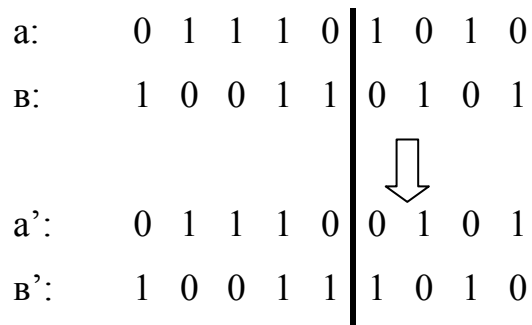


Рис.4.8. Одноточечный кроссинговер

4.3.1.2. Многоточечный кроссинговер

Разработаны различные обобщения классического одноточечного кроссинговера, которые наиболее полно представлены в [23]. При двухточечном кроссинговере потомки наследуют фрагменты хромосом родителей между двумя случайно выбранными точками скрещивания, как это, например, показано на рис. 4.9.

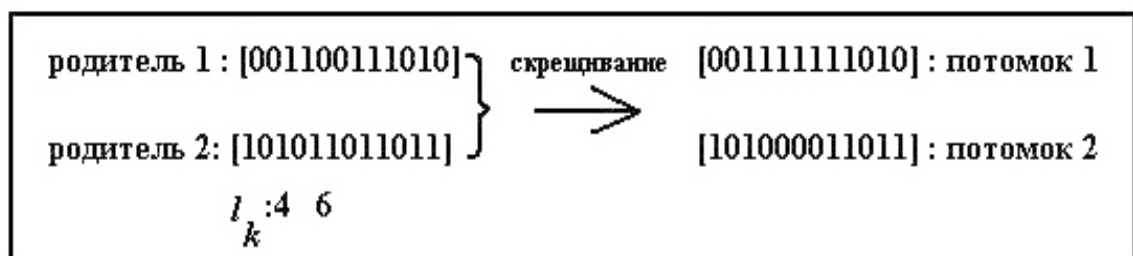


Рис. 4.9. Пример двухточечного кроссинговера

Графически это пример удобно представить в виде, показанном на рис.4.10.

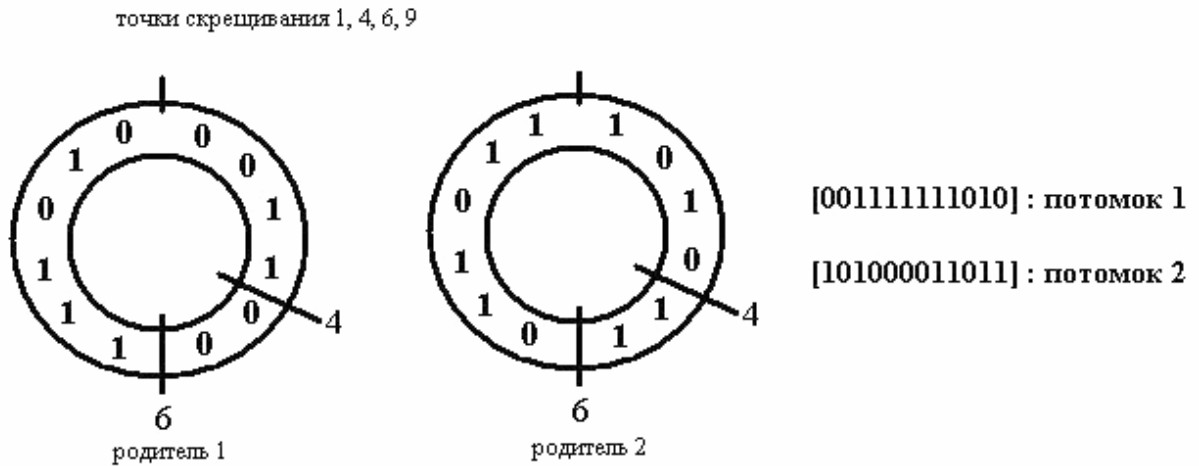


Рис. 4.10. Пример двухточечного кроссинговера

Многоточечный кроссинговер является обобщением предыдущих операторов на случай большего числа точек скрещивания. Например, трехточечный кроссинговер представлен на рис.4.11.



Рис.4.11. Пример трехточечного кроссинговера

Многоточечный кроссинговер с большим четным количеством точек скрещивания графически удобно представить для хромосом в виде «колец», что показано на рис.4.12.[23] При этом хромосома рассматривается как замкнутое кольцо, а точки скрещивания выбираются с равной вероятностью по всей его окружности.

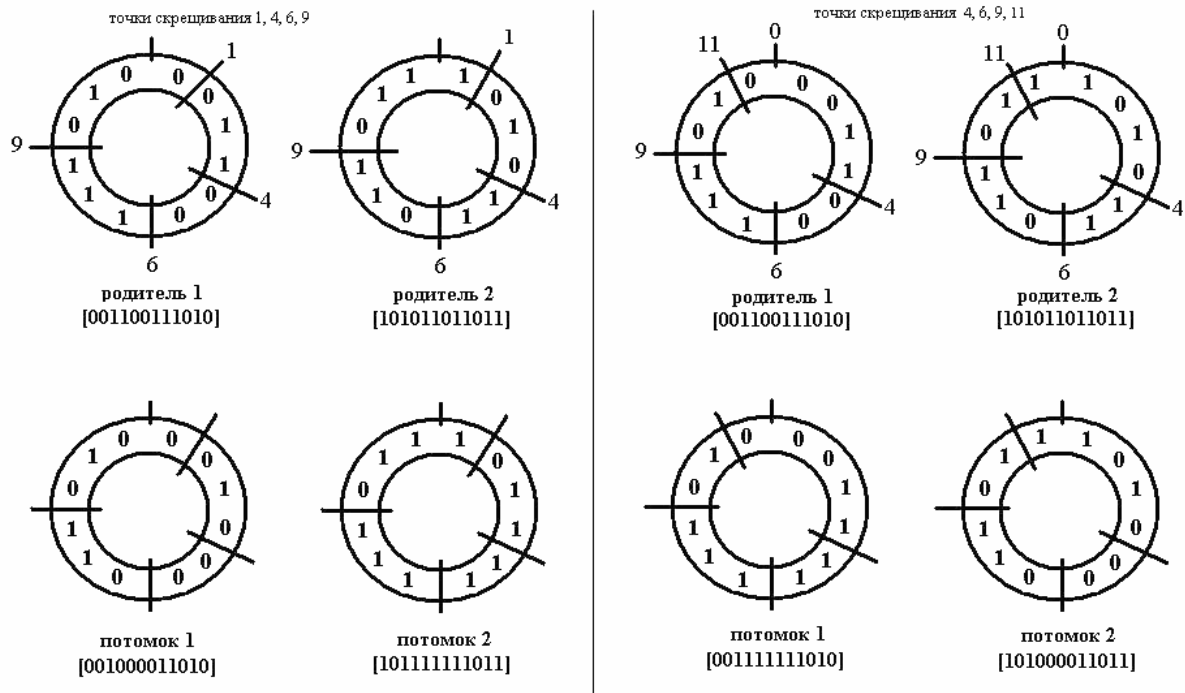


Рис.4.12. Пример четырехточечного кроссинговера

Кроссинговер с нечетным количеством точек скрещивания можно представить таким же способом, если добавить дополнительную точку скрещивания в нулевой позиции.

4.3.1.3. Однородный кроссинговер

Этот вид скрещивания радикально отличается от предыдущих видов. Здесь каждый ген потомка создается путем копирования соответствующего гена из первого или второго родителя, то есть каждая позиция потенциально является точкой кроссинговера.

Для этого случайным образом генерируется двоичная маска кроссинговера той же длины (с тем числом бит), что у хромосом родителей. Четность бита маски показывает родителя, из которого копируется ген потомка. Для определенности допустим, что 1 соответствует первому родителю, а 0 – второму. На рис.4.13 показана схема выполнения этого типа кроссинговера на конкретном примере.

Каждый бит потомка копируется из 1-го или 2-го родителя в соответствии со значением этого бита маски.

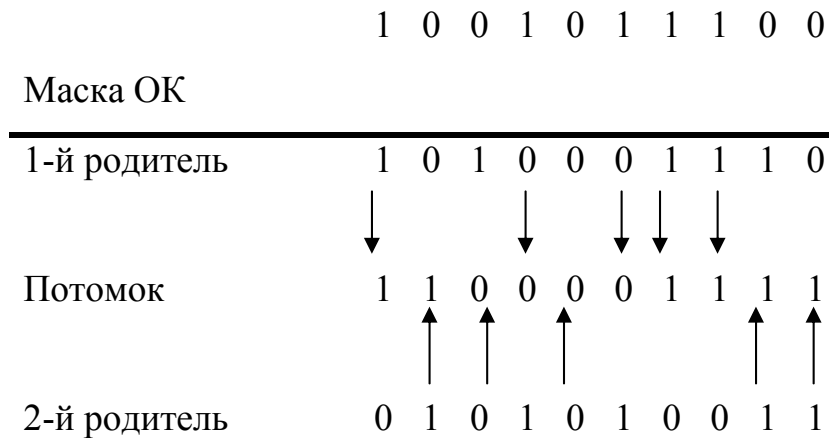


Рис.4.13. Однородный кроссинговер

Таким образом, потомок содержит смесь генов из каждого родителя

4.3.1.4. Ограниченный кроссинговер

В этом виде скрещивания точки кроссинговера могут выбираться только там, где значения генов у родителей различны.

4.3.2. Рекомбинация действительных значений

При этом хромосома представляется действительными (вещественными) числами.

4.3.2.1. Дискретная рекомбинация

Этот вид скрещивания определен над векторами, компонентами которых являются вещественные числа. Рассмотрим на примере двух особей, соответствующие вектора которых имеют три целых значения. Для каждой переменной (позиции) значение переменной выбирается из

первого или второго родителя с равной вероятностью. Пример выполнения этого оператора показан на рис.4.14. Здесь образцы (маски) показывают принадлежность данной компоненты потомка определенному родителю. Отметим, что этот тип рекомбинации применим к значениям любого типа.

4.3.1.2. Промежуточная рекомбинация

Этот метод применим только для особей, представленных только вещественными значениями. Здесь значения потомков строятся в окрестности или между значениями родителей.

1-й родитель	12	25	5
2-й родитель	123	4	34
1-й образец (маска)	2	2	1
2-й образец (маска)	1	2	1
1-й потомок	123	4	5
2-й потомок	12	4	5

Рис.4.14. Дискретная рекомбинация

В случае промежуточной рекомбинации потомок O_1 формируется следующим образом:

$$O_1 = P_1 + \alpha_1 \cdot (P_2 - P_1) , \quad O_2 = P_1 + \alpha_2 \cdot (P_2 - P_1) \quad , \quad (4.5)$$

где P_1, P_2 – вещественные значения, представляющие первого и второго родителя;

O_i – вещественное значение, представляющее потомка;

α_i - масштабирующий множитель, который выбирается случайно из отрезка $[-d, 1+d]$.

При обычной промежуточной рекомбинации $d = 0$ и $\alpha_i \in [0,1]$. Для обобщенной промежуточной рекомбинации $d > 0$, обычно принимают $d = 0,25$. Значение каждой переменной, в том числе и для векторов формируется по приведенному выражению. Отметим, что здесь используются чисто арифметические операции (сложение и умножение). Заметим, что при $d=0$ потомки принадлежат отрезку $[P_1, P_2]$. Эти операторы совершенно не похожи на классический кроссинговер. Фактически, этот оператор заимствован из другого направления эволюционных вычислений «эволюционные стратегии». На рис.4.15 показан пример выполнения этого оператора для тех же данных, которые использовались в предыдущем примере.

1-й родитель	12	25	5
2-й родитель	123	4	34
Случайно выбраны следующие значения коэффициента α			
1-й образец α_1	0,5	1,1	0,1
2-й образец α_2	0,1	0,8	0,5
1-й потомок	67,5	1,9	7,9
2-й потомок	23,1	8,2	19,5

Рис.4.15. Промежуточная рекомбинация

Для примера подробно рассмотрим выполнение оператора для 1-ой компоненты

$$O_1 = P_1 + \alpha_1 \cdot (P_2 - P_1) = 12 + 0.5 \cdot (123 - 12) = 67.5. \quad (4.6)$$

4.3.2.3. Линейная рекомбинация

Этот вид оператора аналогичен предыдущему, за исключением того, что значение масштабирующего множителя α одинаково для всех переменных (компонент) векторов. Пример выполнения этого оператора представлен на рис.4.16.

1-й родитель	12	25	5
2-й родитель	123	4	34
Случайно отобраны следующие значения коэффициента α			
1-й образец	$\alpha = 0,5$		
2-й образец	$\alpha = 0,1$		
1-й потомок	67,5	14,5	19,5
2-й потомок	23,1	22,9	7,9

Рис.4.16. Линейная рекомбинация

4.4. Оператор мутации

После выполнения операторов рекомбинации (скрещивания) полученные потомки с вероятностью P_m подвергаются мутации, которая может быть выполнена различными способами.

4.4.2. Двоичная мутация

4.4.2.3. Классическая мутация

Этот вид оператора мы уже рассматривали в простом ГА. Здесь для каждой особи случайно выбирается позиция и с малой вероятностью (от $P_m=0,01$ до $P_m=0,001$) выполняется инвертирование значения переменной в выбранной позиции. Пример выполнения этого оператора представлен на рис.4.18.

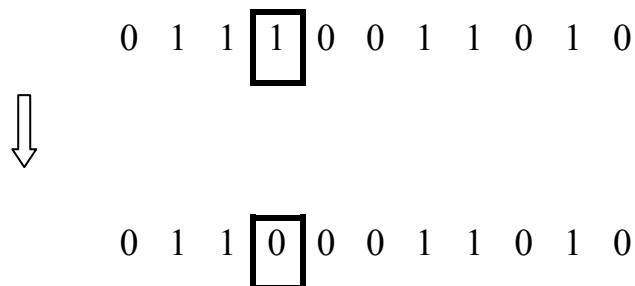


Рис.4.17. Классическая мутация

4.4.1.2 Оператор инверсии

Иногда используется следующий оператор инверсии, фактически являющийся разновидностью мутации. При этом случайным образом выбираются 2 позиции в особи и далее производится обмен значениями генов между ними. Пример выполнения этого оператора представлен на рис.4.19.

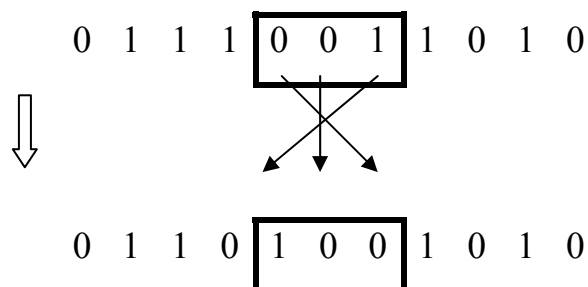


Рис.4.18.Инверсия

4.4.2 Мутация над вещественными числами

Мутация над вещественными потомками выполняется путем сложения особи с небольшим случайным значением, которое называется шагом мутации. Выбор размера шага мутации зависит от рассматриваемой проблемы, и шаг в общем случае может изменяться в процессе решения задачи. Маленький шаг дает большую точность, но ведет к большим временным затратам. Мутация с постоянным шагом и постоянной вероятностью называется однородной.

Оператор мутации над вещественным числом выполняется следующим образом

$$V_m = v \pm r \cdot \Delta, \quad (4.7)$$

где V , V_m – значения вещественной переменной до и после мутации;
 $r=0.5$ (диапазон изменения переменной).

Часто для повышения эффективности поиска вероятность мутации и шаг изменяются в процессе решения задачи. Рассмотрим далее способы изменения вероятности мутации. Мутация с равной вероятностью может привести как к увеличению, так и к уменьшению значения целевой функции. На этапе сходимости ГА к оптимуму целесообразно уменьшать вероятность случайной мутации. Обычно на начальном этапе $P_m = 0.05 \dots 0.1$, а на конечном этапе вероятность мутации уменьшают. Для реализации этой процедуры иногда используют метод моделирования отжига (simulation annealing), который дает следующий закон изменения вероятности мутации:

$$P_m = P_m^0 \cdot e^{-\frac{1}{t}}, \quad (4.8)$$

где t – номер поколения.

Здесь изменяется шаг мутации. Вначале шаг мутации имеет достаточно большое значение, которое далее постепенно уменьшается. Рассмотрим этот тип оператора для векторного случая

$$S_v^t = \langle V_1, V_2, \dots, V_k, \dots, V_m \rangle \\ V_k \in [l_k, U_k]$$

В результате выполнения мутации получаем следующий вектор (особь популяции)

$$S_v^{t+1} = \langle V_1, V_2, \dots, V'_k, \dots, V_m \rangle, \\ k \in [1, \dots, n] \quad (4.9)$$

где компонента вектора вычисляется следующим образом

$$V'_k = \begin{cases} V_k + \Delta(t, U_k - V_k) & \text{при случ. число} = 0 \\ V_k - \Delta(t, V_k - l_k) & \text{при случ. число} = 1 \end{cases}, \quad (4.10)$$

где $\Delta(t, y) \in [0, y]$ и $\Delta(t, y)$ определяет шаг мутации, который с увеличением номера поколения t уменьшается.

$$\Delta(t, y) \rightarrow 0 \\ t \rightarrow \infty$$

Один из вариантов реализации функции, определяющей шаг $\Delta(t, y)$

$$\Delta(t, y) = y \cdot \left(1 - r^{\left(1 - \frac{t}{T} \right)^b} \right), \quad (4.11)$$

где $r \in [1, \dots, n]$ (r -случайное число) и T – максимальное число поколений; $b = 2$ – параметр, определяющий степень неоднородности.

На рис.4.19 показан для наглядности график изменения шага мутации, который в пределе стремится к 0.

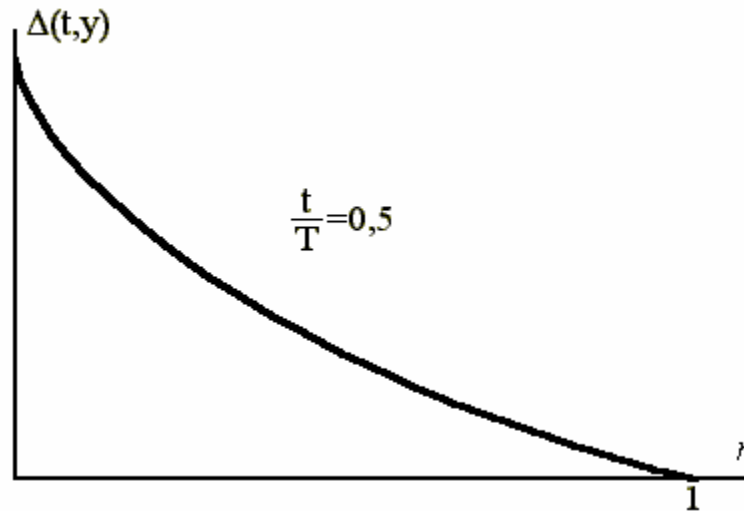


Рис.4.19. Уменьшение шага мутации

4.5. Сокращение промежуточной популяции

Необходимой компонентой является устранение плохих решений, полученных в процессе формирования новой популяции.

4.5.1. Глобальная редукция

Промежуточную популяцию (репродукционную группу) составляют все особи t -го поколения и новые особи, полученные в результате скрещивания и мутации. Численность этой популяции можно определить следующим образом

$$R^{t+1} = r^t + r_{cr}^t + r_m^t, \quad (4.12)$$

где: r^t - число особей предыдущей популяции;

r_{cr}^t - численность особей, полученных путем скрещивания;

r_m^t - число «мутантов».

Обычно в стационарных ГА мощность популяции поддерживается постоянной $N = |P(t)|$. Поскольку $R^{t+1} > N$, то необходимо устранить неудачные решения. Для этого существуют различные методы редукции.

4.5.1.1. Чистая замена

В простейшем случае с помощью скрещивания и мутации генерируются столько потомков, сколько было родителей. Далее родители устраняются, а потомки формируют следующее поколение $P(t+1)$. При этом каждая особь живет лишь одно поколение. Такая схема часто используется в простом ГА. Однако при этом очевидно возможно, что некоторые очень хорошие решения могут быть заменены худшими, и лучшее решение будет потеряно.

4.5.1.2. Элитарная схема

В ней потомков генерируется меньше, чем было родителей. Далее вновь построенные потомки заменяют худших родителей согласно значениям фитнес-функции. Для этой схемы возможна преждевременная сходимость к локальным экстремумам.

4.5.1.3. Равномерная случайная замена

При таком подходе потомков также генерируется меньше, чем было родителей, далее случайно удаляется необходимое число родителей, которые заменяются новыми потомками.

4.5.1.4. Пропорциональная редукция

Здесь потомков генерируется больше, чем необходимо для замены. Потом заменяют заданное число родителей только лучшими потомками согласно значениям фитнес-функции.

4.5.1.5. Селекционная схема

Здесь родители и потомки выступают на равных правах - они помещаются в одну репродукционную группу. В ней все особи этой группы ранжируются и в следующее поколение включаются только лучшие N особей. Иногда применяется следующая модификация этого метода. Сначала вычисляется среднее значение ЦФ группы и в следующее поколение включаются те особи, у которых значение ЦФ больше среднего значения группы.

В общем случае к репродукционной группе R^{t+1} может быть применен любой метод отбора родителей, которые были рассмотрены в разделе 3.2.

4.5.2. Локальная замена

При локальном отборе особи выбираются из ограниченного окружения множества соседей. Замена особей потомками выполняется по такой же схеме. Таким образом сохраняется локальность информации. При этом используются те же структуры и отношения соседства, что и в разделе 2.4.

Родитель особи определяются первым родителем из локального окружения (множества соседей). Для выбора удаляемых родителей и отбора потомков для замены применяются следующие схемы:

- Ввести в новое поколение каждого потомка и заменить случайным образом особи из их окружения;
- Ввести каждого потомка и заменить худшей особью соответственного окружения;
- Ввести потомка лучше, чем худшая особь в окружении и заменить худшие особи.

4.6. Асинхронные генетические алгоритмы

До сих пор мы рассматривали стационарные синхронные эволюционные алгоритмы, где мощность популяции поддерживается постоянной и переход к следующему поколению «синхронизирован». Здесь обрабатываются все потомки предыдущего поколения, а затем следует переход на следующую итерацию (поколение).

Иногда на практике применяется и асинхронный эволюционный алгоритм, где нет явного разбиения эволюции на поколения, а процесс развития реализуется «по событию» в виде непрерывного потока событий отбора, кроссинговера, мутации и т.п. Здесь вновь порожденные потомки сразу замещают (в некотором смысле худшие) особи (не дожидаясь остальных родительских пар своего поколения). Ниже представлен один из возможных вариантов асинхронного алгоритма реализации ГП с применением турнирного отбора родителей.

- 0) установка параметров эволюции;
- 1) инициализация начальной популяции;
- 2) случайный выбор подмножества особей популяции – участников турнира;
- 3) оценка фитнес-значений каждого участника турнира;
- 4) отбор победителей турнира;

- 5) выполнение генетических операторов кроссинговера и мутации для победителей;
- 6) замена проигравших особей потомками победителей;
- 7) если критерий окончания не выполнен, переход на 2);
- 8) выбор лучшего решения в конечной популяции.

Следует отметить, что такой метод, как правило, проще в реализации (особенно аппаратной) и имеет некоторые преимущества.

4.7. Генетические алгоритмы с изменяемой мощностью популяций

Мощность популяции N является важнейшим параметром ГА, который критичен во многих приложениях. Если N мало, то ГА работает быстро, но при этом увеличивается опасность преждевременной сходимости к локальному экстремуму. Большая мощность популяции увеличивает генофонд, но процесс поиска замедляется.

На разных этапах работы ГА оптимальное значение N может быть различным. На начальном этапе N должно быть большим, а на заключительном N можно уменьшить.

При одном из подходов в ГА с изменяемым размером популяции каждой особи после ее рождения на текущем этапе оценки ЦФ присваивается «время жизни» (life time) L_f – параметр, зависящий от ЦФ особи. Таким образом, каждая особь живет определенное число поколений и умирает по окончании срока жизни. Очевидно, значение этого параметра влияет на размер популяции. В этом случае ГА можно реализовать, например, следующим образом [17].

Нестационарный_ГА

```

{
    t=0;
    Инициализация;
    Оценка ЦФ p(t);
    While (условие окончания не выполнено)
    {
        t=t+1;
        Увеличение возраста каждой особи на 1;
        Рекомбинация p(t):
        Мутация p(t):
        Оценка ЦФ p(t):
        Определение срока жизни особей;
        Удаление из p(t) всех особей с возрастом больше срока
        жизни;
    }
}

```

Здесь в текущем поколении t алгоритм обрабатывает популяцию $P(t)$. В процессе рекомбинации и мутации генерируется промежуточная популяция, состоящая из потомков, и ее размер пропорционален числу исходной $r_{cr}^t + r_m^t = r_a^t = r \cdot P$.

Тогда $|P(t+1)| = |P(t)| + r_a^t - D(t)$ – число особей в новой популяции.

Срок жизни для каждой особи определяется после оценки ЦФ (по формулам, приведенным ниже) и является окончательным, то есть постоянным в процессе эволюции. В таком случае особь живет определенное число поколений, а затем умирает. Срок жизни определяет число поколений, в течении которых особь держится в популяции.

Очевидно, что в этом случае, чем больше срок жизни, тем больше потомков может дать особь, так как на каждом этапе родители для рекомбинации выбираются случайно с равной вероятностью. При этом подходе важнейшую роль играет метод определения срока жизни особи.

Очевидно, постоянное значение $L_f \geq 2$ для каждой особи ведет к экспоненциальному росту размеров популяции. Поэтому для каждой особи срок жизни вычисляется индивидуально в зависимости от значения ее ЦФ.

Наиболее часто используются следующие три способа определения срока жизни.

Для их определения введем следующие обозначения:

\overline{f} - среднее значение ЦФ по популяции;

f_{\max} - максимальное значение ЦФ по популяции;

f_{\min} - минимальное значение ЦФ по популяции;

$f_{a \max}$ - абсолютное максимальное значение;

$f_{a \min}$ - абсолютное минимальное значение;

L_{\max} - максимальный срок жизни;

L_{\min} - минимальный срок жизни.

1. При пропорциональном методе определения срока жизни

L_f определяется по следующей формуле:

$$L_f = \min \left\{ L_{\min} + \eta \cdot \frac{f[i]}{\overline{f}}, L_{\max} \right\}, \quad (4.13)$$

где $\eta = \frac{1}{2} \cdot (L_{\max} - L_{\min})$ (используется для всех формул).

2. При линейном методе определения срока жизни

$$L_f = L_{\min} + 2 \cdot \eta \cdot \frac{f[i] - f_{a \min}}{f_{a \max} - f_{a \min}} \quad (4.14)$$

3. В билинейном методе определения срока жизни

$$L_f = \begin{cases} L_{\min} + \eta \cdot \frac{f[i] - f_{\min}}{f - f_{\min}}, & \text{если } \bar{f} \geq f[i] \\ \frac{1}{2} \cdot (L_{\min} + L_{\max}) + \eta \cdot \frac{f[i] - \bar{f}}{f_{\max} - \bar{f}}, & \text{если } \bar{f} < f[i] \end{cases} \quad (4.15)$$

Очевидно, что первый метод соответствует пропорциональному отбору отбора родителей - «рулетке». К минимальному сроку жизни добавляется премиальный срок, который пропорционален значению ЦФ для данной особи. Однако, эта стратегия имеет серьезный недостаток – она не учитывает информацию, о некоторых объективных характеристиках особи, такой, например, как отношение $\frac{f[i]}{f_{\max}}$ (или $\frac{f[i]}{f_{\min}}$) целевой функции по популяции.

Эту проблему решает вторая (линейная) стратегия, где срок жизни определяется исходя из значения ЦФ данной особи относительно максимального значения в популяции f_{\max} . Но этот метод тоже имеет свои недостатки - если в популяции много особей имеют значение ЦФ стремящееся к максимальному ($f[i] \rightarrow f_{\max}$) значению, то такой подход приведет к чрезмерному увеличению размера популяции.

В третьей стратегии (билинейной) предпринята попытка найти компромисс между первыми двумя методами. В ней учитываются разница между сроками жизни, близких к лучшей особи, используя информацию о

среднем значении популяции. Однако, в тоже время принимается во внимание минимальное и максимальное значение по популяции.

4.8. Адаптивные ГА

В адаптивных генетических алгоритмах изменяются параметры ГА, прежде всего, вероятности кроссинговера и мутации P_c и P_m .

Для оптимизации, особенно мультимодальных функций, наиболее существенными являются две характеристики ГА:

- способность сходиться к оптимуму (локальному или глобальному) после нахождения области, содержащей этот оптимум;
- способность находить новые области в пространстве решений в поисках глобального оптимума.

Баланс между этими характеристиками ГА определяется значениями вероятности P_c и P_m и типом используемых генетических операторов (прежде всего кроссинговера). Увеличение значений P_c и P_m ведет к расширению пространства поиска.

Обычно используют следующие значения вероятностей $P_c \in [0.5; 1]$ и $P_m \in [0.001; 0.01]$. Далее мы рассмотрим другой подход [24], который использует различные значения P_c и P_m в зависимости от значения ЦФ текущих особей. Для мультимодальных функций существует серьезная проблема преждевременной сходимости к локальным экстремумам.

Чтобы изменять P_c и P_m адаптивно, с целью предотвращения преждевременной сходимости к локальному экстремуму, надо научиться идентифицировать ситуации, когда ГА сходится к оптимуму. Рассмотрим этот подход на примере поиска максимума для мультимодальной функции (которая имеет несколько экстремумов), которая показана на рис. 4.20.

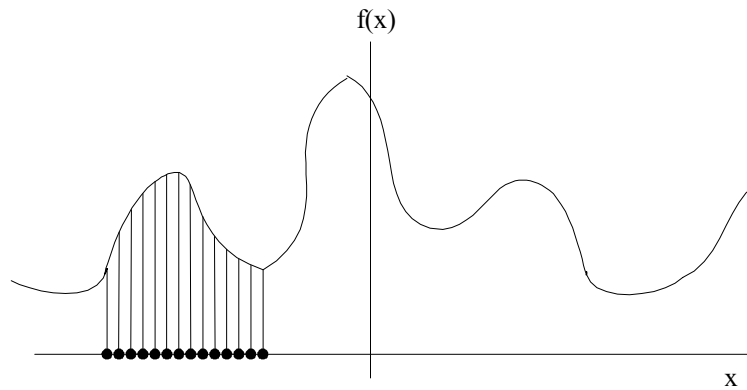


Рис.4.20. Мультимодальная функция

Один из возможных способов обнаружения сходимости – наблюдение разности среднего и максимального значения целевой функции по популяции $(f_{\max} - \bar{f})$. Обычно эта разность меньше для популяции, которая сходится к оптимуму, чем для популяции, «разбросанной» по пространству поиска решений. Будем использовать разность $(f_{\max} - \bar{f})$ в качестве основного признака сходимости к оптимуму, причем не обязательно глобальному.

Поскольку вероятности P_c и P_m должны увеличиваться при преждевременной сходимости к локальному оптимуму (чтобы «выпрыгнуть из ловушки» локального экстремума), то значение $(f_{\max} - \bar{f})$ должны изменяться обратно пропорционально разности $(f_{\max} - \bar{f})$:

$$P_c = \frac{k_1}{f_{\max} - \bar{f}} \quad P_m = \frac{k_2}{f_{\max} - \bar{f}}. \quad (4.16)$$

Здесь P_c и P_m не зависят от значений ЦФ для конкретной особи, а определяются для всей популяции, но для разных популяций различных поколений они уже будут разными.

Отметим, что как для хороших решений с высокими значениями целевой функции, так и для плохих, используются одни и те же величины P_c и P_m . Это не рационально, так как когда популяция сходится к глобальному или локальному оптимуму, то P_c и P_m увеличиваются, что ведет к разрушению близких к оптимуму решений. В результате популяция может не сойтись к глобальному оптимуму. Таким образом, мы избежали ловушки локального экстремума ценой снижения характеристик ГА по поиску глобальных экстремумов.

Чтобы решить эту проблему, нужно сохранить хорошие решения текущей популяции. Это можно сделать, полагая низкие значения вероятности P_c и P_m для особей с высоким значением ЦФ и высокие значения P_c и P_m для плохих особей с низкими значениями ЦФ. Тогда лучшие решения будут способствовать сходимости, а худшие – предотвращать ГА от «ловушек» локальных экстремумов. Таким образом, значение P_c и P_m должны зависеть не только от разности $(f_{\max} - \bar{f})$, но еще и от значений ЦФ конкретных особей. Чем ближе значение функции к максимальному, тем меньше должно быть значение P_c и P_m :

$$\begin{aligned} P_c &= k_1 \cdot \frac{f_{\max} - f'}{f_{\max} - \bar{f}}, & k_1 &\leq 1 \\ P_m &= k_2 \cdot \frac{f_{\max} - f}{f_{\max} - \bar{f}}, & k_2 &\leq 1 \end{aligned}, \quad (4.17)$$

где f' – лучшее значение целевой функции у двух родителей.

Положим $P_c = P_m = 0$ для решений, имеющих максимальное значение ЦФ и

$$\begin{aligned} P_c &= k_1, & f' &= \bar{f} \\ P_m &= k_2, & f' &= \bar{f} \end{aligned} \quad (4.18)$$

Отметим, что для плохих решений $(f, f' < \bar{f})$ значения вероятностей P_c и P_m в соответствии с формулой (4.17) могут быть больше 1, что некорректно. Поэтому для плохих решений примем:

$$\begin{aligned} P_c &= k_3, & f' &\leq \bar{f} \\ P_m &= k_4, & f' &\leq \bar{f} \end{aligned}$$

В итоге получим:

$$\begin{aligned} P_c &= \begin{cases} k_1 \cdot \frac{f_{\max} - f'}{f_{\max} - \bar{f}}, & f' > \bar{f} \\ k_3, & \text{иначе} \end{cases} \\ P_m &= \begin{cases} k_2 \cdot \frac{f_{\max} - f}{f_{\max} - \bar{f}}, & f > \bar{f} \\ k_4, & \text{иначе} \end{cases} \end{aligned} \quad (4.19)$$

$$k_1 = k_3 = 1$$

$$k_2 = k_4 = 0.5$$

Лучшие решения при этом сохраняются и переходят в следующее поколение. Этот факт может привести к чрезмерному росту популяции, что чревато преждевременной сходимостью. Поэтому иногда дополнительно вводится одна (default) мутация (с вероятностью $P_d = 0.005$) для всех особей популяции.

Существуют и более строгие адаптивные ГА, где вероятности P_c и P_m вычисляются аналитически, но они, как правило, сильно привязаны к конкретным задачам. Достаточно эффективным средством

адаптации является использование «нечетких контроллеров» в виде, например, системы продукций в нечеткой логике.

4.9. Ниши в генетических алгоритмах

Для мультимодальных функций, которые имеют много экстремумов, часто представляет интерес найти не одно, а несколько экстремальных значений. С помощью стандартного ГА это трудно сделать, поскольку в процессе эволюции, как правило, благодаря «генетическому дрейфу» особи концентрируются в окрестности одного экстремума. Поэтому для поиска экстремумов мультимодальных функций были разработаны соответствующие методы.

Простейший из них основан на многократном запуске ГА на различных подмножествах пространства поиска решений. Показано [24], что, если все экстремумы имеют примерно одинаковую небольшую вероятность поиска (быть найденными), то число независимых запусков ГА должно быть

$$p \sum_{i=1}^p \frac{1}{i} \approx p(\gamma + \log p), \quad (4.20)$$

где p – число экстремумов и $\gamma \approx 0,577$ – константа Эйлера. К сожалению, в большинстве реальных задач экстремумы не являются равновероятными и поэтому число запусков должно быть больше приведенной оценки. Возможна также параллельная реализация этого итеративного метода.

В [25] предложен наиболее известный метод для данной проблематики, который основан на разделении популяции на несколько подпопуляций. Основная идея состоит в том, что фитнес-функция модифицируется таким образом, что в том случае, когда особи концентрируются вокруг экстремума, значение фитнес-функции для них уменьшается пропорционально числу особей в этой области. При этом

модифицированное значение фитнес-функции особи I , $s_f(i)$, называемое разделенной фитнес-функцией, определяется следующим образом

$$s_f(i) = \frac{f(i)}{m(i)}, \quad (4.21)$$

где $f(i)$ – значение исходной фитнес-функции и $m(i)$ называется счетчиком ниши. Для особи i величина $m(i)$ вычисляется путем суммирования значений разделяющей функции sh для особей всей популяции

$$m(i) = \sum_{j=1}^N sh(d_{ij}), \quad (4.22)$$

где d_{ij} – евклидово расстояние между двумя особями i и j . Разделяющая функция $sh(d_{ij})$ должна обладать следующими свойствами:

$$\begin{aligned} 0 \leq sh(d_{ij}) \leq 1 \text{ для каждого } d_{ij}, \\ sh(0) = 1, \\ \lim_{d_{ij} \rightarrow \infty} sh(d_{ij}) = 0 \end{aligned} \quad (4.23)$$

Одной из применяемых на практике функций, для которой эти условия выполняются, является следующая

$$sh(d_{ij}) = \begin{cases} 1 - \left(\frac{d_{ij}}{\sigma_s} \right)^\alpha & \text{если } d_{ij} < \sigma_s \\ 0 & \text{иначе} \end{cases} \quad (4.24)$$

Здесь α и σ^s являются константами. Наибольшие трудности в этом методе вызывает выбор значения σ^s , который требует априорного знания числа экстремумов функции, что, как правило, заранее неизвестно.

Например, в программе FlexTool [23] $\sigma^s = 0,5 * q^{-\frac{1}{p}}$, где q полагается равным примерному числу экстремумов. Значение ω часто полагают равным 1, что означает одинаковую степень соучастия соседних особей. Таким образом, функция $sh(d_{ij})$ определяет уровень близости и степень соучастия для каждой особи в популяции. Если особь находится в своей нише в одиночестве, то $s_f(i) = f(i)$. В противном случае значение модифицированной фитнес-функции уменьшается пропорционально количеству и степени близости соседствующих хромосом. При этом увеличение количества похожих друг на друга хромосом в одной нише ограничено, поскольку такое увеличение ведет к уменьшению значения фитнес-функции таких особей. Имеются различные модификации этого метода. Например, расстояние d_{ij} между особями иногда определяются не на уровне фенотипа (евклидово расстояние), а на уровне генотипа, где используется расстояние Хэмминга между двоичными кодами хромосом.

В работе [26] приведено сравнение параллельных и последовательных методов обработки ниш. Параллельные методы формируют и сохраняют ниши одновременно с популяцией. Последовательные методы обрабатывают различные ниши в разные моменты времени. Как правило, по эффективности параллельные методы превосходят последовательные.

4.10. Многокритериальная оптимизация

В большинстве реальных практических задач, как правило, необходимо выполнить оптимизацию по нескольким критериям. Многокритериальная оптимизация основана на поиске решения, которое одновременно оптимизирует не одну, а несколько функций. Это требует применения специальных методов, которые существенно отличаются от

стандартной техники, ориентированной на оптимизацию одной функции. При многокритериальной оптимизации выполняется поиск не одной особи, а множество хромосом, оптимальных в смысле Парето [17, 23]. Обычно пользователь имеет возможность выбирать оптимальное решение из этого множества.

Для этих целей удобно классифицировать потенциальные решения многокритериальной проблемы на доминируемые и недоминируемые решения. Решение x называется доминируемым, если существует решение y , не хуже чем x по всем критериям, то есть для всех оптимизируемых функций f_i ($i=1, \dots, k$):

$$\begin{aligned} f_i(x) &\leq f_i(y) \text{ для всех } 1 \leq i \leq k \text{ при максимизации функции } f_i \text{ и} \\ f_i(x) &\geq f_i(y) \text{ для всех } 1 \leq i \leq k \text{ при минимизации функции } f_i. \end{aligned}$$

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

Разработано несколько классических методов многокритериальной оптимизации [17, 23]. Наиболее известным является метод взвешенной функции, где новая «общая» целевая функция строится из заданных в виде взвешенной суммы

$$F(x) = \sum_{i=1}^k w_i f_i(x), \text{ где веса } w_i \in [0,1] \text{ и } \sum_{i=1}^k w_i = 1. \quad (4.25)$$

При этом различные веса w_i дают разные решения в смысле Парето.

Кроме изложенного, часто применяется также метод функции расстояния. Этот метод основан на сравнении значений $f_i(x)$ с заданными значениями y_i согласно следующему выражению

$$f(x) = \left(\sum_{i=1}^k |f_i(x) - y_i|^r \right)^{\frac{1}{r}}. \quad (4.26)$$

Как правило, используется Евклидова метрика, для которой $r=2$.

В некоторых случаях многокритериальная оптимизация основана на разделении популяции на подпопуляции одинакового размера, каждая из которых «отвечает» за свою оптимизируемую функцию. Отбор производится автономно для каждой подпопуляции (функции), но оператор кроссинговера выполняется без учета границ подпопуляций.

4.11. Генетические микроалгоритмы

Данная модификация предназначена для решения задач, которые не требуют популяций с большим числом особей и длинных хромосом. Такой подход оправдан в том случае, когда решение (не обязательно глобальный оптимум) необходимо найти как можно быстрее. В этом случае необходимо уменьшить трудоемкость, связанную с большим количеством итераций, ценой возможно ухудшения качества решения. Такой подход, например, реализован в программе FlexTool [23] следующим генетическим микроалгоритмом.

1. Формирование популяции из 5 особей. Можно либо случайно выбирать все 5 хромосом, либо сохранять одну «хорошую» особь, полученную на предыдущих итерациях, и случайно генерировать остальные 4 особи.
2. Расчет значений фитнес-функции особей в популяции и выбор лучшей особи. Присвоить ей номер 5 и перенести в следующее поколение (согласно элитарной стратегии).
3. Выбор для репродукции остальных четырех хромосом на основе турнирного метода селекции. При этом хромосомы группируются случайным образом и соседствующие пары соперничают за оставшиеся 4 места. Необходимо следить, чтобы родительская пара не формировалась из двух копий одной и той же хромосомы.
4. Выполнение кроссинговера с вероятностью $p_m=1$ (вероятность мутации положить $p_m=0$).

5. Проверка сходимости алгоритма (на основе сравнения фенотипов или генотипов). В случае сходимости возврат на шаг 1.

6. Переход на шаг 2.

Отметим, что в генетическом микроалгоритме используется небольшой фиксированный размер популяции и элитарная стратегия отбора родителей, которая предотвращает потерю хороших особей. Мутация здесь не применяется поскольку разнообразие генетического материала обеспечивается формированием новой популяции при каждом рестарте алгоритма(переходе на шаг 1 при обнаружении сходимости). Процедуры «старта» и «рестарта» предназначены для предотвращения преждевременной сходимости.

4.12 Контрольные вопросы к разделу 4

1. Какие методы применяются для генерации начальной популяции?
2. Какая информация используется при отборе родителей?
3. Какие недостатки имеет «метод рулетки»?
4. Чем отличается ранжирование от пропорционального отбора?
5. Что такое локальный отбор?
6. Опишите метод турнирного отбора.
7. Как используется метод Больцмана при отборе особей?
8. Опишите методы отбора пар для скрещивания.
9. Что такое неявные методы отбора?
10. Опишите двоичную рекомбинацию.
11. Чем отличается многоточечный кроссинговер от классического?
12. Что такое однородный кроссинговер?
13. Чем отличается рекомбинация действительных чисел от классического кроссинговера?

14. Что такое дискретная рекомбинация?
15. Опишите промежуточную рекомбинацию.
16. Чем отличается линейная рекомбинация от промежуточной?
17. Что такое инверсия?
18. Как выполняется мутация над вещественными числами?
19. Чем отличается неоднородная мутация от обычной?
20. Какие существуют методы сокращения популяции?
21. Опишите нестационарный ГА.
22. Чем заменяется отбор родителей в нестационарном ГА?
23. Какие методы определения сроков жизни вы знаете?
24. В чем заключается адаптация в ГА?
25. Как изменяются вероятности кроссинговера и мутации при адаптации?
26. Что такое ниши в ГА?
27. Чем отличается многокритериальная оптимизация от обычной?
28. Как многокритериальная оптимизация может быть сведена к однокритериальной?
29. Что такое оптимизация «по Парето»?
30. В каких случаях целесообразно применять генетический микроалгоритм?

5. ПАРАЛЛЕЛЬНЫЕ ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ

5.1. Структуризация ГА

Для нетривиальных задач выполнение одного репродуктивного цикла – поколения в ГА требует значительных вычислительных ресурсов. При решении многих задач используется не двоичное представление особи – решения проблемы, а более сложные структуры – массивы (матрицы) действительных чисел, связные списки [22], деревья, графы [5,27] и т.д. Поэтому вычисление значения фитнес-функции для каждой особи, потенциального решения проблемы, часто является самой трудоемкой операцией в ГА. Для повышения эффективности разрабатываются новые методы кодирования особей, генетические операторы кроссинговера и мутации, гибридные алгоритмы, параллельные алгоритмы и т.п.[28-30].

Присущие ГА "внутренний" параллелизм и заложенная в них возможность распределенных вычислений способствовали развитию параллельных ГА (ПГА). Первые работы в этом направлении появились в 60-х годах, но только в 80-е годы, когда были разработаны доступные средства параллельной реализации, исследования ПГА приняли систематический массовый характер и практическую направленность. В этом направлении разработано множество моделей и реализаций, некоторые из которых представлены ниже.

Прежде всего, необходимо отметить, что в основе ПГА лежит структуризация популяции (множества потенциальных решений) – его разбиения на несколько подмножеств (подпопуляций). Это разбиение можно сделать различными способами, которые и определяют различные виды ПГА. Согласно современной классификации различают глобальные ПГА, распределенные ГА (РГА), клеточные ГА (КГА) и коэволюционные ГА (КЭГА). В простом ГА, графически представленном на рис. 5.1а),

5.2. Параллельный генетический алгоритм на основе модели «рабочий хозяин»

В данном разделе для распараллеливания ГА используется модель «рабочий - хозяин» (иногда она называется «клиент-сервер»), поскольку она требует наименьших изменений в существующей версии программного обеспечения, реализующего последовательный ГА и дает неплохие результаты. Ниже представлен укрупненный алгоритм параллельного ГА на основе модели «рабочий - хозяин».

Параллельный ГА «Рабочий - хозяин»

```
{
  Генерация популяции Р хромосом случайным образом;
  выполнение параллельно для всех особей
  {
    Оценка значения фитнес-функции для каждой особи;
  }
  While(критерий останова не выполнен)
  {
    отбор лучших особей;
    выполнение генетических операторов кроссиговера и мутации;
    формирование промежуточной популяции;
    выполнение параллельно для всех особей
    {
      Оценка значения фитнес-функции для каждой особи;
    }
    внесение лучших новых особей;
    удаление худших старых особей;
    формирование новой популяции ;
  }
}
```

}
}

При этом затраты по вычислению значений фитнес-функций равномерно распределяются по всем процессорам, для которых используется одна и та же фитнес-функция. Поэтому для n особей и P (одинаковых) процессоров мы каждому процессору относим n/P особей. Значения фитнес-функции вычисляются соответствующими (рабочими) процессорами и посылаются в один процессор (хозяин), который собирает всю информацию, обрабатывает и передает ее снова рабочим процессорам. Процессор «хозяин» имеет информацию о значениях фитнес-функции для всех особей и может генерировать следующее поколение на этой основе.

Итак, процессор – хозяин выполняет центральную часть (ядро) алгоритма, в то время как «черновая работа» - вычисление значений фитнес-функции для всех особей реализуется на процессорах – рабочих. Для баланса множество обрабатываемых особей популяции разбивается на примерно одинаковые подмножества.

В конце каждого из этапов помещаются точки синхронизации. Когда процессор-хозяин достигает эти точки, он переходит в режим ожидания, пока все рабочие процессоры не закончат свои задания, что гарантирует глобальную корректность алгоритма. При этом работа между процессором-хозяином и рабочими распределяется следующим образом.

Процессор-хозяин:

- выполняет все вход-выходные операции с пользователем и файловой системой, читает задание и записывает результаты;
- первоначально запускает «рабочие» процессы на доступных ресурсах;
- распределяет задания каждому рабочему процессору;

- организует управление процессом поиска решения и по мере

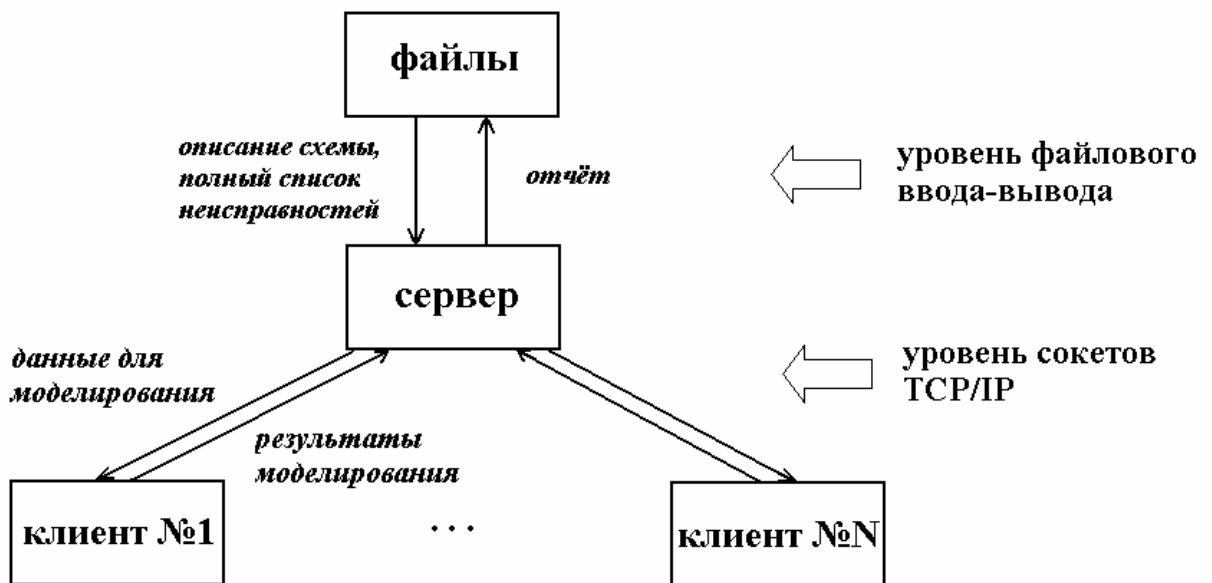


Рис.5.2. Поток данных в модели «рабочий – хозяин» («клиент-сервер»)

необходимости посылает соответствующие сообщения активации рабочих процессоров; по окончании задания рабочим процессором процессор-хозяин принимает полученные результаты и соответственно изменяет глобальные структуры данных (общий список заданий, значения фитнес-функции для особей и т.п.).

Каждый «рабочий» принимает задание от «хозяина» и определяет значение фитнес-функции для особей, полученный результат посылает хозяину и ожидает следующего задания. Поскольку размер популяции много больше числа процессоров, достигается хороший баланс в загрузке процессоров. Диаграмма потоков данных в данном алгоритме приведена на рис.5.2 (на примере распределенного логического моделирования).

При использовании модели «рабочий - хозяин» окончательные результаты (поиска решения данной задачи) близки к тем, что получены на однопроцессорной компьютерной системе с использованием аналогичного алгоритма. Качество решения при этом не теряется и в большинстве

случаев несколько улучшается, а время его поиска существенно сокращается. В целом данная модель позволяет быстро (с минимальными модификациями) выполнить параллелизацию ГА и дает, прежде всего, ускорение процесса поиска решения. Данную модель не сложно реализовать в локальной сети с использованием технологии сокетов.

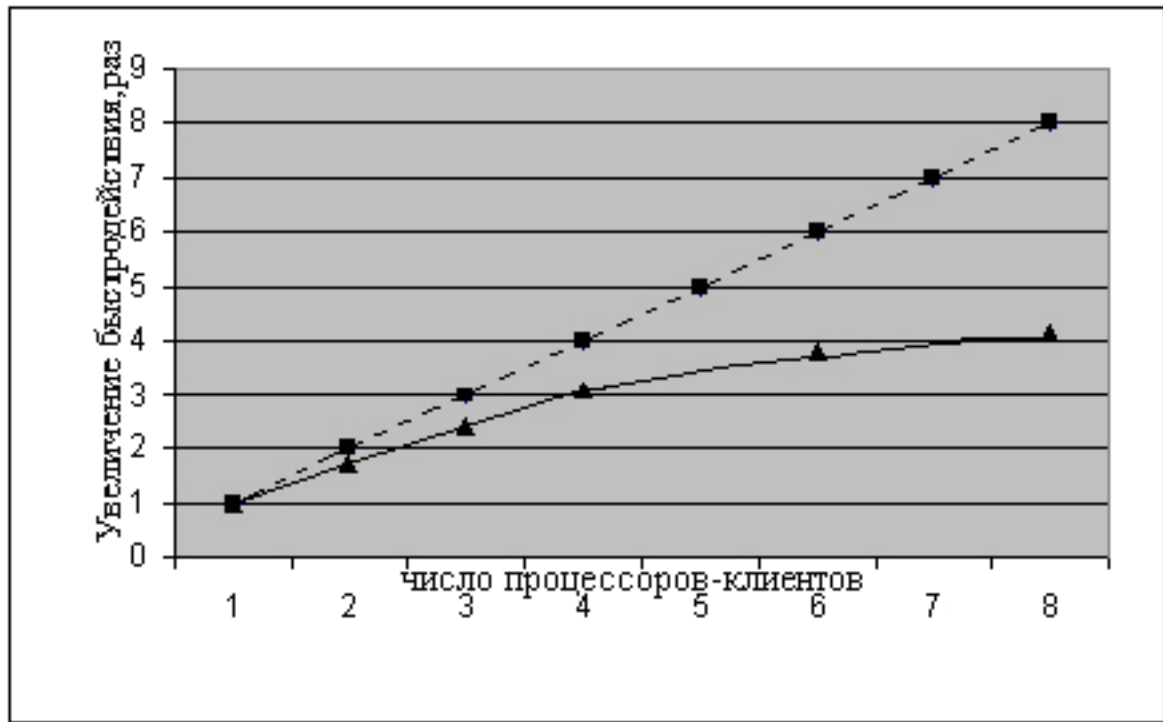


Рис.5.3. Рост быстродействия при увеличении числа процессоров-клиентов

Типичный график увеличения быстродействия (для задачи построения проверяющих тестов) представлен на рис.5.3.[31]. Здесь сверху для сравнения представлен «идеальный» линейный график роста ускорения в зависимости от увеличения числа процессоров. Реальное ускорение (особенно для большого числа процессоров) естественно несколько меньше.

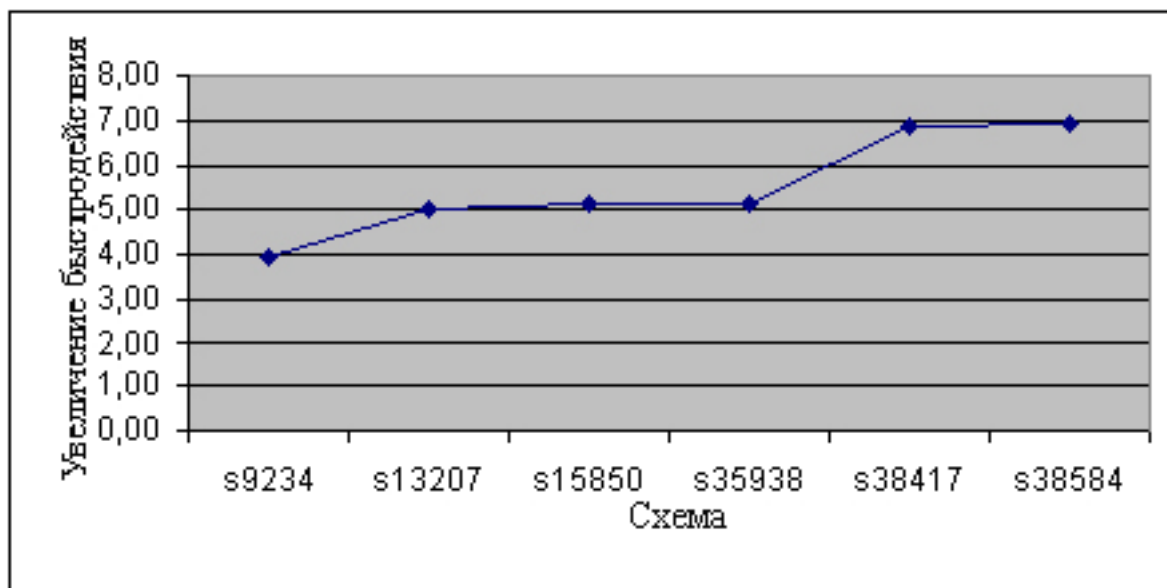


Рис.5.4. Увеличение быстродействия с ростом размерности задачи (сложности схемы)

Следует отметить, что задач большей размерности выигрыш в увеличении быстродействия обычно больше, так как «накладные расходы» на пересылки по сравнению с основными затратами на поиск решения составляют меньшую долю. На рис.5.4 для примера представлен график увеличения быстродействия с ростом размерности задачи (сложности обрабатываемой схемы) [31]. Эксперименты показывают, что при грамотной реализации этой модели даже в локальной сети можно выиграть порядок в увеличении быстродействия (при относительно небольших модификациях исходного программного обеспечения).

5.3. Параллельные генетические алгоритмы на основе «модели островов»

Распределенные ГА используют, в основном, так называемую "модель островов", где каждая подпопуляция развивается на своем "острове". Между островами производится (достаточно редко) обмен

лучшими особями. Эта модель может быть реализована в распределенной памяти компьютерной системы, имеющей MIMD-архитектуру согласно классификации Flynn. Преимущество РГА в том, что они работают быстрее даже на однопроцессорных компьютерных системах вследствие лучшей структуризации. Причина заключается в том, что число вычислений сокращается благодаря распределению поиска в различных областях пространства решений. Разработаны различные виды РГА, но практически все они являются вариациями базового алгоритма, который представлен следующим псевдокодом.

Распределенный ГА

```
{
  Генерация популяции Р хромосом случайным образом;
  Разбиение Р на подпопуляции  $P_1, P_2, \dots, P_N$ ;
  Определение структуры соседства для  $SP_i, i=1, \dots, N$ ;
  While(критерий останова не выполнен)
    выполнение для  $SP_i, i=1, \dots, N$  параллельно следующих шагов
      {
        В течение  $f_m$  поколений выполнение отбора и генетических
операторов;
        Посылка  $n_m$  хромосом в соседние подпопуляции;
        Прием хромосом от соседних подпопуляций;
      }
}
```

Основными факторами, которые влияют на миграцию в модели островов (и следовательно, на их эффективность) являются следующие.

1) **Топология**, определяющая отношение соседства между подпопуляциями. Здесь обмен особями происходит только между

соседними подпуляциями. Существует также несколько стандартных схем обмена особями между подпуляциями, которые представлены ниже.

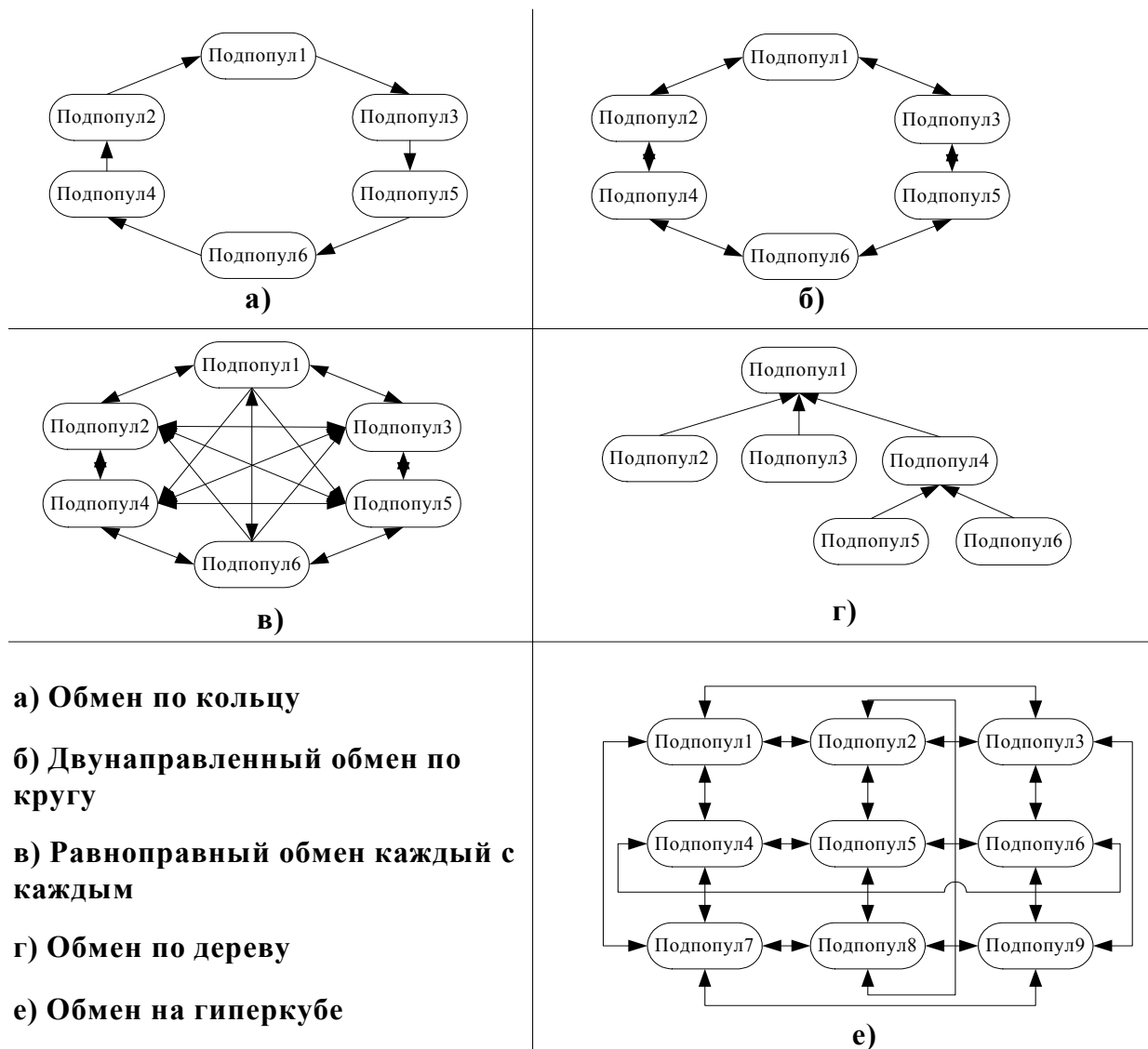


Рис.5.5. Типовые схемы обмена особями в модели островов

Для каждой подпуляции формируется “пул” - множество потенциальных эмигрантов из других подпуляций. Далее из этого “пула” по определенному закону выбираются эмигранты для данной подпуляции.

- 2) **Степень миграции**, которая определяет количество мигрирующих особей.
- 3) **Время изоляции**, определяющее число поколений между сеансами миграции.
- 4) **Стратегия отбора особей** в пул обмена. Здесь наиболее распространенными являются два метода. При первом подходе из подпопуляции особи выбираются случайным образом – при этом сохраняется разнообразие генетического материала. При втором методе из каждой подпопуляции выбираются лучшие в некотором смысле особи, что делает процесс более направленным. При этом также существуют различные методы отбора лучших хромосом (раздел 4.2).
- 5) **Стратегия замены** особей на мигрировавшие хромосомы из соседних подпопуляций. Здесь также существуют различные подходы: из подпопуляции удаляются худшие, случайные особи и т.п.
- 6) **Стратегия репликации** мигрирующих особей. При первом подходе мигрирующая особь остается также и в "родной" подпопуляции. Второй подход требует удаления мигрирующей особи из "родной" подпопуляции. Первая стратегия может привести к доминированию в различных подпопуляциях одних и тех же сильных особей. При второй стратегии особь может через некоторое время вернуться назад в исходную подпопуляцию, что ведет к лишним затратам вычислительных ресурсов.

5.4. Виды распределенных ГА

Распределенные ГА можно классифицировать по следующим признакам.

1) По методу миграции принято разделять:

а) изолированные РГА (isolated DGA), где нет миграции между подпопуляциями (иногда называются разделенные РГА – partitioned DGA);

б) синхронные РГА (synchronous DGA), в которых миграции между подпопуляциями синхронизированы – выполняются в одно и то же время;

в) асинхронные РГА (asynchronous DGA), где миграции могут происходить по событию в разные моменты времени в различных подпопуляциях, в зависимости от активности в каждой подпопуляции (асинхронное поведение характерно для естественной эволюции, которая развивается по-разному в зависимости от внешней среды).

2) По изменению схемы обмена особями различают:

а) статическая схема соединений между подпопуляциями, которая не изменяется в процессе эволюции;

б) динамическая схема соединений, в которой вид обмена особями между подпопуляциями может изменяться в процессе эволюции.

3) По однородности подпопуляций принято разделять на:

а) однородные РГА (homogeneous DGA), где каждая подпопуляция использует одни и те же способы кодирования хромосомы, генетические операторы, вид фитнес-функций и т.п.;

б) неоднородные РГА (heterogeneous DGA), в которых каждая подпопуляция может иметь различные параметры управления, метод кодирования хромосомы, генетические операторы и т.п.

Неоднородные РГА по мнению некоторых авторов являются хорошим инструментом для предотвращения преждевременной сходимости в локальных экстремумах и позволяют эффективно решать проблему расширения (исследования) и эксплуатации зоны поиска решений. В этой области разработаны некоторые интересные модификации РГА, которые представлены ниже.

- 1) *Адаптация конкурирующих подпопуляций.* Здесь для каждого возможного вида генетических операторов формируется подпопуляция. Общее число особей фиксировано (во всех подпопуляциях), в то время как мощность каждой подпопуляции может варьироваться. Каждая подпопуляция соревнуется с конкурентами таким образом, что она приобретает или теряет особи в зависимости от ее качества относительно других подпопуляций. Например, разные подпопуляции могут иметь различный шаг мутации (над вещественным представлением) или отличающиеся арифметические операторы рекомбинации. С другой стороны каждая подпопуляция может иметь различные схемы обмена, степень миграции и т.п. Через определенное число поколений происходит ранжирование различных стратегий и параметры каждой адаптируются к значениям лучшей подпопуляции.
- 2) *РГА, основанные на миграции и искусственной селекции (GAMAS).* Эта модификация использует четыре подпопуляции – "породы" I – IV. Сначала создаются "породы" I – IV. Порода I является основной и накапливает найденные лучшие решения. Порода II предназначена для исследования (новых) областей поиска (exploration). Для этого в ней применяется мутация с высокой вероятностью $p_m=0,05$. Порода III используется как для исследования, так и для эксплуатации зон поиска решений и имеет среднее значение вероятности мутации $p_m=0,005$. Порода IV используется для эксплуатации найденной зоны поиска решений (exploitation). Поэтому в ней применяется низкая вероятность мутации $p_m=0,003$. РГА отбирает лучшие особи из пород II-IV и вводит их в породу I, если они лучше уже имеющихся в этой породе (I). Таким образом, порода I сохраняет лучшие хромосомы,

появляющиеся в других породах. Через заданное число поколений ее хромосомы вводятся в породу IV и вытесняют ее текущие особи.

3) Неоднородные РГА с различным кодированием.

В этой модификации РГА, иногда называемой "модель островов с инъекцией", в каждой подпопуляции особи кодируются с различной точностью. Это дает возможность исследовать пространство поиска в разных подпопуляциях с различным шагом. Из подпопуляции с "крупной сеткой" лучшие особи "впрыскиваются" в подпопуляцию с "мелкой сеткой". При этом подпопуляция с низкой точностью имеет меньшую размерность пространства поиска и поэтому зона возможного экстремума обычно находится быстрее. Далее лучшие особи "впрыскиваются" в подпопуляцию с большей точностью, где решение уточняется с меньшим шагом.

4) Известны также модификации РГА, в которых в разных подпопуляциях используются арифметические операторы кроссинговера с различной точностью. Здесь, как и в предыдущей модификации, производится впрыскивание лучших решений из "грубой" подпопуляции в "тонкую" для последующей "доводки".

В целом распределенные ПГА более целесообразно использовать для повышения качества решения, в том случае, если его не удастся достичь с помощью обычного (последовательного) ГА. При этом мощность каждой подпопуляции должна быть достаточно большой, чтобы не происходило «вырождение особей», которое часто ведет к преждевременной сходимости в локальном экстремуме. Увеличение быстродействия по сравнению с последовательным ГА также имеет место, но может быть меньше чем у модели «рабочий -хозяин». Для повышения быстродействия в этом случае мощность подпопуляций должна быть небольшой, что существенно увеличивает быстродействие (ценой потери качества решения).

5.5. Клеточные ГА

Модель клеточных ГА (cellular GA), часто называемой также диффузией или "модель с тонкой структурой" (fine grain model), основана на пространственно распределенной популяции, в которой эволюционные взаимодействия возможны только с (в некотором смысле) ближайшими соседними особями. При этом особи обычно расположены в узлах некоторой регулярной структуры – сетки размерности $d=1,2$ или 3. Параметрами КГА являются: тип и топология сетки, размерность структуры, тип окрестности, вид отбора особей и т.п. КГА итеративно рассматривают взаимодействие группы особей, принадлежащих определенному локальному окружению. В простейшем случае рассматривается окрестность Фон-Неймана, где центральный элемент и его четыре ближайших соседа по вертикали и горизонтали (как это показано на рис.5.1 в) образуют небольшой пул, в котором применяются генетические операторы. В каждом поколении КГА рассматривает в качестве центрального элемента окрестности только одну особь. Так как особь может принадлежать только нескольким окрестностям, то ее изменение влияет на соседей "мягко". Это обеспечивает хороший компромисс между медленной сходимостью и расширением пространства поиска.

Эта модель работает с каждой особью индивидуально и выбирает партнера для скрещивания подобно локальному отбору. Таким образом, имеет место диффузия информации в популяции. Интересно отметить, что в процессе поиска решения возникают и эволюционируют виртуальные острова – смежные области особей с примерно одинаковыми значениями ЦФ, что хорошо видно на рис.5.6.

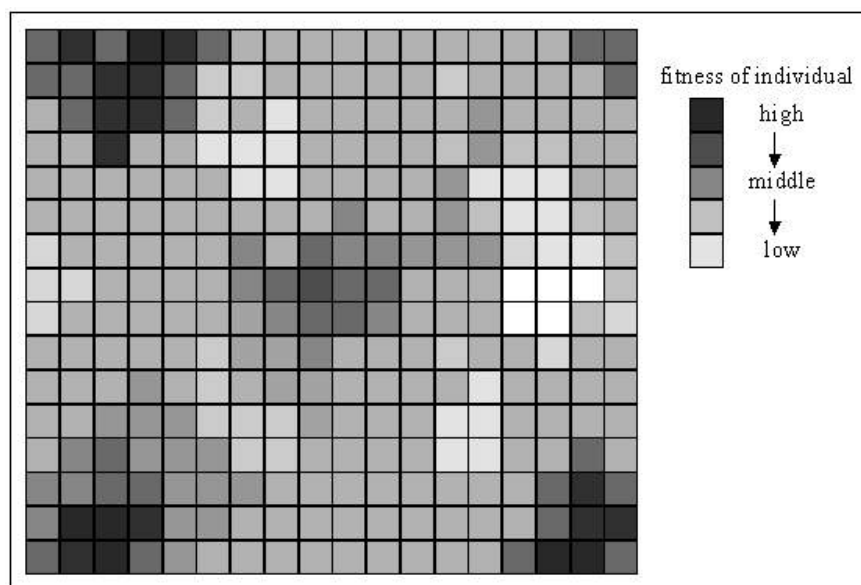


Рис.5.6. Виртуальные острова вследствие диффузии информации.

По синхронизации взаимодействия соседних элементов различают синхронные и асинхронные КГА. В синхронных КГА по синхросигналу (одновременно) вычисляется все новое поколение и записывается во временную (буферную) популяцию, которая затем заменяет старую популяцию согласно следующему алгоритму.

Синхронный клеточный ГА (ГА)

```
{
  для s=1 до МАКС_ ЧИСЛО_ПОКОЛЕНИЙ
  {
    для x=1 до ШИРИНА
    {
      для y=1 до ВЫСОТА
      {
        сосед_список=вычисление_соседей(ГА, позиция(x,y));
        родитель1=выбор(сосед_список);
```

```

    родитель2=выбор(сосед_список);
    кроссинговер(Pc, родитель1,родитель2, потомок);
    мутация(Pm, потомок);
    фитнесс_особи=вычисл_фитнесс(декодирование(потомок));
    внедрение_особи(времен_популяция, позиция(x,y),особь,
                    [если фитнесс_особи лучше или всегда]);

    }
}
текущ_популяция=врем_популяция;
сбор_стат_данных_популяции(текущ_популяция);
}
}

```

Клеточные ГА часто рассматриваются как стохастические клеточные автоматы, где мощность множества состояний равна числу точек в пространстве поиска решений. В синхронном КГА все клетки формально одновременно изменяют свое значение. Разработаны также и асинхронные КГА, где изменение клеток происходит "по событию" – изменению соседних элементов структуры.

5.6 Коэволюционные ГА

Данный тип ГА заимствует у природы явления кооперации и конкуренции и использует обычно две подпопуляции (в общем случае число подпопуляций может быть и больше). Разработаны несколько видов коэволюционных ГА, которые разделяются на кооперативные и конкурирующие ГА. На практике более распространены кооперативные коэволюционные ГА. Здесь взаимодействие между подпопуляциями

осуществляется только за счет оценки значений фитнес-функций. При этом значение фитнес-функции особи зависит от ее способности "сотрудничать" с особями других подпопуляций. Этот подход часто позволяет произвести декомпозицию сложной проблемы на несколько менее сложных задач, каждая из которых решается с помощью ГА. Данный тип ПГА наиболее широко применяется при решении задач многокритериальной оптимизации.

5.7. Параллельная реализация ГА

На этом этапе при заданной структуризации ГА реализуется собственно параллелизация. При этом ожидаются следующие преимущества:

- 1) поиск альтернативных решений одной и той же проблемы;
- 2) параллельный поиск из различных точек в пространстве решений;
- 3) допускают хорошую реализацию в виде островов или клеточной структуры;
- 4) большая эффективность поиска даже в случае реализации не на параллельных вычислительных структурах;
- 5) хорошая совместимость с другими эволюционными и классическими процедурами поиска;
- 6) существенное повышение быстродействия на многопроцессорных системах.

Рассмотрим основные современные методы реализации ПГА. Наиболее известной является представленная на рис.5.7 а) **глобальная параллелизация**. Эта модель основана на простом (классическом) ГА с вычислениями, выполняемыми параллельно. Она быстрее, чем классический ГА, выполняемый последовательно, и обычно не требует баланса по загрузке поскольку на разных процессорах чаще всего вычисляются **значения** фитнес-функций для различных особей (имеющие

примерно одинаковую вычислительную сложность). Исключение составляет генетическое программирование, где различные особи могут сильно отличаться по своей сложности (древовидные или граф-структуры). Эту модель часто называют "*раб (рабочий) - хозяин*". Многие исследователи используют пул процессоров для повышения скорости выполнения алгоритма, поскольку независимые запуски алгоритма на различных процессорах выполняются существенно быстрее чем на одном процессоре. Отметим, что в этом случае нет никакого взаимодействия между различными прогонами алгоритма. Это чрезвычайно простой метод выполнения одновременной работы (если это возможно) и он может быть очень полезным. Например, он может быть использован для решения одной и той же задачи с различными начальными условиями. В силу своей вероятностной природы ГА позволяют эффективно использовать этот метод. При этом ничего нового в сам алгоритм по сути ничего не вносится, но выигрыш во времени может быть значительным.

На рис. 5.7. б) представлена также чрезвычайно популярная "*модель островов*" (*coarse grain*), где множество подалгоритмов совместно работают параллельно, обмениваясь в процессе поиска некоторыми особями. Эта модель допускает прямую реализацию на компьютерных системах с MIMD- архитектурой. При этом каждый "остров" соответствует своему процессору. Очевидно, модель характеризуется параметрами, которые были представлены в разделе 5.3.

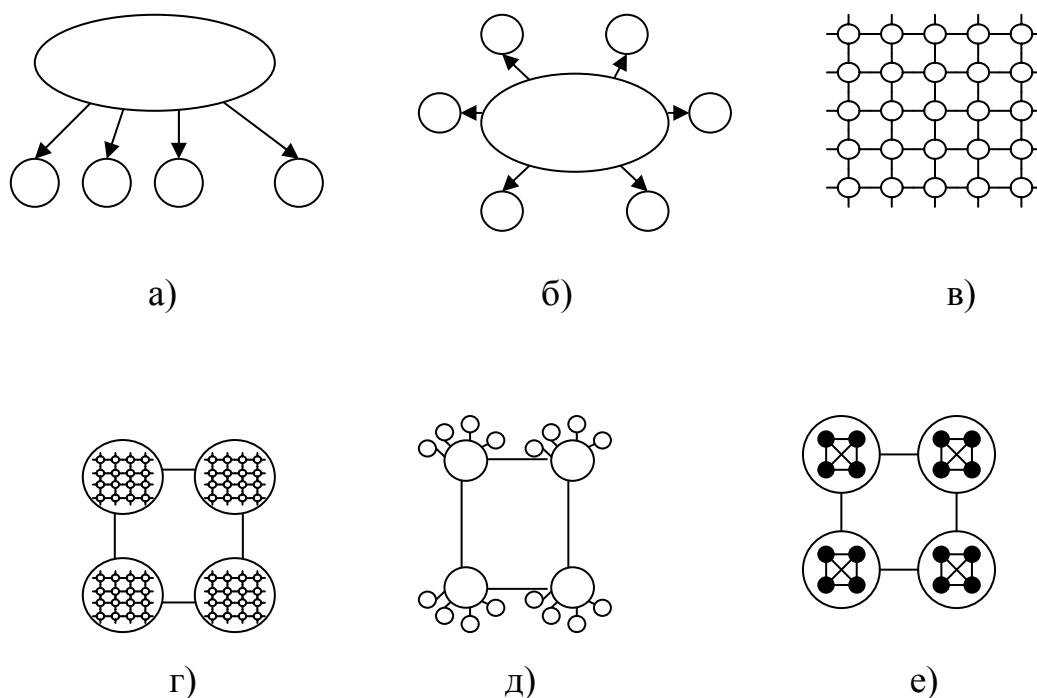


Рис.5.7. Различная реализация параллельных ГА.

В *клеточных ГА (fine grain)*, показанных на рис.5.7 в), распараллеливание обычно реализуется на компьютерных системах с SIMD-архитектурой, где каждый процессор представляет подпопуляцию (из одной особи). Хотя известны работы, в которых используются однопроцессорные компьютеры и системы с MIMD-архитектурой.

На рис.5.7 г-е) представлены гибридные модели, в которых при распараллеливании используется двухуровневый подход. В данном случае на верхнем уровне для распараллеливания применяется РГА (coarse-grain implementation). В модели рис.5.7 .г) каждый из островов реализуется (на нижнем уровне) в виде клеточного ГА, что позволяет объединить преимущества этих двух различных подходов. Модель рис.5.7 д) каждый из островов на нижнем уровне реализуется по схеме глобального распараллеливания - "раб (рабочий) - хозяин". Таким образом, здесь параллелизм используется для ускорения вычислений (на нижнем уровне) и в тоже время он применяется при реализации взаимодействующих подпопуляций – островов (на верхнем уровне). И, наконец, в модели

рис.5.8.е) на верхнем и нижнем уровнях реализуется модель островов. Таким образом, как классический ГА, так и структурированный ГА могут быть реализованы различными способами как на монопроцессорной так и на многопроцессорных компьютерных системах.

5.8. Инструментарий распараллеливания

Как отмечалось выше, для реализации ПГА могут быть использованы компьютерные системы с различными архитектурами: SISD, SIMD, MIMD и т.д. Вместо описания многочисленных специальных архитектур и программных конструкций, которые используются при реализации ПГА, далее мы кратко рассмотрим те параллельные и распределенные модели, которые не зависят от конкретных структур. Почти все ПГА реализуются на основе модели передачи сообщений коммуникации, поскольку она позволяет описывать многопроцессорные системы с распределенной памятью, которые являются наиболее удобным и распространенным средством реализации ПГА. В модели передачи сообщений процессы в одном или физически различных процессорах общаются между собой путем передачи друг другу сообщений через среду коммуникации, которая представлена стандартом или специальной схемой соединения. Основными элементами здесь являются процедуры отправления и приема сообщений. В простейшей форме, отправление определяет локальный буфер передаваемых данных. Процедура приема обычно определяет процесс отправления и локальный буфер, в котором сохраняются входящие данные. В качестве инструментария чаще всего используются следующие средства: сокет (*sockets*), параллельная виртуальная машина (*parallel virtual machine -PVM*), интерфейс передачи сообщений (*message passing interface -MPI*), Ява (*Java*), архитектура общего назначения запрос-посредник (*common object request broker*

architecture) - *CORBA* и *Globus*, которые обеспечивают большие функциональные возможности чем простой сервис передачи сообщений.

5.9. Иерархические (многоуровневые) ГА

Иерархические ГА по своей идеологии примыкают к параллельным, и как мы видели в разделе 5.7, часто используются совместно с ними. Это позволяет комбинировать различные модели ПГА и разрабатывать гибридные ГА. Рассмотрим ГА как сложную систему, в которой есть, по крайней мере, два уровня объектов – хромосомы нижнего уровня, каждая из которых представляет решение конкретной проблемы. Объекты верхнего уровня представляют собой параметры ГА, такие как размер популяции, вероятность скрещивания и мутации, тип ЦФ и т.д. Схематически это показано на рис.5.8.

Заметим, что объекты верхнего уровня существенно влияют на эффективность ГА нижнего уровня. Рассмотрим популяцию структурных объектов верхнего уровня, каждая из которых представляет по сути ГА нижнего уровня. На верхнем уровне работает ГА над ГА нижнего уровня. Здесь в качестве оператора скрещивания может выступать обмен параметрами ГА, а в качестве мутации изменение этих параметров. Таким образом, имеем двухуровневый иерархический ГА.

На нижнем уровне протекают параллельно процессы обычных ГА, в которых каждая особь представляет решение конкретной проблемы и развивается обычным образом. На верхнем уровне в качестве особей используют ГА параметры нижнего уровня.

Здесь каждая особь представляет свой вариант значений параметров ГА нижнего уровня. Таки образом на верхнем уровне подбираются рациональные параметры ГА нижнего уровня, которые далее передаются на нижний уровень для эффективного поиска решения проблемы.

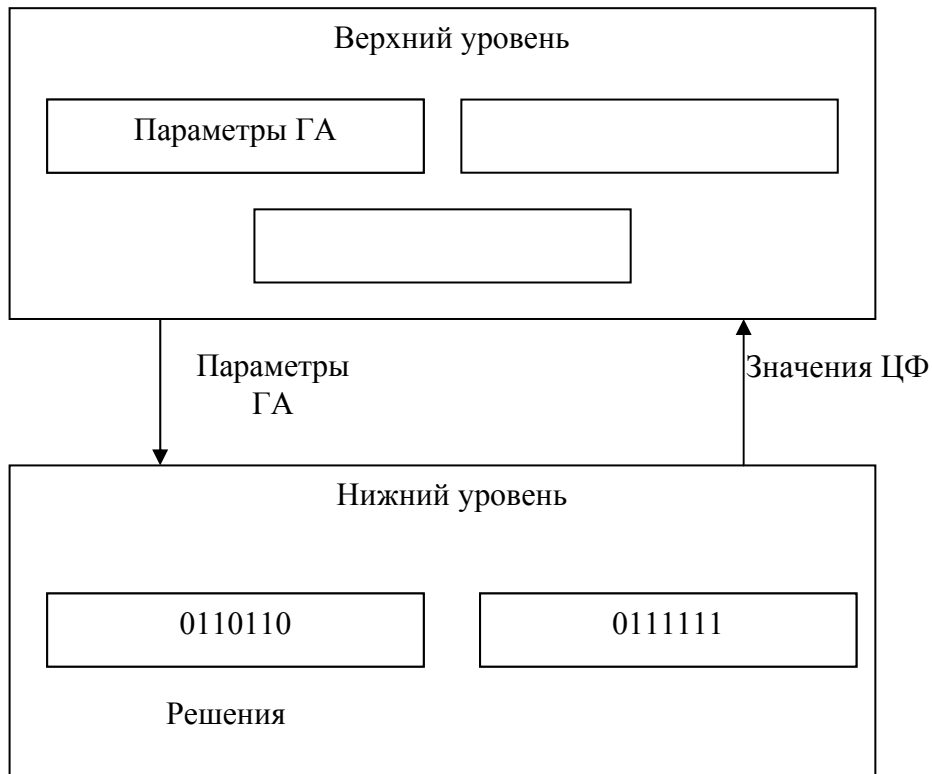


Рис.5.9. Иерархический ГА

Разработаны и многоуровневые (более 2-х) иерархические ГА, которые развивают этот подход.

5.10 Контрольные вопросы к разделу 5

1. Какие свойства ГА способствуют его распараллеливанию?
2. Опишите модель «рабочий - хозяин».
3. Какие функции у процессора – хозяина?
4. Что делает процессор – рабочий?
5. Какой выигрыш дает модель «рабочий - хозяин»?
6. Чем отличается модель «рабочий - хозяин» от «модели островов»?
7. Какие факторы влияют на миграцию в «модели островов»?
8. Какие вы знаете виды распределенных ГА?
9. Опишите клеточные ГА.

10. Что такое виртуальные острова?
11. Что такое коэволюционные ГА?
12. Приведите различные варианты реализации параллельных ГА.
13. Какой инструментарий можно использовать при реализации ГА?
14. Опишите возможный вариант иерархического ГА.

6. ВЕРОЯТНОСТНЫЕ И КОМПАКТНЫЕ ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ

В настоящее время кроме детерминированных ГА разработаны и достаточно широко применяются вероятностные ГА (ВГА), в которых популяция представляется вектором вероятности [32]. Например, популяция, состоящая из двоичных хромосом, может быть представлена вектором P , каждый элемент которого устанавливает относительную частоту появления гена "1" на соответствующей позиции. Пример такого представления для трех различных популяций состоящих из четырех хромосом дан ниже:

1	0	1	0	1	0	1	0	1	0	1	0
1	1	0	0	0	0	1	1	0	1	1	0
1	0	1	0	1	1	0	0	0	1	1	0
1	0	0	0	0	0	1	0	1	0	1	0
<hr/>				<hr/>				<hr/>			

$$P = (1; 0,25; 0,5; 0) \quad P = (0,5; 0,25; 0,75; 0,25) \quad P = (0,5; 0,5; 1; 0).$$

6.1. Вероятностные генетические алгоритмы

Впервые такое представление было введено в [32] (Equilibrium Genetic Algorithm) и взято за основу в последующих работах [33–36]. В [32] на этой основе проведены теоретические исследования вопросов сходимости ГА и показано, что для ряда классических задач вероятностный ГА дает результаты, не уступающие стандартному ГА с однородным кроссинговером и стратегией элитизма при отборе родителей.

В этом случае эволюция популяции соответствует траектории в гиперкубе пространства (p_1, p_2, \dots, p_N) . Траектория начинается в середине

единичного гиперкуба ($p_i = 0,5$ для всех $1 \leq i \leq N$) и заканчивается в одной из его вершин, которая соответствует найденному решению (двоичному коду), как это показано на рис.6.1.

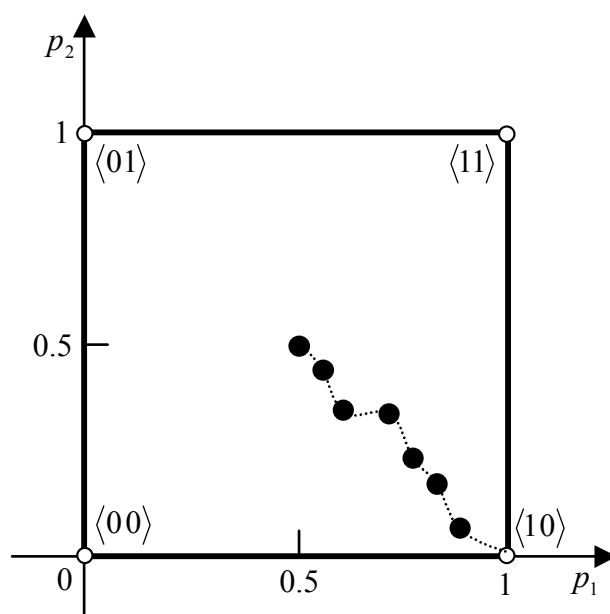


Рис. 6.1. Траектория эволюции популяции на плоскости

В классическом ГА популяция представляется множеством двоичных векторов и изменяется путем применения операторов репродукции, кроссинговера и мутации. «Вероятностный» ГА работает не с исходной популяцией (множеством двоичных векторов), а непосредственно с ее вероятностным представлением – вектором вероятностей $P = (p_1, p_2, \dots, p_N)$. В этом случае [32] генетические операторы выполняются следующим образом.

Репродукция. В соответствии с текущим распределением вероятностей P генерируется некоторое (относительно небольшое) множество двоичных векторов – особей. Для каждой из построенных особей вычисляется значение фитнес-функции. Затем вектор вероятностей P сдвигается в сторону особи v (вершины гиперкуба), имеющей лучшее значение фитнес-функции: $P' = (1 - \theta)P + \theta v$. Здесь $\theta \in (0; 1)$

– фиксированное вещественное число. Если несколько особей имеют наилучшее значение, то случайным образом выбирается одна из них.

Мутация. Каждая координата вектора P корректируется случайным образом с заданной вероятностью π : $p'_i = (1 - \mu)p_i + \mu z_i$, где $\mu \in (0; 1)$ – вещественный параметр, а z_i принимает значения 0 и 1 с равной вероятностью. Таким образом, p_i сдвигается на небольшое расстояние в направлении 0 или 1 в соответствии со случайным значением z_i . В [32] показано, что оператор кроссинговера в этом случае не является необходимым.

Очевидно, что из построенного вектора вероятностей P легко получить двоичный вектор, представляющий решение задачи. Если $p_i = 1$ (или близко к 1), то значение гена $x_i = 1$, в противном случае $x_i = 0$. Эксперименты показали, что для некоторых задач этот подход дает результаты, сравнимые с классическим ГА при меньших затратах вычислительных ресурсов.

6.2. Пошаговое обучение на основе виртуальной популяции

Следующим рассмотрим подход, предложенный в [33] – PBIL (Population-Based Incremental Learning). На основе представления популяции вектором вероятности $P = (p_1, p_2, \dots, p_N)$ вместо стандартного ГА используется следующий алгоритм поиска оптимального решения:

Пошаговое обучение()

{

Инициализация вектора вероятностей P ;

// присваивание $p_i = 0,5$ для всех $1 \leq i \leq N$


```

// основной цикл алгоритма
while (критерий_останова <> TRUE)
{
    // генерация особей виртуальной популяции  $S$  мощности  $M$ 
    while ( $k \leq M$ )
    {
        генерация особи  $s_k$  согласно вектору вероятности  $P$ ;
        оценка значения фитнес-функции для  $s_k$ ;
    }

    // поиск лучшей особи виртуальной популяции
     $max = \text{поиск\_максимума}(S)$ ;

    // изменение вектора вероятности  $P$ 
    while ( $i \leq N$ )
    {
         $p_i = p_i * (1.0 - LR) + max * LR$ 
    }

    // мутация вектора вероятности  $P$ 
    while ( $i \leq N$ )
    {
        if (random (0,1] <  $mut\_prob$ ) then
             $p_i = p_i * (1.0 - mut\_shift) + \text{random}(0 \text{ or } 1) * mut\_shift$ 
        }
    }

    // формирование результирующего вектора  $X = (x_1, x_2, \dots, x_N)$ 

```

```

while (  $i \leq N$  )
{
     $x_i = \text{round}(p_i)$ 
}
}

```

Как видно из псевдокода, виртуальная популяция каждый раз случайным образом генерируется согласно текущему вектору вероятностей, который корректируется в сторону лучшей особи. Метод корректировки вектора вероятностей, по сути, взят из конкурирующего метода обучения нейронных сетей без учителя. Таким образом, здесь «эволюционирует» вектор вероятностей. Этот подход развит многими авторами и некоторые его модификации допускают достаточно простую аппаратную реализацию.

6.3. Компактный генетический алгоритм

Компактный ГА (КГА)[34], как и PBIL, модифицирует вектор вероятности, однако помимо фазы кроссинговера здесь также игнорируется и фаза мутации, что позволяет сконцентрировать все внимание на механизме селекции, который основан на турнирном методе отбора особей. Кроме того, компактный ГА моделирует конечную популяцию и требует меньше вычислительных ресурсов. Псевдокод алгоритма представлен ниже:

```

Компактный ГА
{
    Инициализация вектора вероятностей  $P$ ;

```

```

// присваивание  $p_i = 0,5$  для всех  $1 \leq i \leq N$ 

// основной цикл алгоритма
while (критерий_останова  $\neq$  TRUE)
{
    // генерация двух особей виртуальной популяции мощности  $M$ 
     $a$  = сгенерировать ( $P$ );
     $b$  = сгенерировать ( $P$ );

    // определение победителя и проигравшего
    if (фитнесс ( $a$ ) > фитнесс ( $b$ ))
        then победитель =  $a$ 
        else победитель =  $b$ 

    // изменение вектора вероятности  $P$ 
    while ( $i \leq N$ )
    {
        if (победитель [ $i$ ]  $\neq$  проигравший [ $i$ ]) then
            if (победитель [ $i$ ] = 1) then  $p_i = p_i + \frac{1}{M}$  else  $p_i = p_i - \frac{1}{M}$ 
        }
    }

    return  $P$ .
}

```

Критерием остановки работы алгоритма выступает следующее условие:

$$(p_i > 0) \& (p_i < 1) \text{ для всех } 1 \leq i \leq N.$$

6.4. Генетический алгоритм SELFISH

Данный алгоритм был разработан на основании современной интерпретации Дарвиновской теории механизмов естественного отбора предложенной биологом Ричардом Доукинсом [35]. Ключевым моментом нового подхода стала гипотеза о том, что базовым элементом эволюции является ген, а не особь. В связи с этим рассматриваются хромосомы, гены которых могут принимать различное количество значений в зависимости от своего расположения внутри хромосомы. Поэтому вектор вероятностей трансформируется в более сложную структуру:

$$P = (P_1, P_2, \dots, P_i, \dots, P_N),$$

где

$$P_i = (p_{i1}, p_{i2}, \dots, p_{in_i}).$$

Отметим, что здесь P_i могут иметь различную мощность n_i , поскольку разные гены могут принимать различное число значений. Таким образом, был предложен следующий алгоритм:

Алгоритм SELFISH

{

Инициализация вероятностей;

// присваивание $p_{ij} = \frac{1}{n_i}$ для всех $1 \leq i \leq N$

// инициализация начального значения лучшей хромосомы

$B = \text{выбор_особи}()$;

```

// основной цикл алгоритма
while (критерий_останова <> TRUE)
{
    G1 = выбор_особи ();
    G2 = выбор_особи ();
    // турнирный отбор
    if (фитнесс (G1) > фитнесс (G2)) then
    {
        поощрение (G1);
        наказание (G2);
        if (фитнесс (G1) > фитнесс (B)) then B = G1;
    }
    else
    {
        поощрение (G2);
        наказание (G1);
        if (фитнесс (G2) > фитнесс (B)) then B = G2;
    }
}

return B.
}

```

Здесь функция «поощрение» увеличивает, а функция «наказание» уменьшает текущие вероятности для соответствующих значений всех генов в хромосоме на ε_i . То есть

$$p_{iH_i} = p_{iH_i} \pm \varepsilon_i \text{ для всех } 1 \leq i \leq N,$$

где H – некоторая хромосома.

Функция «выбор_особи» выполняется следующим образом:

```

выбор_особи()
{
    for ( $1 \leq i \leq N$ )
        if (random (0,1) < mut_prob)
            then  $H_i$  = случайный_аллель (1,  $n_i$ ); // мутация
            else  $H_i$  = выбрать_аллель ( $P_i$ );
    return  $H$ ;
}

```

Как видно из псевдокода она, помимо всего прочего, реализует процесс мутирования генов.

В заключение добавим, что критерий останова работы алгоритма определяется как состояние, в котором для каждого гена (локуса) L_i существует аллель (одно из возможных значений) a_{ij} , чья вероятность больше заданного p_i (обычно принимает значение 0,95).

6.5. Сравнение простых и вероятностных генетических алгоритмов

Для апробации описанных в работе алгоритмов использовались традиционные задачи численной и комбинаторной оптимизации. Рассмотрим полученные результаты более подробно.

I. Задачи численной оптимизации (нахождение наибольшего либо наименьшего значения функции многих переменных). Для тестирования использовались функции De Jong (F_1, \dots, F_5). Графическое сравнение эффективности алгоритмов представлено ниже.

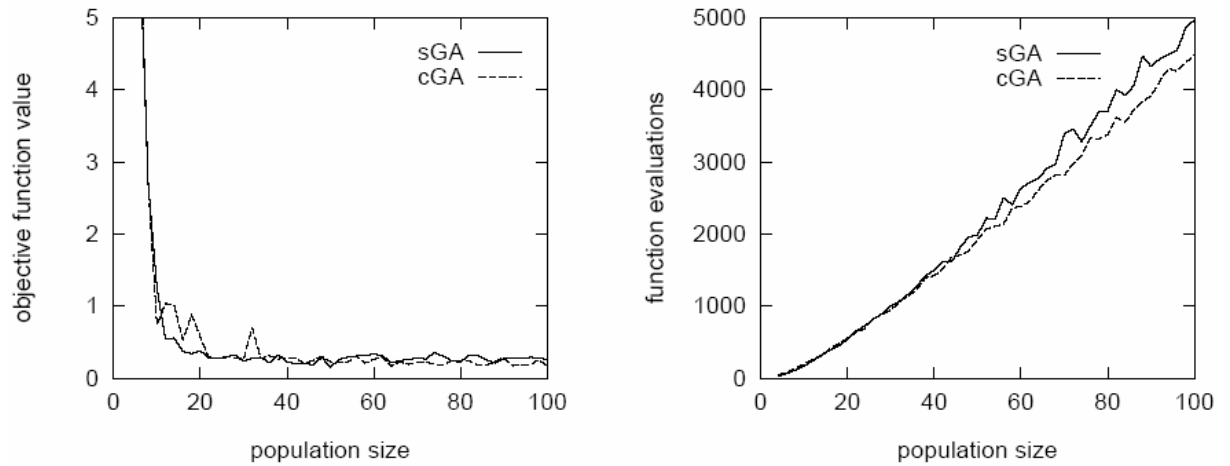


Рис..6.2. Сравнение простого (sGA) и компактного (cGA) ГА.

II. Задачи комбинаторной оптимизации.

1) Задача коммивояжера (Traveling salesman problem).

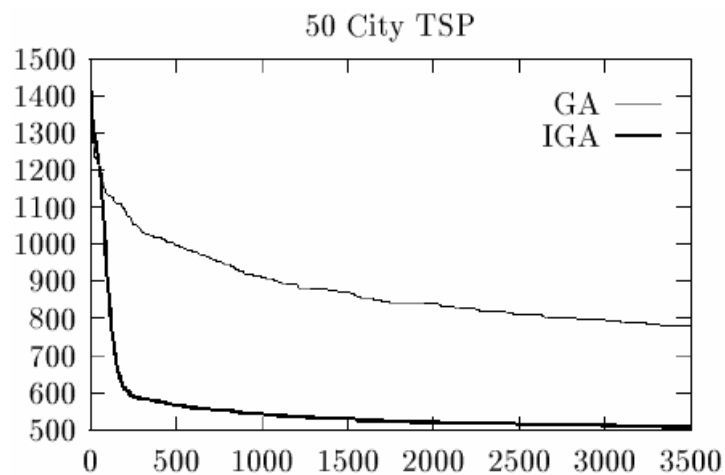


Рис.6. 3. Сравнение простого (GA) и вероятностного(IGA) ГА.

2) Задача упаковки рюкзака (Bin packing).

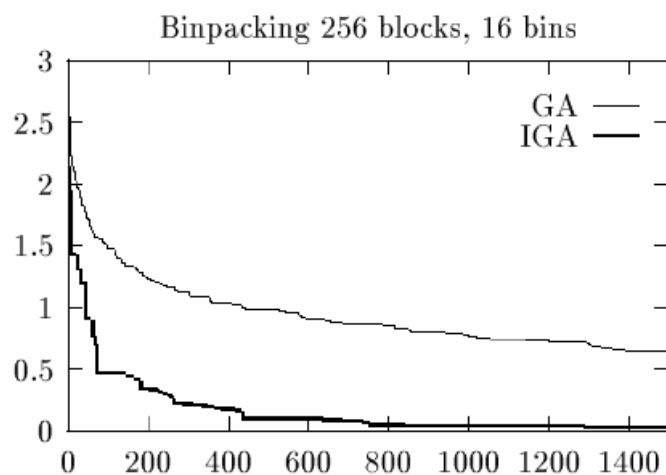


Рис. 6.4. Сравнение простого (GA) и вероятностного (IGA) ГА.

3) Задача календарного планирования (Job-shop scheduling problem).

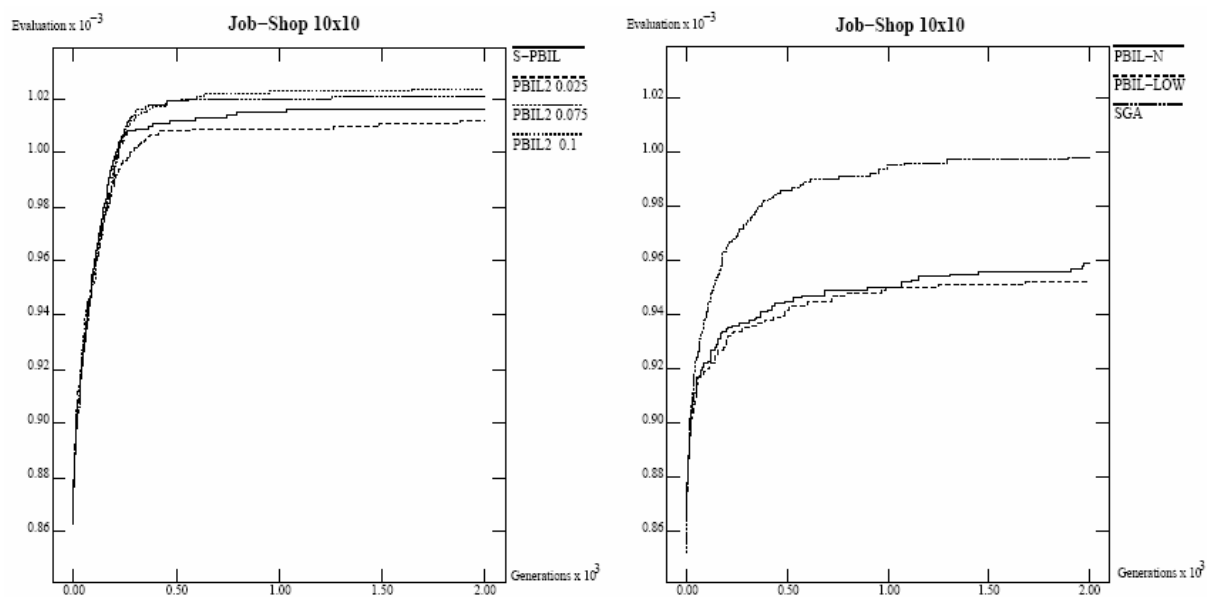


Рис. 6.5 Сравнение простого ГА и различных модификаций PBIL.

Итак, представленное в графической форме сравнение эффективности работы простого (или классического) ГА с вероятностными и компактными ГА практически однозначно свидетельствует в пользу последних. Более подробное освещение данного вопроса можно найти в первоисточниках.

Генетические алгоритмы в последние годы широко и довольно успешно используются при решении задач численной и комбинаторной оптимизации. Их популярность, в первую очередь, связана с той универсальностью, которая заложена в них теорией эволюции Ч. Дарвина. Однако, несмотря на значительные преимущества ГА перед остальными методами поиска, приходится констатировать и ряд недостатков. Так, до сих пор не решена проблема преждевременной сходимости ГА к локальным экстремумам. Очередной попыткой, сделанной исследователями в этом направлении, стали вероятностные и компактные ГА. По сравнению с традиционными ГА они имеют ряд преимуществ. Во-первых, это контроль и управление скоростью сходимости генетического поиска. Во-вторых, это поддержка и сохранение разнородности генетической информации в популяции. И, в-третьих, это экономия вычислительных ресурсов. Пока рано говорить о том, что такие ГА полностью решили проблему преждевременной сходимости, однако они существенно сгладили тот негативный эффект, который она создавала. Таким образом, мы считаем перспективными дальнейшие исследования в этой области и рекомендуем использовать вероятностные и компактные ГА при решении практических задач. Особо следует отметить компактные ГА, которые вследствие своей простоты допускают эффективную аппаратную реализацию.

6.6 Контрольные вопросы к разделу 6

1. Как представляется популяция в вероятностных ГА?
2. Чему соответствует эволюция популяции в вероятностном ГА?
3. Как реализуется оператор репродукции?
4. Как реализуется оператор мутации?
5. Чем отличается пошаговое обучение от вероятностного ГА?
6. Что отличает компактный ГА от других вероятностных ГА?
7. Опишите алгоритм SELFISH.
8. Какие вы видите преимущества и недостатки у вероятностных ГА по сравнению с классическими?
9. Какова сложность реализации вероятностных ГА по сравнению с классическими?
10. Как вы оцениваете экспериментальные результаты по тестированию вероятностных ГА?

ЧАСТЬ 3

ГЕНЕТИЧЕСКОЕ ПРОГРАММИРОВАНИЕ И МАШИННОЕ ОБУЧЕНИЕ

7. ГЕНЕТИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Эксперименты по компьютерному синтезу программ проводились с конца 50-х годов [37] и являлись одной из важнейших компонент машинного обучения, которое относится к одному из самых перспективных направлений искусственного интеллекта. В процессе обучения хромосомы или некоторые структуры, автоматически генерируются с помощью генетических операторов и представляют компьютерные программы различной сложности. Первые эксперименты проводились с использованием двоичных кодов программ и показали скромные результаты обусловленные, в первую очередь, состоянием вычислительной техники и программного обеспечения (ПО) того времени. И только в 80-х годах с развитием достаточно мощной компьютерной техники и ПО сформировалось генетическое программирование (ГП), в первую очередь на основе работ Koza[5].

В генетическом программировании (ГП) в качестве особи выступает программа, представленная в определенном формате, которая решает некоторую задачу. Часто это выполняется с использованием обучающих данных и индуктивного вывода. ГП очень близко к машинному обучению и поэтому в качестве фитнес-функции достаточно часто выступают

функции ошибки (рассогласования, невязки в различной метрике). Следует отметить, что ГП работает с генетическим материалом переменной длины, что требует нестандартной формы представления генома и соответствующих генетических операторов.

7.1. Функциональное и терминальное множество

Программы составляются из переменных, констант и функций, которые связаны некоторыми синтаксическими правилами. Поэтому необходимо определить терминальное множество, содержащее константы и переменные, и функциональное множество, которое состоит, прежде всего, из операторов и необходимых элементарных функций ($\exp(x)$, $\sin(x)$ и т.п.). Следует отметить, что терминалы и функции играют различную роль. Терминалы обеспечивают входные значения в систему (программу), в то время как функции используются при обработке значений внутри системы. Термины «функции» и «терминалы» взяты из древовидного, наиболее часто применяемого, представления программ, которое широко используется в теории формальных языков и грамматик. Терминалы и функции соответствуют узлам древовидных (или графоподобных) структур.

7.1.1. Терминальное множество

Терминальное множество включает в себя: 1) внешние входы в программу; 2) используемые в программе константы; 3) функции, которые не имеют аргументов. Слово «терминал» используется, так как перечисленные выше объекты соответствуют терминальным (конечным, висячим) вершинам в древовидных структурах и соответственно терминалам в формальных грамматиках. Терминал дает некоторое

(численное) значение, не подвергаясь никаким входным значениям. У него нет входных аргументов, и он имеет нулевую «арность».

Следует отметить тесную связь внешних входов с обучающими выборками, которые часто используются в ГП. При этом каждая переменная (признак, фактор и т.п.) обучающей выборки соответствует своему терминалу. В этом смысле ГП отличается от других методов машинного обучения. Здесь переменная выборки не привязана прямо к входным данным – связь производится через терминалы. При необходимости терминал можно связать с другой внешней переменной.

Терминальное множество включает в себя также константы. В классическом ГП, основанном на древовидном представлении, множество числовых констант выбирается для всей популяции и не меняется в процессе эволюции. Но в линейном ГП (использующем линейное представление программы) случайно генерированные (обычно из заданного диапазона) константы могут мутировать при поиске решения.

7.1.2. Функциональное множество

Функциональное множество состоит из операторов (взятых у языков программирования) и различных функций. Оно может быть достаточно широким и включать типовые конструкции различных языков программирования, такие как:

- булевы функции И, ИЛИ, НЕ и т.п.;
- арифметические функции сложения, вычитания, умножения, деления;
- трансцендентные функции (тригонометрические, логарифмические);
- функции присваивания значения переменным ($a:=2$);
- условные операторы (if then, else: case или switch операторы ветвления);
- операторы переходов (go to, jump, call- вызов функции);

- операторы цикла (while do, repeat until, for do);
- подпрограммы и функции.

С одной стороны, терминальное и функциональное множество должны быть достаточно большими для представления потенциального решения. Например, вряд ли функциональное множество из операторов сложения и вычитания может быть эффективно использовано при решении достаточно сложных проблем. С другой стороны не следует сильно без необходимости расширять функциональное множество, поскольку при этом резко возрастает пространство поиска решений.

Конечно набор функций существенно зависит от решаемой задачи. Можно начинать с простейшего множества, состоящего из арифметических операторов сложения, вычитания, умножения, деления и логических - И, ИЛИ, НЕ, ИСКЛЮЧАЮЩЕЕ ИЛИ.

Это также относится к константам. Во многих реализациях используется 256 узлов для представления функций и терминалов. Например, 56 используются для кодирования функций и 200 для констант. Важным свойством функционального множества является его замкнутость относительно принимаемых значений. То есть каждая функция должна принимать все значения, которые могут принимать ее аргументы. Самым известным контрольным примером является обычное деление, в котором второй аргумент (делитель) не может принимать нулевое значение. В этом случае может быть аварийный останов. Поэтому иногда используют «защищенное» деление, которое обрабатывает указанную ситуацию, возвращая в этом случае, например, некоторое большое число или ноль. Желательно, чтобы все функции (корень квадратный, логарифм и т.п.) имели подобную «защиту».

7.2. Структуры для представления программ

Терминалы и функции должны быть объединены по определенным правилам в некоторые структуры, которые могут представлять программы и использоваться при их выполнении. Выбор структуры существенно влияет на порядок выполнения программы, распределение и использование локальной или глобальной памяти.

В настоящее время наиболее распространенными структурами для представления особей (потенциальных решений проблемы) являются:

- 1) древовидное представление;
- 2) линейная структура;
- 3) графоподобная структура.

7.2.1. Древовидное представление

Значительная часть работ в области ГП, в которых были получены положительные результаты, выполнялась на языке программирования задач искусственного интеллекта LISP, где

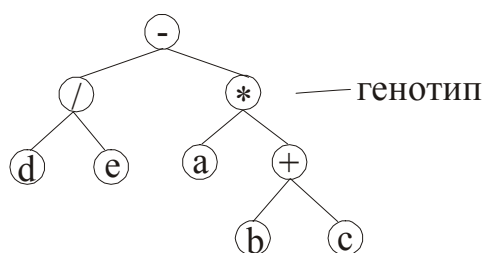


Рис.7.1. Древовидное представление формулы $d/e-a*(b+c)$.

программу удобно представлять в виде дерева. Поэтому в ГП была предложена древовидная форма генома. Мы в качестве примера для удобства будем использовать арифметические формулы, которые также

удобно представлять деревом. Рассмотрим арифметическую формулу $\frac{d}{e} - a * (b + c)$ (в обычном представлении). Этой формуле соответствует S-выражение $((/de)(*(+bc)))$, которое по сути является префиксной (польской) записью формулы, где знак операции стоит перед двумя аргументами. Следует отметить, что скобки здесь при желании можно убрать. Часто используется также и постфиксная запись формулы, где знак операции стоит после аргументов – $((de/)(a(bc+)*)-)$. Легко заметить, что дерево (генотип) рис.7.1 также представляет эту формулу (фенотип) $\frac{d}{e} - a * (b + c)$.

При этом листья дерева соответствуют терминалам, а внутренние узлы – функциям. Заметим, что префиксная и постфиксная запись может быть получена из дерева путем различного обхода дерева. Например, постфиксная запись строится из дерева по Кнуту [38] с помощью следующего обхода:

- 1) обход левого дерева снизу;
- 2) обход правого дерева снизу;
- 3) посещение корня.

Для нашего примера вершины дерева посещаются в следующем порядке: $d \rightarrow e \rightarrow / \rightarrow a \rightarrow b \rightarrow c \rightarrow + \rightarrow * \rightarrow -$.

Древовидная форма представления генотипа оказывается для данного класса задач более эффективной, и позволяет работать с программами или выражениями различной длины. Важным аспектом является также использование памяти при выполнении программы. Древовидная структура позволяет использовать только локальную память в процессе выполнения. Локальность памяти встроена в саму древовидную структуру. Значения переменных доступны для функции только в дереве, корню которого соответствует функция. Например, значения переменных d, e являются локальными относительно узла $/$.

7.2.2. Линейные структуры

Древовидное представление особей первоначально было ориентировано на программы, написанные на LISP, и менее подходит, например, для программ, написанных на Си. Далее будет рассмотрен один из возможных вариантов линейной структуры ГП, ориентированного на подмножество Си [34]. При этом каждая особь (программа) представлена последовательностью переменной длины операторов Си. На рис.7.2 представлен пример такой программы [39].

```
void int u
double v[8]
{
...
v[0]=v[5]+75;
v[7]=v[0]-69;
if (v[1]>0)
    if (v[4]>21)
        v[4]=v[2]*v[1];
v[2]=v[5]+v[4];
v[6]=v[4]-4;
v[1]=sin(v[6]);
if(v[0]>v[1])
    v[5]=v[7]+115;
if(v[1]<=v[6])
v[1]=sin(v[7]);
}
```

Рис.7.2. Линейное представление программы.

Здесь функциональное множество ("instruction set" или "functional set") состоит из арифметических операций, условных операторов if и

вызовов функций. Общая нотация для операторов каждого типа представляется в таблице 7.1.

Таблица 7.1.

Тип оператора	Общая нотация
Арифметический	$v_i := v_j \text{ op } v_k \mid c, \text{ op} \in \{+, -, /, *\}$
Условный	$\text{if}(v_i \text{ cmp } v_k \mid c) \text{ cmp} \in \{>, \leq\}$
Вызов функции	$v_j = f(v_k), \quad f \in \{\sin, \cos, \text{sqrt}, \log, \exp \dots\}$

Из табл.7.1 видно, что за исключением условных операторов, все операторы имеют явную операцию присваивания переменной v_i . Это позволяет использовать программы с "кратными выходами", которые в процессе выполнения изменяют и передают на выход значение нескольких переменных в отличие от программ построенных на древовидных структурах, где на выход передается, как правило, значение только 1-й переменной связанной с корнем дерева. Все операторы выполняются либо над двумя переменными, либо над переменной и константой. До начала работы программы переменным должны быть присвоены соответствующие значения.

Переменные и константы вместе образуют множество терминальных символов. При этом подходе любой оператор кодируется 4-х мерным вектором, компонентами которого является тип оператора (одна компонента) и адреса (указатели) переменных и констант.

Например, оператор $v_i = v_j + c$ представляется вектором $(+, i, j, c)$. Каждая компонента использует 1 байт памяти, следовательно, число переменных (и констант) ограничено сверху 256.

Такое представление позволяет эффективно выполнять рекомбинацию программы и их интерпретацию. Для частично

определенных операторов и их функций, в случае неопределенных значений на выходе возвращается "-1". Последовательности условных операторов `if` интерпретируются как вложенные условные операторы (как это трактуется в Си). В случае ложного значения условия оператора пропускается один следующий по порядку оператор. Такая интерпретация условных операторов дает, с одной стороны, достаточно выразительную мощность, и, с другой стороны, упрощает в дальнейшем работу по обнаружению и устранению интронов, что само по себе представляет в ГП значительную проблему. Следует отметить, что в линейном представлении, в отличие от древовидного, для функции нет очевидного способа определения значений аргументов (в древовидном представлении они однозначно определяются узлами, находящимися ниже соответствующего функционального узла). Существенным отличием является также глобальное использование памяти при выполнении программы в отличие локального в древовидных структурах. Здесь значения переменных доступны для всех функций.

Следует отметить, что данный подход является более выразительным (позволяющим генерировать более сложные программы), чем «регистровая машина» с линейной последовательностью команд, которая часто используется для представления линейных структур [27].

В эволюционном алгоритме при этом подходе обычно используется метод турнирного отбора родителей в промежуточную популяцию, описанный в разделе 4. Мощность популяции обычно поддерживается постоянной.

7.2.3. Графоподобные структуры

Эта форма представления разработана для задач, которые не могут быть решены методами, которые используют рассмотренные выше модели

программ. Данный подход использует более выразительную модель для представления программы [40]. Поэтому данный подход позволяет эффективно решать многие задачи, которые нельзя было решить с помощью предыдущих методов.

Сначала рассмотрим типичную блок-схему программы написанной вручную, которая представлена в качестве примера на рис.7.3.

Здесь каждая особь – программа представляется в виде графа. Вершина графа представляет линейный участок программы. Она имеет 2 части – собственно линейную программу и узел ветвления. Из рисунка видно, что блок-схема линейного графа более естественна, чем линейная (древовидная) форма представления программы.

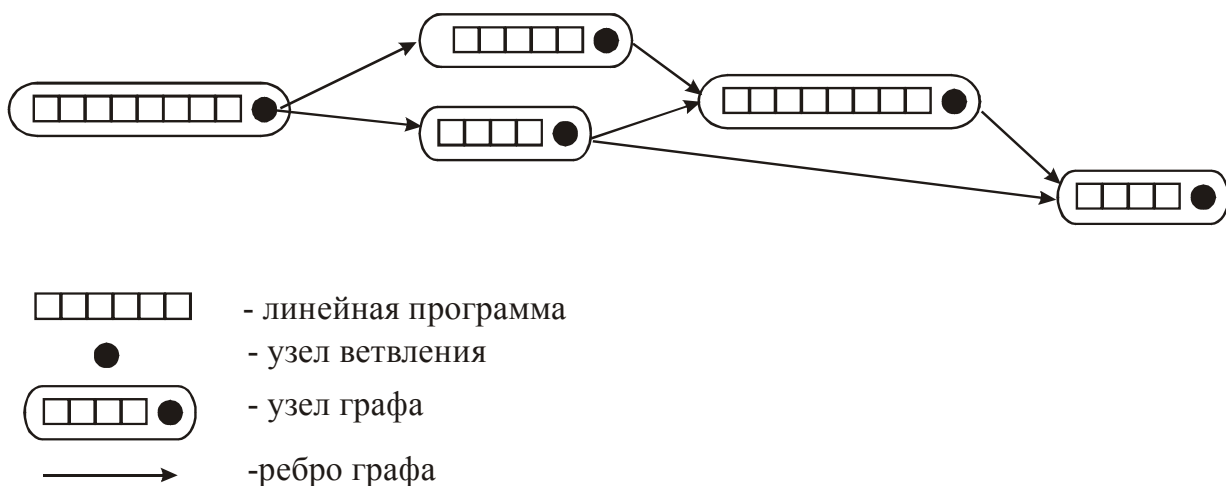


Рис.7.3. Представление программы в виде графа

Это объясняется тем, что многие программы содержат операторы ветвления, после которых может быть вызов другой части кода программы, не входящей в линейную последовательность операторов. Линейная программа выполняется в том случае, когда данный узел графа достигается при интерпретации программы на конкретных данных. После выполнения линейной части выбирается узел-последователь в соответствии с функцией ветвления этого узла. В течение интерпретации

узлы только одного пути графа от входов до выхода будут выполняться в зависимости от текущих значений переменных условий операторов ветвления.

Реализация линейных подструктур использует список операторов Си (или другого языка программирования, в т.ч. и ассемблера) переменной длины, которые оперируют с переменными или константами и полученные значения присваиваются переменным (например, $a = b + 1.33$). После выполнения программы, вычисленное значение запоминается в “выходные” переменные.

Таблица 7.2.

Оператор Ветвления	Описание оператора
$A < 0$ (аккумулятор меньше 0)	Если значение аккумулятора-сумматора меньше нуля то выбирается левый (верхний) преемник, иначе правый (нижний)
$A > 0$	Если значение аккумулятора-сумматора больше нуля то выбирается левый (верхний) преемник, иначе правый (нижний)
$A < op$	Если значение аккумулятора-сумматора меньше значения операнда, то выбирается левый (верхний) преемник, иначе правый (нижний)
$A > op$	Если значение аккумулятора-сумматора больше значения оператора то выбирается левый (верхний) преемник, иначе правый (нижний)
$RD < 0$ (регистр данных)	Если значение регистра данных меньше нуля то выбирается левый (верхний) преемник, иначе правый (нижний)
$RD > 0$	Если значение регистра данных больше нуля то

	выбирается левый (верхний) преемник, иначе правый (нижний)
$RD < op$	Если значение регистра данных меньше значения оператора то выбирается левый (верхний) преемник, иначе правый (нижний)
$RD > op$	Если значение регистра данных больше значения оператора то выбирается левый (верхний) преемник, иначе правый (нижний)

Функция ветвления также является оператором Си, который оперирует с теми же переменными, что и линейная программа, но этот оператор только читает значения этих переменных. Таблица 7.2 содержит множество операторов ветвления, которые могут быть использованы в данной модели.

На рис.7.4 представлен детально узел графа, соответствующий некоторой части линейной программы.

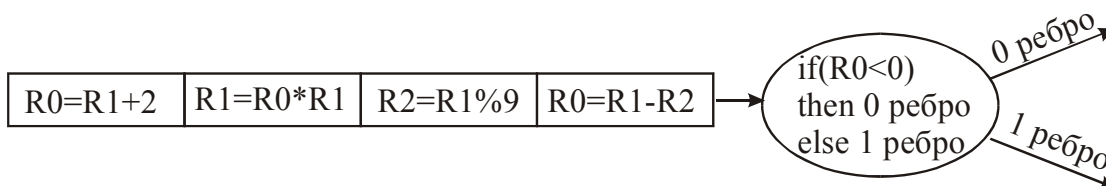


Рис.7.4. Подпрограмма, соответствующая узлу графа.

Данная модель, как и предыдущая, использует при выполнении программы глобальную память. Графоподобные структуры позволяют расширить класс задач, которые могут быть решены с помощью ГП, так

как являются более общими по сравнению с первыми двумя и допускают эффективную реализацию [40].

Известен другой способ применения структур графов в ГП, называемый PADO [27]. Здесь каждая программа представляется ориентированным графом, содержащим узлы, индексированную память для входных переменных и стек, как показано на рис.7.5. В произвольном ориентированном графе каждый узел может иметь N исходящих дуг.

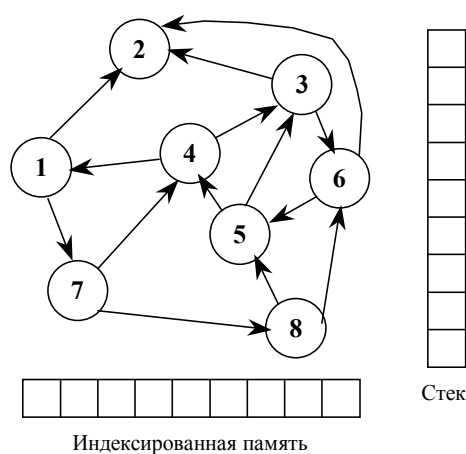


Рис.7.5. Представление программы ориентированным графом.

Эта структура не просто показывает потоки информации и управление ходом выполнения программы. Узел в графе соответствует сегменту программы и имеет две части: «действие» и «ветвление». Часть «действие» содержит константы и функцию, которая выполняется при достижении данного узла в процессе интерпретации программы. Данные передаются между узлами через стек. Часть «действие» получает данные из (верхушки) стека и после выполнения соответствующей функции передает преобразованные данные снова в стек. После этого выполняется «ветвление», которое определяет ветвь следующего выполняемого узла на основании данных стека, памяти или специальных констант ветвления. Очевидно, что в процессе интерпретации

необязательно посещаются все вершины графа. Каждая программа имеет два специальных вершины «старт» и «конец» и кроме этого может содержать некоторые специальные узлы типа «вызов подпрограммы». Поскольку граф может содержать обратные связи, то вершина «конец» при интерпретации на некоторых входных данных может быть недостижимой вследствие «зацикливания». Поэтому необходимо контролировать и ограничивать время выполнения программы.

7.2.4. Другие формы представления программ.

Кроме рассмотренных трех форм представления, в настоящее время предложены многочисленные модификации и обобщения ГП, в которых разработаны различные формы представления особей и определены на них соответствующие генетические операторы кроссинговера и мутации. В табл.7.3 приведены основные структуры, используемые в настоящее время в ГП [36].

В работе [42] для представления программ введены более сложные структуры в узлы дерева. Этот подход основан на хорошо известном методе группового учета аргументов (МГУА) Ивахненко, который широко используется при решении многих задач. При этом внутренние узлы дерева соответствуют полиномам второго порядка, которые используются в МГУА. Далее на указанных древовидных структурах определяются генетические операторы кроссинговера и мутации. При этом обычно используется локальный поиск и замена функций – процедура «переразметки», которая применяется для нахождения оптимальных значений параметров при заданной структуре. При определении фитнес-функции здесь часто используют кратный (множественный) регрессионный анализ. Иногда используются и другие меры, например, сложность особи (например, минимальная длина) или их комбинация.

Таблица 7.3.

Структура	Описание
S-выражение	Древовидная структура
Основана на методе группового учета аргументов (МГУА)	Древовидная структура
Язык регистрового уровня (ТВ)	Линейная постфиксная запись
Язык регистрового уровня (JB)	Линейная префиксная запись
Битовая	Линейные геномы
Битовая	Машинные команды
Абстрактные типы данных	Списки, очереди, стеки
Правила продукций	Грамматики
Клеточное кодирование	Древовидные грамматики
Правила продукций	Графы
Параллельные алгоритмы (PADO)	Графы

Клеточное кодирование [43] также использует структуру графов, которые широко используются в компьютерных науках (для описания электрических цепей, нейронных сетей, конечных автоматов и т.п.), и поэтому область применения такого представления достаточно широкая (в частности, структурный синтез). Основная идея заключается в том, что фенотип отделяется от генотипа, который основан на древовидной структуре. С другой стороны фенотип представляется структурой (возможно циклического) графа. Генотип представляется грамматическим

деревом (разбора), использующим правила грамматического вывода. При этом используются древовидные грамматики, в которых при грамматическом выводе нетерминальный символ заменяется соответствующим деревом. Эти операции выполняются параллельно подобно L-системам [44]. Клеточное кодирование позволяет определить модули, которые последовательно могут применяться в различных участках грамматического дерева.

Генетическое программирование на основе формальных грамматик предложено в работе [45]. Здесь используется очень общая форма ГП, которая основана на использовании теории формальных языков и грамматик. В процессе эволюции особей применяются контекстно-свободные грамматики, что позволяет преодолеть некоторые ограничения в теоретическом плане, характерные для классического ГП. Применение КС-грамматик гарантирует синтаксическую корректность потомков в процессе эволюции и позволяет определить простые и эффективные генетические операторы. Кроссинговер выполняется следующим образом. Случайно выбирается нетерминальный символ в грамматическом дереве первого родителя и затем производится поиск этого же нетерминала во втором родителе. Если во втором родителе этого нетерминала нет, то оператор не выполняется. Иначе производится обмен поддеревьев, соответствующих нетерминальным символам. Мутация выполняется путем «выращивания» грамматического дерева из случайно выбранного нетерминального символа. В дальнейших работах авторы допускают изменение продукций КС-грамматики в процессе эволюции для вывода лучших правил. Известны также работы, в которых используются и более общие контекстно-зависимые грамматики.

В работе [44] определено понятие L-систем, которые первоначально были разработаны для моделирования структур биологических формаций

или в, более общем контексте, развивающихся процессов. В этом подходе замена нетерминальных символов при «выращивании деревьев» выполняется параллельно. Это позволяет развиваться различным ветвям независимо друг от друга, как это имеет место в реальных сложных процессах. При этом целью эволюции является генерация L-системы, множество продукции которой позволяет решать поставленную задачу. Таким образом здесь в качестве особи выступает вся L-система. Каждая L-система оценивается после вывода грамматического дерева. Следует отметить, что такие особи не могут изменяться произвольным образом, а развиваются согласно правилам мета-грамматики.

Интересным является применение в ГП методов нечеткой (fuzzy) логики [42]. В частности, нечеткая логика может использоваться в адаптивном ГП для подстройки таких параметров как вероятности кроссинговера и мутации в процессе эволюции, что позволяет повысить эффективность. При этом используется коэволюция правил нечеткого контролера, который управляет параметрами ГП.

Поиск решения на основе генетического программирования может быть реализован согласно алгоритму, блок-схема которого представлена на рис.1.1. Но, учитывая специфику кодирования особи, имеющую переменную длину и сложную структуру, реализация каждого этапа этого алгоритма имеет свои особенности, которые мы рассмотрим ниже.

7.3. Инициализация начальной популяции

Данный этап в ГП является не таким простым, как в классическом ГА, где генерация случайных двоичных строк не представляет особых проблем. Это связано с различной сложностью особей и их структурой. Естественно методы инициализации начальной популяции различны для

разных форм представления программ. Одним из важнейших параметров в ГП является максимально возможный размер (сложность) программы.

7.3.1. Инициализация древовидных структур

Для деревьев в качестве меры сложности используется максимальная глубина (иногда высота) дерева или общее число узлов в дереве. Глубиной узла называется минимальное число узлов, которые необходимо пройти от корня дерева к этому узлу. Максимальной глубиной дерева D_m называется максимально возможная глубина в дереве для терминального символа (листа). Если арность каждого узла равна двум, то общее число узлов не превышает 2^{B_b} , которое также используется в качестве меры сложности.

Инициализация древовидных структур выполняется путем случайного выбора функциональных и терминальных символов при заданной максимальной глубине дерева. Пусть для определенности выбраны следующее терминальное $T=\{a,b,c,d,e\}$ и функциональное $F=\{+,-,*,\%\}$ множества, где $\%$ означает деление нацело. Применяются два основных метода: 1) полная (full) и 2) растущая (grow) инициализация [5].

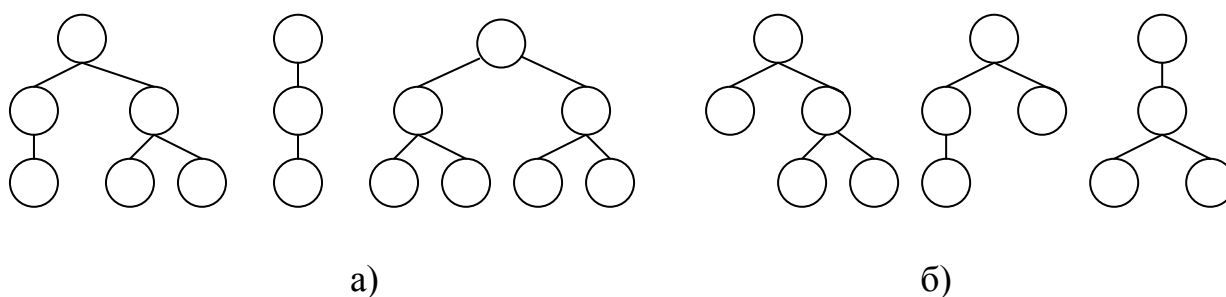


Рис.7.6. Деревья, генерируемые при инициализации разными методами.

В полном методе при генерации дерева, пока не достигнута максимальная глубина, допускается выбор только функциональных символов, а на последнем уровне (максимальной глубины) выбираются

только терминальные символы. Например, на рис.7.6а) представлено дерево с $D_m=3$.

В растущей инициализации генерируются нерегулярные деревья с различной глубиной листьев вследствие случайного на каждом шаге выбора функционального или терминального символа. Здесь при выборе терминального символа рост дерева прекращается по текущей ветви и поэтому дерево имеет нерегулярную структуру.

Например, для $D_m=3$ при растущей инициализации для указанных функциональных и терминальных символов могут быть построены деревья, приведенные на рис.7.6б).

При использовании первого метода начальная популяция содержит однородное множество структур, что способствует вырождению генетического материала (и преждевременной сходимости в локальных экстремумах). Поэтому на практике часто эти два метода используют одновременно следующим образом. Начальная популяция генерируется так, чтобы в нее входили деревья с разной максимальной длиной примерно поровну (для нашего примера $D_m=1$, $D_m=2$, $D_m=3$, $D_m=4$). Для каждой глубины первая половина деревьев генерируется полным методом, а вторая – растущей инициализацией.

7.3.2. Инициализация линейных структур

В этом случае процесс инициализации выполняется совершенно по-другому. Прежде всего, особь – программа разбивается на следующие четыре части:

- 1) Заголовок;
- 2) тело;
- 3) «подвал»;
- 4) выход (возврат).

Из них только тело программы генерируется с помощью эволюции, остальные части программы используются стандартные и заготавливаются заранее. Алгоритм инициализации можно сформулировать следующим образом:

- 1) Выбор случайной длины из заданного диапазона;
- 2) Копирование заготовленного заголовка;
- 3) Инициализация и пополнение собственно операторов в программу пока не достигнута длина, определенная в пункте 1. Операторы выбираются случайно, сначала тип, затем переменная или константа из заданного диапазона;
- 4) Копирование в конец программы заготовленного «подвала»;
- 5) Копирование в конец программы заготовленных операторов выхода.

7.4. Кроссинговер в генетическом программировании

В начальной популяции особи (потенциальные решения), как правило, имеют низкие (плохие) значения фитнес-функции. В процессе эволюции с помощью генетических операторов популяция развивается и значения фитнес-функции улучшаются. В ГП используются те же, что и в ГА, генетические операторы кроссинговера, мутации и репродукции. Отметим, что в терминах машинного обучения они соответствуют операторам поиска решения. Далее мы рассмотрим выполнение генетических операторов на ранее определенных структурах.

7.4.1. Выполнение кроссинговера на древовидных структурах

Для древообразной формы представления используются следующие три основных оператора кроссинговера (ОК):

- а) узловой ОК;

- б) кроссинговер поддеревьев;
- в) смешанный.

В узловом операторе кроссинговера выбираются два родителя (два дерева) и узлы в этих деревьях. Первый родитель называется доминантом, второй – рецессивом. Узлы в деревьях могут быть разного типа. Поэтому сначала необходимо убедиться, что выбранные узлы у родителей являются взаимозаменяемыми. Если узел во втором родителе не соответствует типу узла первого родителя, то случайным образом выбирается другой узел во втором родителе, который опять подлежит проверке на предмет совместимости. Далее производится обмен узлов.

Рассмотрим ОК на следующем примере, представленном на рис.7.7

для родительских особей: $\frac{3}{4} * x^2 + x * y$, $\frac{x}{2} + y + z * y$

В кроссинговере поддеревьев родители обмениваются не узлами, а определяемыми ими поддеревьями. Оператор выполняется следующим образом:

1. Выбираются родители (один – доминантный, другой – рецессивный). Далее необходимо убедиться, что выбранные узлы взаимозаменяемы, т.е. принадлежат одному типу. Иначе, как и в предыдущем случае в рецессивном дереве выбирается другой узел с последующей проверкой.
2. Затем производится обмен поддеревьев, которые определены этими узлами.
3. Вычисляется размер ожидаемых потомков. Если ожидаемый размер (сложность потомка) не превышает заданный порог, такой обмен ветвями запоминается. На рис.7.8 показан пример выполнения этого ОК.

Этот тип ОК является основным. При этом под размером (под)дерева понимается, как и ранее, либо его высота, либо число его вершин .

При *смешанном операторе кроссинговера* для некоторых узлов выполняется узловой ОК, а для других - кроссинговер поддеревьев.

Отметим, что выполнение кроссинговера на древовидных структурах выполняется достаточно просто с помощью S-выражений, что показано в таблице 7.4. Например, кроссинговер поддеревьев сводится к обмену «скобками», которые здесь соответствуют поддеревьям.

В табл.7.5 приведены типовые операторы кроссинговера для древовидных структур[41].

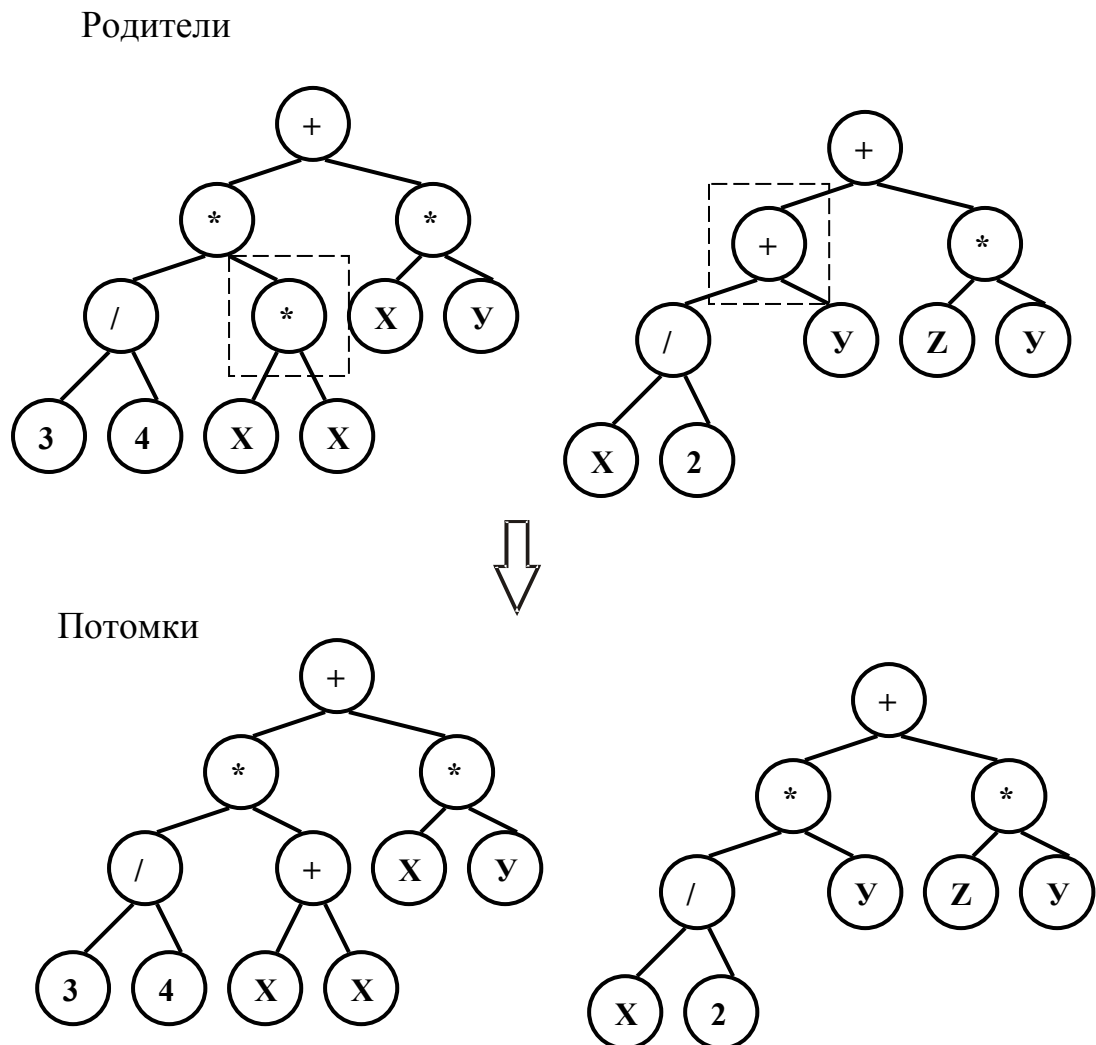


Рис.7.7. Узловой кроссинговер.

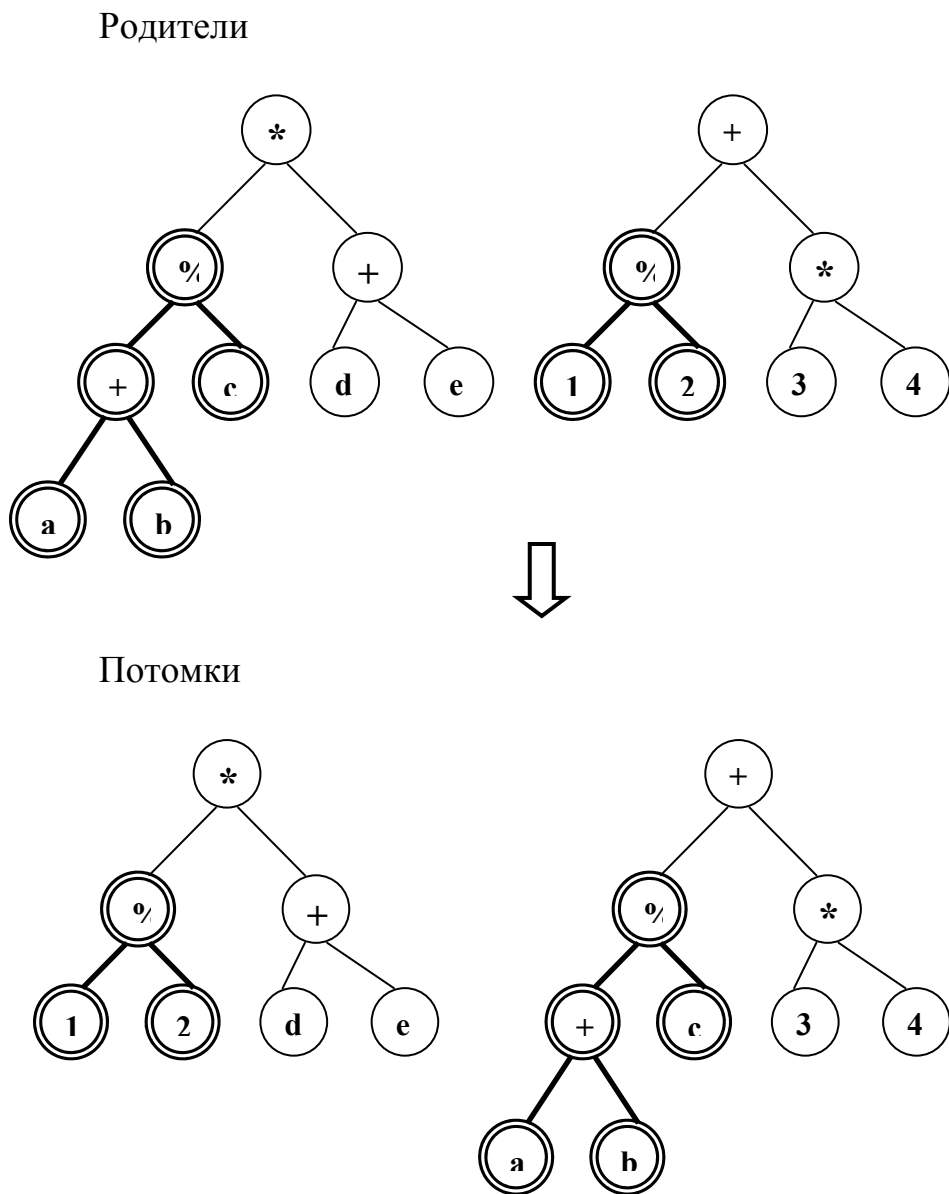


Рис.7.8. Кроссинговер поддеревьев.

Таблица 7.4.

Родители	Потомки
$(- (* a b) c)$	$(- (- (+ d e) f) c)$
$(* (- (+ d e) f) g)$	$(* (* a b) g)$

Таблица 7.5

Наименование	Описание производимых действий
Кроссинговер обмена поддеревьев	Обмен поддеревьями родителей
Само-кроссинговер	Обмен поддеревьями в одном родителе
Модульный кроссинговер	Обмен модулями (фрагментами) родителей
Контекстно-сохраняющий кроссинговер	Полный обмен поддеревьями, если он согласуется с контекстом, иначе частичный обмен

7.4.2. Кроссинговер на линейных структурах

Скрещивание на линейных структурах выполняется достаточно просто. Здесь у родителей выполняется обмен линейными сегментами, как это показано на рис.7.9.

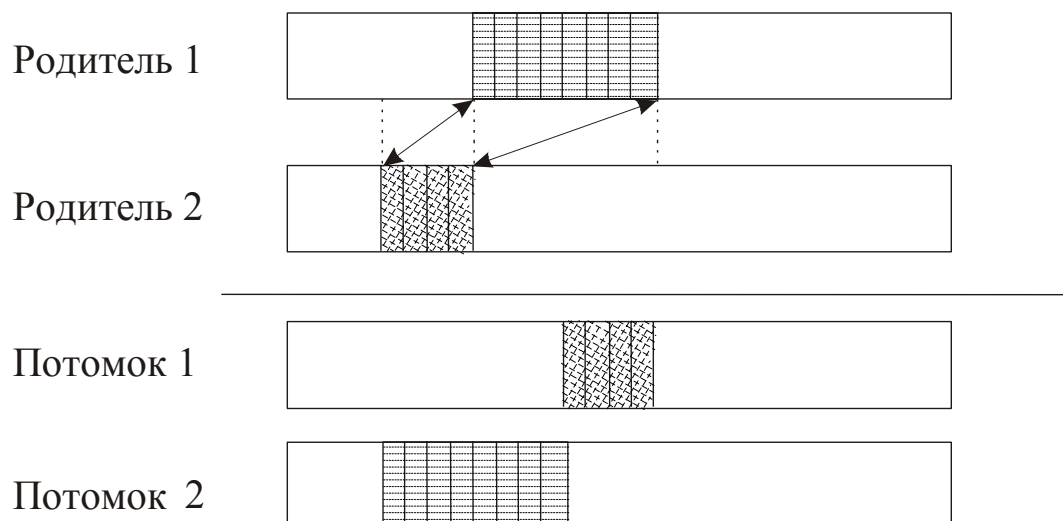


Рис.7.9. Кроссинговер на линейных структурах.

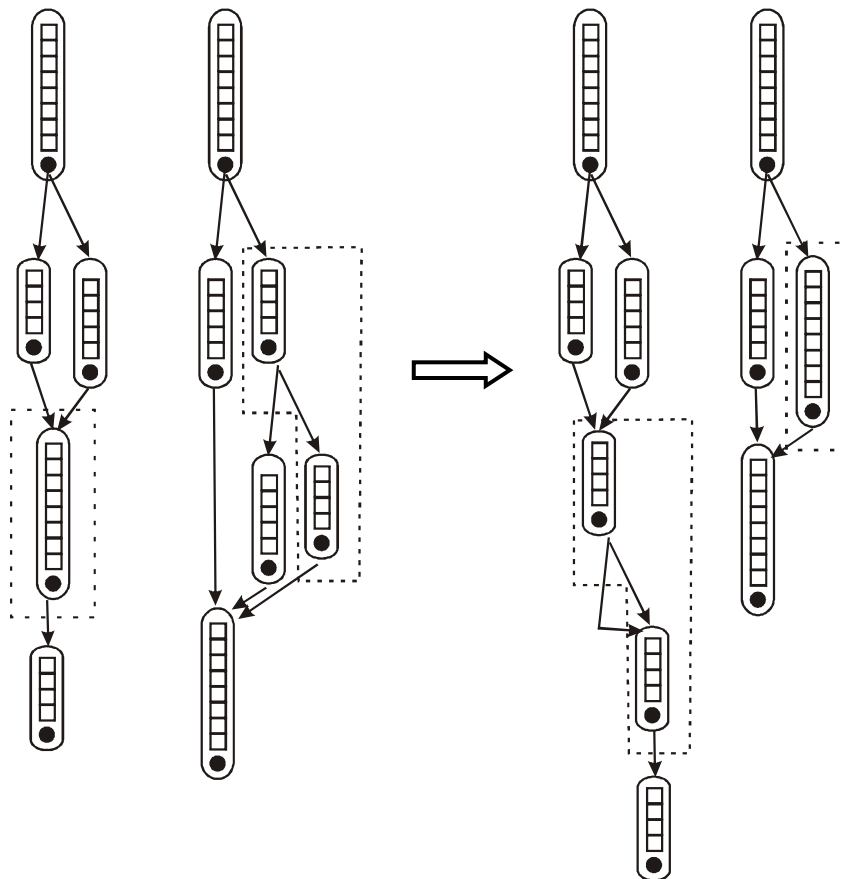
При этом в каждом из двух родителей выбирается сегмент, начиная со случайной позиции и имеющий случайную длину. Далее производится

обмен выбранных сегментов программ. Если размер хотя бы одного из потомков превышает некоторый порог, то результаты ОК аннулируются. При этом точки скрещивания выбираются только между операторами.

7.4.3. Выполнение кроссинговера для графоподобных структур

Кроссинговер комбинирует генетический материал из двух родительских программ путем обмена некоторых частей программ. ОК для этой модели может быть реализован двумя способами.

Первый способ похож на кроссинговер поддеревьев, который определен на древообразных структурах путем обмена поддеревьев. Здесь обмен производится подграфами, как это показано на рис.7.10.



Родитель 1 Родитель 2

Потомок 1 Потомок 2

Рис.7.10. Кроссинговер на графоподобных структурах.

При этом в каждой особи - родителе выбирается случайным образом множество смежных узлов и производится обмен 2-х подграфов между этими подграфами.

Второй тип кроссинговера выполняет линейный ОК. Здесь для каждого родителя выбирается линейный сегмент (в одном узле графа), начинающийся со случайной позиции данного сегмента и имеющий случайную длину. Далее, как обычно, производится обмен этими линейными сегментами. Если размер хотя бы одного потомка превышает некоторый порог, то результаты ОК аннулируются и выполняется обмен равными сегментами меньшей длины. Обычно линейный ОК выполняется в вероятностью $P_l = 0.1$. Пример выполнения этого ОК представлен на рис.7.11.

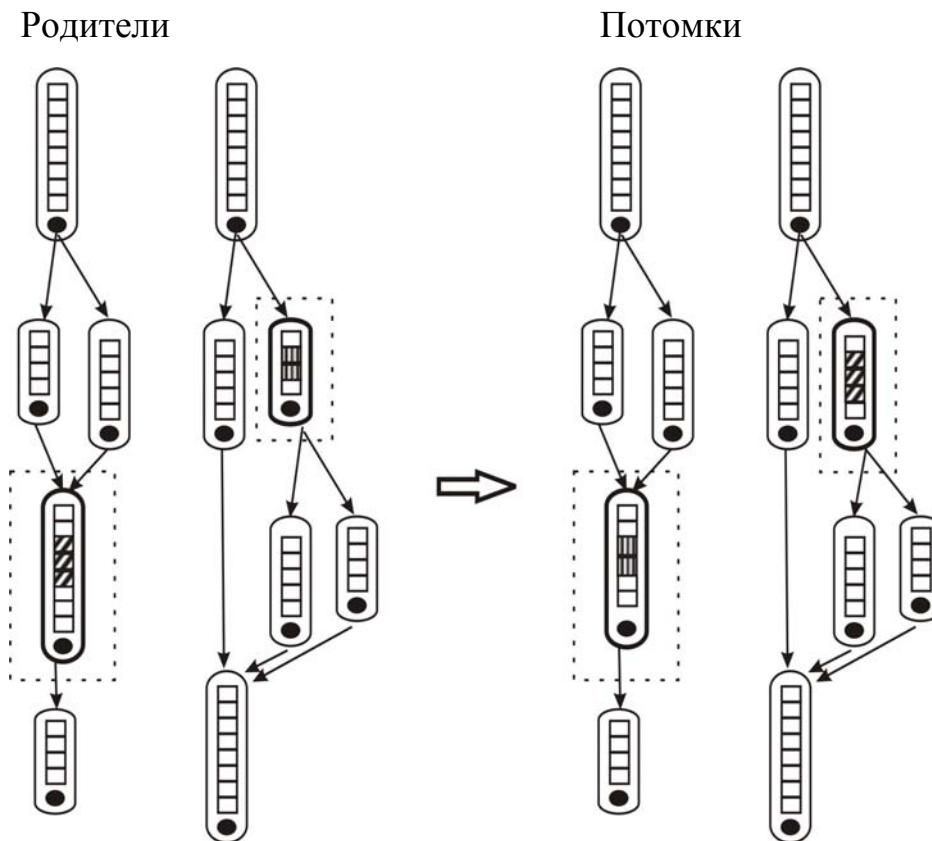


Рис.7.11. Линейный кроссинговер на графах.

Как правило, в процессе эволюции используются ОК обоих типов. В целом ОК выполняется следующим образом:

1. Выбор точки скрещивания P_1, P_2 в обоих родителях.
2. Выбор с заданной вероятностью типа ОК (для 1-го типа с вероятностью P_G , для 2-го с вероятностью $1 - P_G$).

Если выбран 1-й тип то переход на п.3, иначе на п.4.

3. Если размер потомка не превышает порог, то выполнить ОК 1-го типа, переход на п.5.

4. Если размер потомка не превышает порог, то выполнить ОК 2-го типа.

- 5.Конец.

7.5. Мутация в генетическом программировании

После выполнения кроссиговера с заданной малой вероятностью выполняется мутация для выбранной одной особи-программы.

7.5.1. Выполнение мутации на древовидных структурах

Для деревьев используются следующие операторы мутации (ОМ):

- а) узловая;
- б) усекающая;
- в) растущая.

Узловая мутация выполняется следующим образом:

- выбрать случайным образом узел, подлежащий мутации, определить его тип;
- случайно выбрать из соответствующего множества вариантов узлов отличный от рассматриваемого узел;
- поменять исходный узел на выбранный.

Усекающая мутация производится так:

- определяется или выбирается узел;
- случайным образом выбирается терминальный символ из заданного множества;
- обрезаются ветви узла мутации;
- вместо обрезанной ветви помещается выбранный терминальный символ.

На рис. 7.12 показан пример выполнения усекающей мутации.

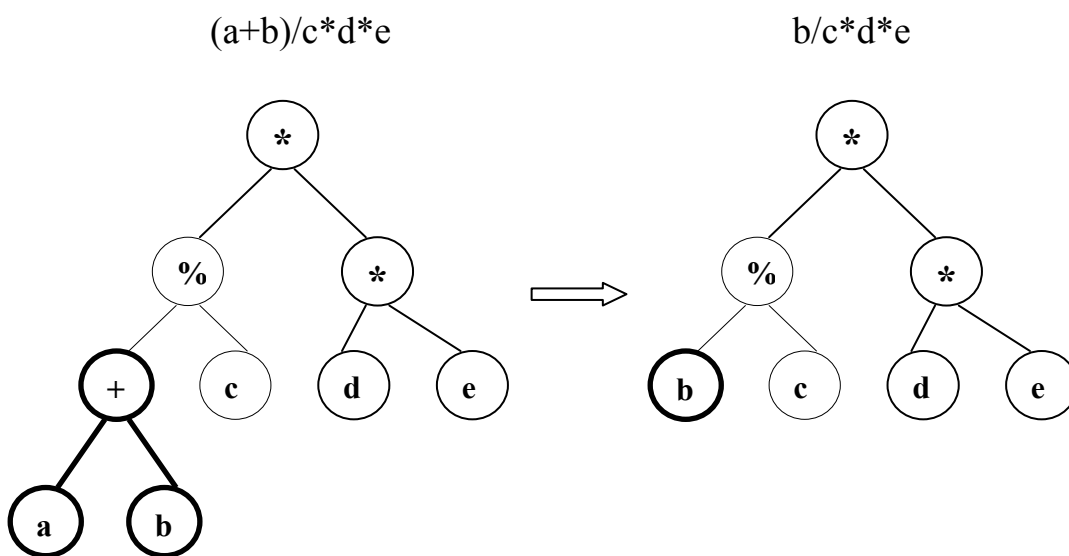


Рис.7.12. Усекающая мутация.

Растущая мутация выполняется следующим образом:

- определяется узел мутации;
- если узел нетерминальный то необходимо отсечь ветви исходящие из него, иначе выбрать другой узел.
- вычислить размер (сложность) остатка дерева;
- вместо отсеченного дерева вырастить случайным образом новое дерево так, чтобы размер нового построенного дерева не превышал заданный порог.

Это очень мощный оператор, который обладает большими возможностями. На рис.7.13 представлен пример выполнения этого оператора. В некоторых работах растущая мутация выполняется с использованием формальных грамматик. При этом нетерминальный символ заменяется в соответствии с правилами грамматики, а не случайным образом.

$$\frac{1}{2} * (d + e)$$

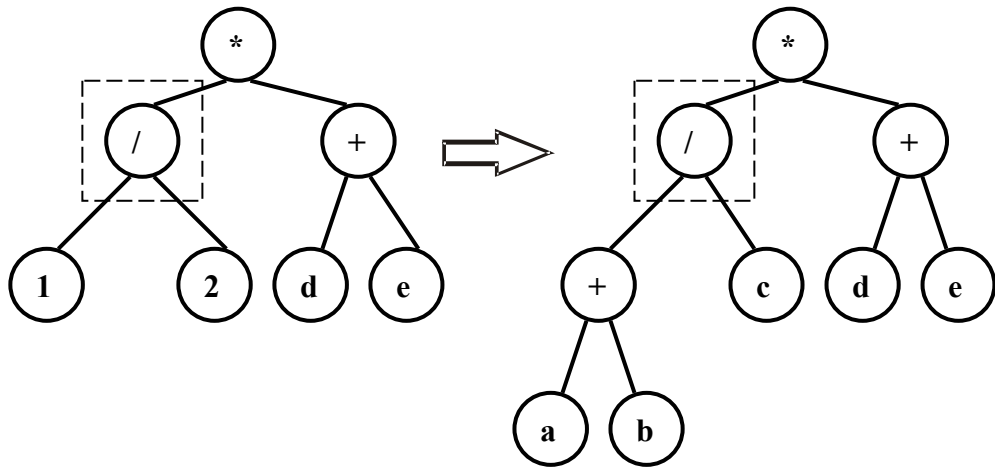


Рис.7.13. Растущая мутация.

В настоящее время для древовидных структур в ГП разработаны различные типы мутаций, которые производят различные изменения и представлены в табл.7.6 [27]. Для примера на рис.7.14 представлен результат выполнения мутации - перестановки (аргументов) для дерева, представляющего выражение $\frac{1}{2} * (d + e)$.

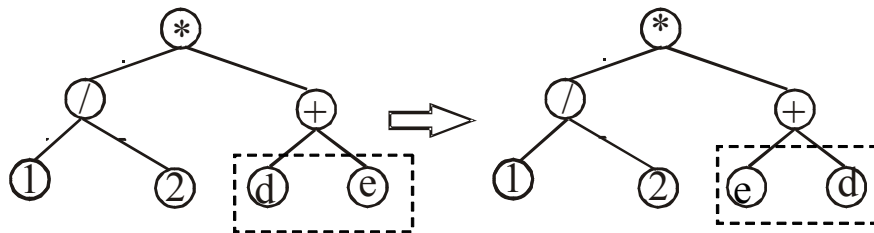


Рис.7.14 Пример мутации – перестановки аргументов

Таблица 7. 6.

Наименование	Описание производимых действий
Точечная мутация	Случайное изменение типа одного узла из того же класса
Перестановка	Перестановка аргументов одного узла
“Подъем”	Случайная генерация новой особи из поддерева
Растущая мутация	Замена терминального символа случайным поддеревом
Секущая мутация	Замена поддерева случайным терминальным символом
Мутация поддерева	Замена поддерева случайным поддеревом

7.5.2. Выполнение мутации на линейных структурах

Здесь мутация выполняется совершенно другим способом. Прежде всего, из линейного сегмента (тела программы) случайно выбирается оператор (команда) и в нем производятся изменения одного из следующих видов:

- 1) имя переменной (или регистра) заменяется на другое случайно выбранное из заданного множества;
- 2) оператор может быть также изменен случайным образом на некоторый другой из функционального множества;
- 3) может быть случайно изменено значение константы на некоторое другое значение из заданного диапазона.

Мутация констант выполняется путем стандартного отклонения от текущего значения $P = P + \delta \cdot h_m$. Вероятность мутации обычно убывает по мере удаления от начала процесса.

7.5.3. Выполнение мутации на графоподобных структурах

В этом случае оператор мутации случайным образом выбирает множество узлов и изменяет либо узел линейной программы, либо узел ветвления, либо число исходящих из условного оператора ветвей. То есть ОМ не строит новые линейные последовательности. Измененная ОМ программа помещается назад в популяцию.

7.6. Фитнесс-функция в генетическом программировании

В отличие от генетических алгоритмов, где часто при поиске экстремумов в качестве фитнесс-функции используется исходная целевая функция, в ГП фитнесс-функция обычно определяет меру близости между реальными y_i и требуемыми d_i выходными значениями (например, при использовании ГП в символьной регрессии). Поэтому в качестве фитнесс-функции часто используется абсолютная или квадратичная ошибка.

В этом случае фитнесс-функция использует обучающее множество данных, на котором выполняется обучение системы. С помощью фитнесс-функции в процессе обучения реализуется обратная связь, которая показывает насколько хорошо данная особь-программа реализует необходимую функцию на обучающем множестве. Для этого можно использовать различные виды фитнес-функций, некоторые из которых мы рассмотрим ниже.

«Непрерывной» (*continuous*) фитнес-функцией называют [27] функцию вычисления фитнес-значений, в которой малое улучшение в обучении программы вызывает малые улучшения измеряемых фитнес-значений и большие улучшения в обучении связаны соответственно с

большими изменениями (в сторону улучшения) фитнес-значений. В [27] отмечается, что такая «непрерывность» является важным свойством, так как позволяет ГП итеративно улучшать программы в процессе эволюции.

«Стандартизированной» [27] фитнес-функцией называют преобразованную фитнес-функцию, которая лучшим особям приписывает нулевое значение.

«Нормализованной» [27] фитнес-функцией называют преобразованную фитнес-функцию, которая для всех особей дает значения в интервале (0,1).

Рассмотрим следующий пример [27] с обучающей выборкой, представленной табл.7.7. Каждая строка таблицы определяет один элемент (х,у) обучающей выборки. Необходимо в процессе эволюции построить программу (или формулу в случае символьной регрессии), которая для каждого входного значения x вычисляет необходимое (в соответствии с табл.7.5) значение y (фактически нам необходимо реализовать функцию $f(x) = x^2 + x$).

Таблица 7.7.

№	Вход x	Выход d
1	1	2
2	2	6
3	4	20
4	7	56
5	9	90

Рассмотрим в качестве фитнес-функции ошибку в метрике абсолютных значений $f_a = \sum_{i=1}^n |y_i - d_i|$, где суммирование выполняется по обучающей выборке. Эта фитнес-функция соответствует первому определению «непрерывной» поскольку чем ближе значения y_i к d_i , тем

меньше значение фитнес-функции. Данная фитнес-функция является также стандартизованной, так как случае для идеального решения дает нулевое значение.

Часто в качестве фитнес-функции также используют квадратичную ошибку $f_s = \sum_{i=1}^n (y_i - d_i)^2$. Таблица 7.8 показывает различие для этих двух фитнес-функций в том случае, если на некотором (промежуточном) этапе в качестве особи оценивается (плохо обученная) программа, реализующая функцию $f(x) = x^2$.

Таблица 7.8.

№	Вход x	Выход d	Выход y	Ошибка f_a	Ошибка f_s
1	1	2	1	1	2
2	2	6	4	2	4
3	4	20	16	4	16
4	7	56	49	7	49
5	9	90	81	9	81
	Общая ошибка			23	151

Мы рассмотрели использование в качестве фитнес-функции ошибки в двух метриках, которые характерны для применения ГП в качестве символьной регрессии, что будет детальнее рассмотрено в разделе 7.8. Подобный подход применяется также во многих других задачах, где используются следующие фитнес-функции:

- 1) число правильно отображаемых пикселей в задачах обработки изображений;
- 2) число столкновений робота со стенами при обучении обхода препятствия;
- 3) число правильно классифицируемых примеров в задачах классификации;

- 4) ошибка между реальными и ожидаемыми значениями в задачах прогнозирования;
- 5) приз победителя в игровых задачах;
- 6) количество «пищи», найденной агентом, в многоагентных системах («искусственная жизнь» и т.п.)

Естественно разработано множество фитнес-функций других типов, их вид существенно зависит от исследуемой проблемы, а применение ГП не ограничивается задачами символьной регрессии. Кроме приведенных стандартных разработано много других методов определения фитнес-функции, в частности теоретико-игровых и основанных на идее коэволюции, где особи соревнуются между собой без явного вычисления значений. В некоторых случаях, кроме близости решений учитываются и другие критерии, например, длина или время выполнения программы. В этом случае говорят о многокритериальных фитнес-функциях.

7.7. Интроны

Программы, построенные с помощью методов ГП, имеют тенденцию к накоплению интронов – ненужных и непригодных участков кода.

Например:

$(NOT(NOT\ x))$,
 $(AND\ (ORXX))$,
 $(+\dots\ (-XX))$,
 $(+X0)$,
 $(*X1)$,
 $(* (DIV\ XX))$,
 $(MOVE_LEFT\ MOVE_RIGHT)$,
 $(IF\ (2=1)\ \dots)$,
 $A:=A$.

Таких фрагментов в программе возникает достаточно много (их количество может достигать 60%), и обнаружение и удаление интронов представляет серьезную проблему в ГП. Разработаны специальные методы для их устранения. Интересно отметить, что в живой природе интронов также достаточно много (в частности, на генном уровне существуют "лишние" участки ДНК).

7.8. Общий алгоритм генетического программирования

Таким образом, для решения задачи с помощью ГП необходимо выполнить описанные выше предварительные этапы:

- 1) Определить терминальное множество;
- 2) Определить функциональное множество;
- 3) Определить фитнес-функцию;
- 4) Определить значения параметров, такие как мощность популяции, максимальный размер особи, вероятности кроссинговера и мутации, способ отбора родителей, критерий окончания эволюции (например, максимальное число поколений) и т.п.

После этого можно разрабатывать непосредственно сам эволюционный алгоритм, реализующий ГП для конкретной задачи. Как и в случае ГА здесь также возможны различные подходы, которые рассмотрены в разделах 4-5.

Например, решение задачи на основе ГП можно представить следующей последовательностью действий.

- 1) установка параметров эволюции;
- 2) инициализация начальной популяции;
- 3) $t:=0$;
- 4) оценка особей, входящих в популяцию;
- 5) $t:=t+1$;

- 6) отбор родителей;
- 7) создание потомков выбранных пар родителей – выполнение оператора кроссинговера;
- 8) мутация новых особей;
- 9) расширение популяции новыми порожденными особями;
- 10) сокращение расширенной популяции до исходного размера;
- 11) если критерий останова алгоритма выполнен, то выбор лучшей особи в конечной популяции – результат работы алгоритма. Иначе переход на шаг 4.

Следует отметить, что в ГП достаточно часто применяется асинхронный ГА, рассмотренный в 4.6.

7.9. Символьная регрессия

Этот раздел является одним из важнейших приложений ГП. Данный термин подчеркивает то, что здесь объектом поиска является символьное описание модели, в отличие от множества коэффициентов в стандартных методах. Этот подход существенно отличается от других методов регрессии и использования нейросетей прямого распространения, где структура (и сложность) модели предполагается известной и фактически необходимо найти только ее коэффициенты. В случае символьной регрессии вид и сложность функции заранее неизвестны и могут изменяться в процессе поиска.

Задача регрессии может быть определена на основе множества значений входных независимых переменных x и зависимой выходной переменной y . Целью поиска является аппроксимация y с помощью переменных x и коэффициентов w следующим образом $y = f(x, w) + \varepsilon$, где ε представляет шум (ошибку).

В стандартных методах регрессии вид функции f предполагается известным, например, в линейной регрессии $f(x, w) = w_0 + w_1x_1 + \dots + w_nx_n$. Здесь коэффициенты w_i обычно находятся методом наименьших квадратов. В нелинейных методах, например, с использованием нейронных сетей прямого распространения функция имеет вид $f(x, w) = w_0 \bullet g(w_hx)$. Здесь коэффициенты w_0 и w_h представляют синаптические веса нейронной сети выходного и скрытых слоев соответственно.

Как уже отмечалось, символьная регрессия на основе ГП не использует некоторую заранее predetermined форму функции $f(x, w)$. Здесь функция $f(x, w)$ представляется древовидной структурой и строится эволюционным методом с использованием определенного функционального и терминального множеств. В качестве фитнес-функции обычно используется квадратичная ошибка, которая оценивает качество решения и обеспечивает обратную связь при поиске решения. Для определенности обозначим функции множества, зависящие от одной переменной через h_1, \dots, h_k и функции от двух переменных как g_1, \dots, g_l . В этой нотации функция $f(x, w)$ представляется в виде суперпозиции функций h_i , g_j , и например, может иметь следующий вид $f(x, w) = h_1(g_2(g_1(x_3, w_1), h_2(x_1)))$.

Заметим, что в символьной регрессии при поиске решения не используются численные методы, например, градиентные или стохастические.

Далее рассмотрим детально пример использования символьной регрессии из [27] для аппроксимации данных, представленных в табл.7.9. Необходимо найти функцию $f(x)$, которая аппроксимирует с заданной точностью эти «экспериментальные» данные.

Для решения задачи определим в соответствии с вышесказанным:

1. Терминальное множество: переменная x и константы в диапазоне $[-5, 5]$;

2. Функциональное множество: арифметические функции $+$, $-$, $*$, $\%$ (защищенное деление).
3. Фитнесс-функция – стандартизованная на основе корня квадратного средней квадратичной ошибки.
4. Параметры: мощность популяции, методы инициализации популяции и отбора родителей, значения вероятностей кроссинговера и мутации.

Таблица 7.9

№	Вход x	Выход y
1	0.000	0.000
2	0.100	0.005
3	0.200	0.020
4	0.300	0.045
5	0.400	0.080
6	0.500	0.125
7	0.600	0.180
8	0.700	0.245
9	0.800	0.320
10	0.900	0.405

Koza [5] ввел следующую удобную и «прозрачную» форму для перечисления параметров, которая представлена в табл.7.10.

Далее приведем некоторые полученные экспериментальные данные результатов эволюции для различных запусков программ из [27]. В начальной популяции после инициализации лучшая особь представлена деревом рис.7.15, которая реализует (не минимальным образом !) функцию $f_0(x) = \frac{x}{3}$.

В последующих рисунках рис.7.16 –7.20 представлены лучшие особи последующих поколений. Здесь в первом поколении лучшая особь реализует $f_1(x) = \frac{x}{6-3x}$ и соответствующее дерево рис.7.16 сильно избыточно.

Аналогично во втором поколении лучшая особь реализует функцию $f_2(x) = \frac{x}{x(x-4)-1+\frac{4}{x}-\frac{9(x+1)+x}{6-3x}}$ и дерево тоже сильно избыточно. Наконец в

третьем поколении получена лучшая особь $f_3(x) = \frac{x^2}{2}$, которая дает оптимальное решение в простейшей форме.

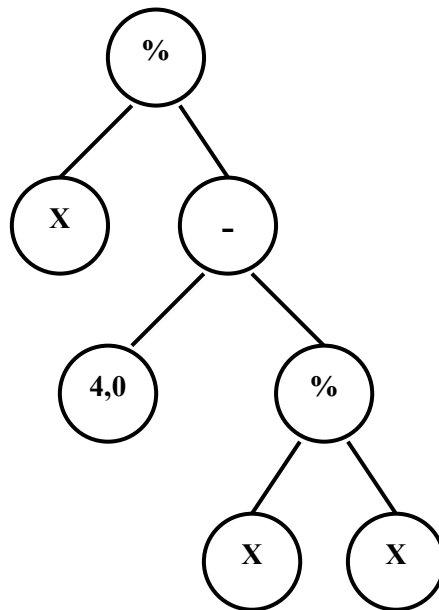


Рис.7.15. Лучшая особь в поколении 0.

Таблица 7.10

Параметры	Значения
Цель:	Эволюция функции, аппроксимирующей данные Табл.7.6
Терминальное множество	Переменная x , Целые от -5 до $+5$
Функциональное множество	ADD, SUB, MUL, DIV
Мощность популяции:	600
Вероятность кроссинговера:	0.90
Вероятность мутации:	0.05
Отбор родителей:	турнирный с мощностью тура 4
Максимальное число поколений:	100
Максимальная глубина после кроссинговера:	200
Максимальная глубина мутации:	4
Метод инициализации:	Растущая

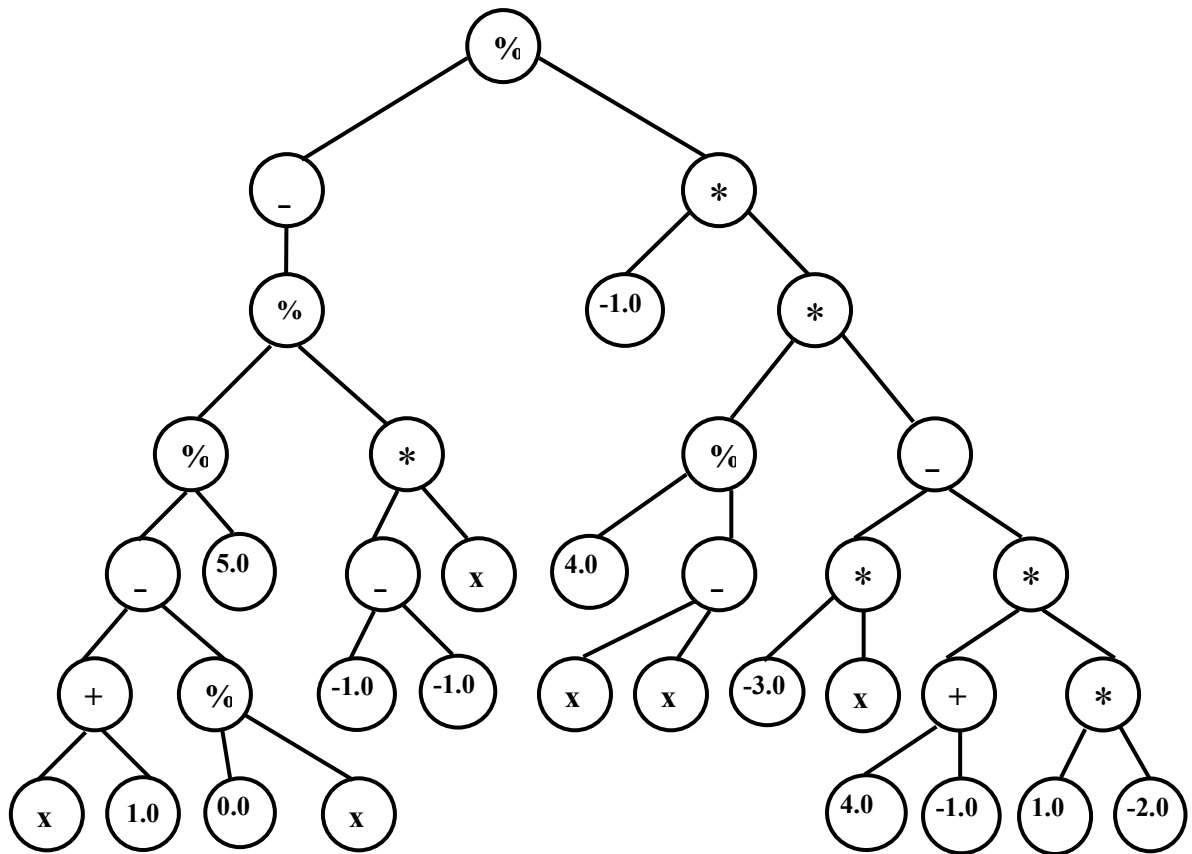


Рис.7.16. Лучшая особь поколения 1.

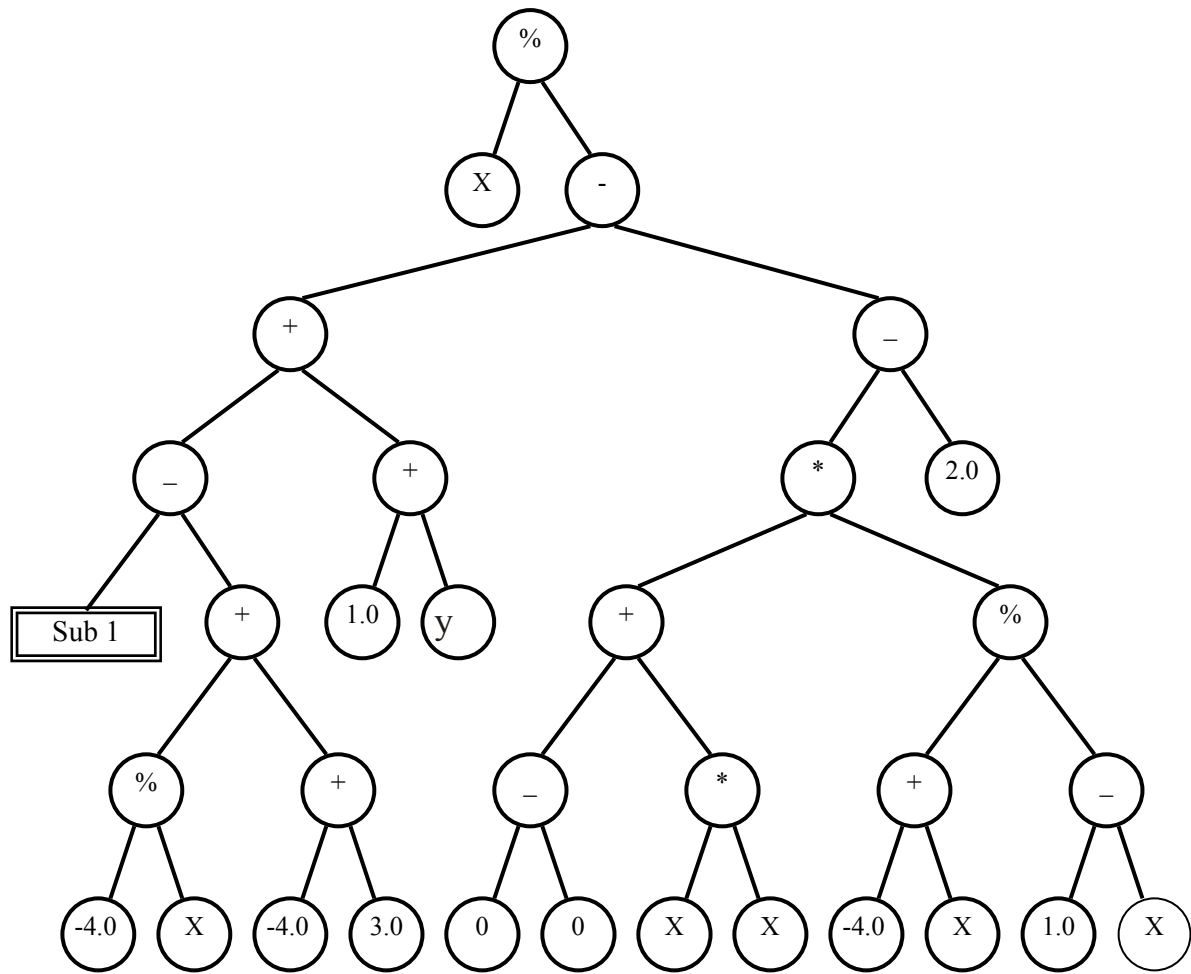


Рис.7.17. Лучшая особь поколения 2.

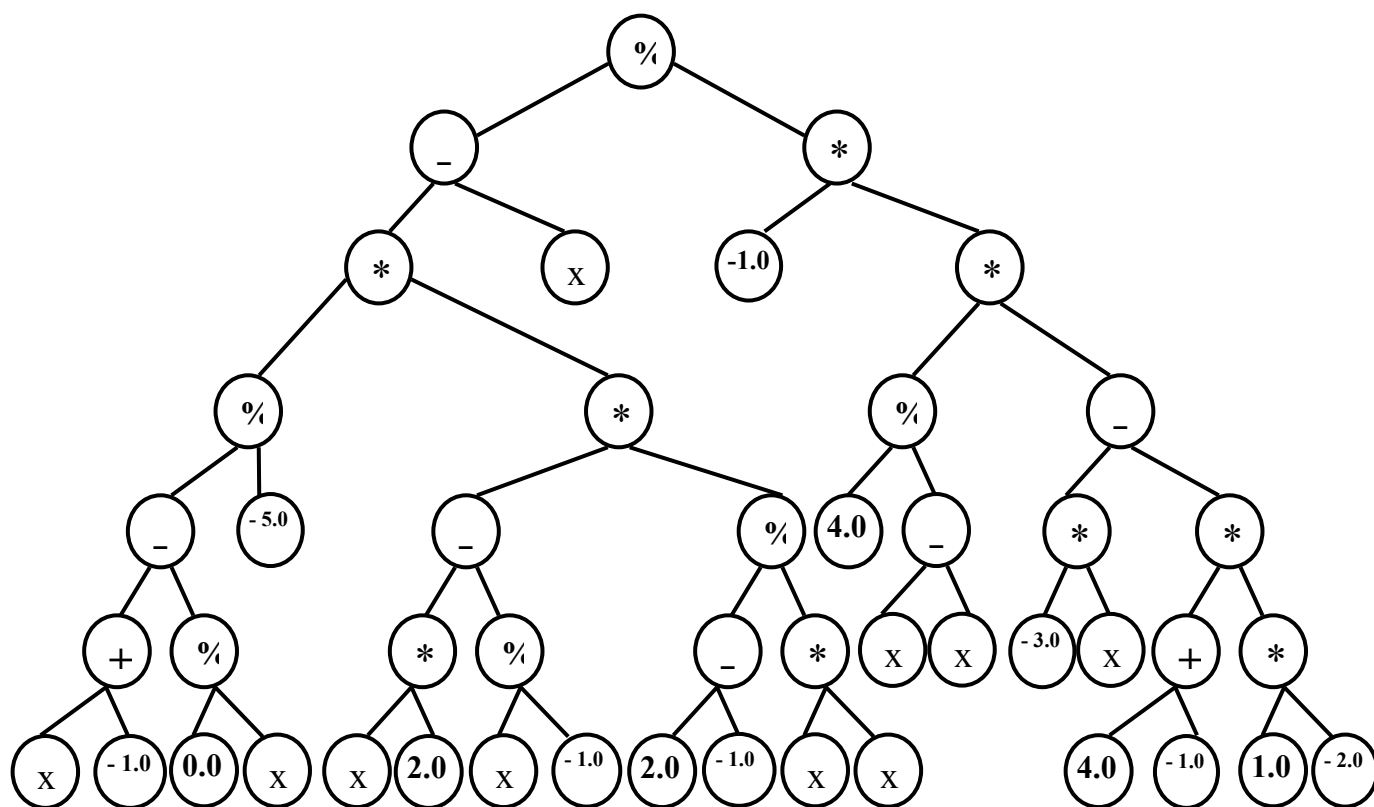


Рис.7.18. Лучшая особь поколения 3.

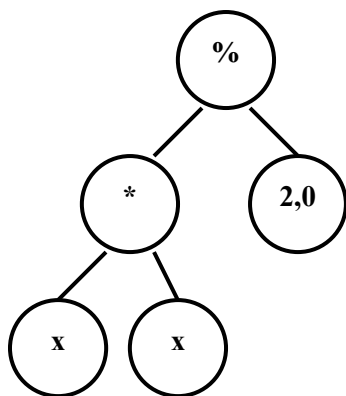


Рис.7.19. Лучшая особь поколения 4.

В табл.7.11 для сравнения представлены значения функций лучших особей первых поколений (0-3).

Отметим, что была выполнена еще одна итерация (4-е поколение). На рис.7.19 представлена лучшая особь четвертого поколения, которая также реализует функцию $f_4(x) = \frac{x^2}{2}$, но избыточность соответствующего дерева выросла. Конечно, для данного примера можно было использовать и классические методы регрессии, но он носит чисто иллюстративный характер. Во многих работах [5,27] приведены результаты для более сложных зависимостей, но они требуют для представления результатов большого объема.

Таблица 7.11

№	y	f ₀	f ₁	F ₂	f ₃
1	0.000000	0.000000	0.000000	0.000000	0.000000
2	0.005000	0.033333	0.017544	0.002375	0.005000
3	0.020000	0.066667	0.037037	0.009863	0.020000
4	0.045000	0.100000	0.058824	0.023416	0.045000
5	0.080000	0.133333	0.083333	0.044664	0.080000
6	0.125000	0.166667	0.111111	0.076207	0.125000
7	0.180000	0.200000	0.142857	0.1222140	0.180000
8	0.245000	0.233333	0.179487	0.188952	0.245000
9	0.320000	0.266667	0.222222	0.287024	0.320000
10	0.405000	0.300000	0.272727	0.432966	0.405000

Следует отметить, что мощность популяций используемых в ГП на порядок выше, чем в классических ГА, а число поколений, как правило, меньше. Применение ГП в качестве символьной регрессии является важным для многих технических задач (например, в гидро- и аэро-динамике), но далеко не единственным.

7.10 . Диагностическое дерево

В настоящее время ГП достаточно широко используется в задачах классификации и прогнозирования и составляет конкуренцию искусственным нейронным сетям [39]. Одним из возможных вариантов применения ГП основано на построении «диагностического дерева»[46]. Ниже рассмотрено применение ГП для построения «диагностического дерева» на примере решения задачи диагностирования синдрома внезапной смерти грудных детей (СВСГД) [47,48].

Здесь аппарат ГП позволяет на основе обучающей выборки построить с помощью эволюционного алгоритма программу, которая на ранней стадии позволяет определить степень риска (высокую или низкую) данного заболевания. В данной задаче в качестве обучающего множества используются реальные данные обследования 240 пациентов, (120 детей, которые умерли за период 1990-1999г. в Донецкой области от СВСГД, и контрольная группа из 120 живых детей на первом году жизни). Данные составляют информацию общего характера и образа жизни беременных, а так же перенесенные заболевания и результаты некоторых анализов.

Собрана максимально полная информация о возможных параметрах, которые в той или иной степени могут влиять на СВСГД поскольку в настоящее время это заболевание плохо изучено. К возможным факторам риска выделили следующую информацию:

– информация о матери: место жительства – город или село; вредные условия труда; образование; состоит ли в браке; бытовые условия и количество м² на человека; рост и вес; возраст на момент первой беременности; чем закончилась первая беременность; возраст на момент первых месячных; регулярность, болезненность, длительность и интервал месячных; возраст на момент беременности; номер беременности; роды по счету; чем закончились предыдущая беременность; количество аборт, самоабортов, мертворождений; плодность текущей беременности; курение, алкоголь, наркотики в течении беременности; перенесенные заболевания; способы контрацепции; TORCH – инфекции; патология беременности; гинекологические заболевания; группа крови и резус фактор.

– информация об отце: возраст; курение; алкоголь; наркотики.

– информация о ребенке: пол; кормили ли грудью, искусственное питание или смешанное; вес; рост; количество баллов по шкале Апгар; срок гестации; врожденные пороки; сразу после родов находился: в палате интенсивной терапии, в палате, с мамой.

Полученные данные, на основании которых, система будет давать ответ, имеют самый разнообразный вид. Поэтому данные должны быть предварительно обработаны, основное назначение предобработки преобразовать входное обучающее множество в булевы значения соответствующих (булевых) переменных.

Предобработка входных данных

Входное обучающее множество должно быть представлено в виде булевых значений. Для этого исходные данные были преобразованы следующим образом:

- место жительства (город – 1, село – 0);
- возраст матери на момент родов (полных лет) <17;
- возраст матери на момент родов (полных лет) <25;

- возраст матери на момент родов (полных лет) < 30 ;
- возраст матери на момент родов (полных лет) > 31 ;
- место работы матери, профвредность (да – 0 , нет – 1);
- образование матери (высшее);
- образование матери (среднее);
- образование матери (средне специальное);
- образование матери (начальное);
- бытовые условия семьи (общежитие);
- бытовые условия семьи (перенаселенная квартира);
- бытовые условия семьи (отдельная квартира);
- бытовые условия семьи (дом);
- экстрагенитальная патология матери (пищеварения);
- экстрагенитальная патология матери (дыхания);
- экстрагенитальная патология матери (зрение);
- экстрагенитальная патология матери (эндокринная система);
- экстрагенитальная патология матери (сердечно сосудистая система);
- экстрагенитальная патология матери (другие);
- гинекологические заболевания матери (воспалительные заболевания);
- гинекологические заболевания матери (нарушения менструального цикла);
- гинекологические заболевания матери (эндометриоз);
- гинекологические заболевания матери (доброкачественные опухоли);
- гинекологические заболевания матери (эрозия шейки матки)
- гинекологические заболевания матери (другие);
- возраст матери на момент начала месячных (полных лет) < 12 ;
- возраст матери на момент начала месячных (полных лет) < 14 ;

- возраст матери на момент начала месячных (полных лет) > 15 ;
- длительность месячных (кол-во дней) < 3 ;
- длительность месячных (кол-во дней) < 5.5 ;
- длительность месячных (кол-во дней) > 6 ;
- цикл месячных (кол-во дней) < 21 ;
- цикл месячных (кол-во дней) < 34 ;
- цикл месячных (кол-во дней) > 35 ;
- регулярность месячных (да – 1 , нет – 0);
- болезненность месячных (да – 1 , нет – 0);
- обильные месячные ;
- нормальные месячные;
- скудные месячные;
- пол ребенка (м – 1, д – 0);
- характер кормления ребенка (грудным молоком);
- характер кормления ребенка (смесью);
- характер кормления ребенка (грудным молоком + смесью);
- возраст отца на момент родов (полных лет) < 20 ;
- возраст отца на момент родов (полных лет) < 25 ;
- возраст отца на момент родов (полных лет) < 30 ;
- возраст отца на момент родов (полных лет) > 31 ;
- брак зарегистрирован;
- брак не зарегистрирован;
- мать-одиночка;
- вдова;
- разведена;
- номер данной беременности =1;
- номер данной беременности =2;
- номер данной беременности > 3 ;

- одноплодная данная беременность;
- многоплодная данная беременность;
- возраст матери на момент первой беременности (полных лет) <17 ;
- возраст матери на момент первой беременности (полных лет) <25 ;
- возраст матери на момент первой беременности (полных лет) >26 ;
- исход первой беременности аборт;
- исход первой беременности роды;
- исход первой беременности самоаборт;
- роды по счету = 1;
- роды по счету = 2;
- роды по счету >3 ;
- путь плодоразрешения – роды;
- путь плодоразрешения – кесарево сечение;
- предыдущая беременность закончилась абортом медицинским в ранний срок;
- предыдущая беременность закончилась абортом медицинским в поздний срок;
- предыдущая беременность закончилась самопроизвольным абортом;
- предыдущая беременность закончилась мертворождением;
- предыдущая беременность закончилась кесаревым сечением;
- предыдущая беременность закончилась замиранием плода с родоразрушением;
- предыдущая беременность закончилась самостоятельными родами;
- предыдущая беременность закончилась самостоятельными родами;
- многоплодная беременность;
- интервал между данными и предшествовавшими родами (месяцев) <14 ;
- интервал между данными и предшествовавшими родами (месяцев) $<$

49;

- интервал между данными и предшествовавшими родами (месяцев) >50;
- контрацепция – спираль;
- контрацепция - таблетированные контрацептивы;
- контрацепция - другие методы;
- срок постановки матери на учет в женской консультации (недель) < 12;
- срок постановки матери на учет в женской консультации (недель) < 20;
- срок постановки матери на учет в женской консультации (недель) >21;
- перенесенные заболевания матери во время беременности – вирусные;
- перенесенные заболевания матери во время беременности – соматические;
- мать во время данной беременности курила – мало;
- мать во время данной беременности курила – средне;
- мать во время данной беременности курила – много;
- мать во время данной беременности употребляла алкоголь – мало;
- мать во время данной беременности употребляла алкоголь – средне;
- мать во время данной беременности употребляла алкоголь - много;
- мать во время данной беременности употребляла наркотики – мало;
- мать во время данной беременности употребляла наркотики – средне;
- мать во время данной беременности употребляла наркотики – много;
- отец курил – мало;
- отец курил – средне;

- отец курил – много;
- отец употреблял алкоголь – мало;
- отец употреблял алкоголь – средне;
- отец употреблял алкоголь – много;
- отец употреблял наркотики – мало;
- отец употреблял наркотики – средне;
- отец употреблял наркотики – много;
- рост матери < 150 ;
- рост матери < 175 ;
- рост матери > 176 ;
- вес матери < 45 ;
- вес матери < 65 ;
- вес матери < 85 ;
- вес матери > 86 ;
- конституционные особенности матери ожирение -1;
- конституционные особенности матери ожирение -2;
- конституционные особенности матери ожирение - 3;
- анемия легкой степени;
- анемия средней степени;
- анемия тяжелой степени;
- гипертония;
- пиелонефрит;
- угроза самопроизвольного аборта;
- угроза преждевременных родов;
- sensibilization по АВ0;
- sensibilization по Rh-
- ЗВУР;
- ХФПН;

- рвота легкой степени;
- рвота средней степени;
- рвота тяжелой степени;
- преклампсия легкой степени;
- преклампсия средней степени;
- преклампсия тяжелой степени;
- отек;
- TORCH инфекции;
- прибавка в весе;
- АД верх.гран < 110 ;
- АД верх.гран < 150 ;
- АД верх.гран > 150 ;
- АД ниж.гран < 70 ;
- АД ниж.гран < 100 ;
- АД ниж.гран > 100 ;
- срок гестации (недель) < 37 ;
- срок гестации (недель) < 41 ;
- срок гестации (недель) > 41 ;
- масса ребенка при рождении (граммы) < 3000 ;
- масса ребенка при рождении (граммы) < 4000 ;
- масса ребенка при рождении (граммы) > 4100 ;
- рост ребенка при рождении (см) < 50 ;
- рост ребенка при рождении (см) < 56 ;
- рост ребенка при рождении (см) > 57 ;
- оценка по шкале Апгар (балы) < 3 ;
- оценка по шкале Апгар (балы) < 5 ;
- оценка по шкале Апгар (балы) < 7 ;
- оценка по шкале Апгар (балы) < 10 ;

- перинатальная патология;
- продолжительность второго периода родов (мин) < 10 ;
- продолжительность второго периода родов (мин) > 11 ;
- срок первого прикладывания ребенка к груди на 1 день;
- срок первого прикладывания ребенка к груди на 2 день и более;
- наличие пороков развития у ребенка;
- после родов ребенок находился - с матерью;
- после родов ребенок находился - в палате новорожденных;
- после родов ребенок находился - в отделении интенсивной терапии;

Наличие каждого фактора принято за единицу, отсутствие за ноль.

Далее в соответствии с приведенным выше алгоритмом определяются терминальное и функциональное множества, фитнес-функция, параметры эволюционного алгоритма и т.п.

Терминальное множество

Терминальное множество в данном случае составляют перечисленные выше параметры, которые после предобработки представляют собой булевы переменные.

Функциональное множество

Функциональное множество состоит из логических операций: AND, OR, NOT. Так как первые две операции могут иметь два и более входов и один выход, а последняя операция всегда имеет один вход и один выход, то для удобства программной реализации операция NOT заменена на AND-NOT и OR-NOT. Такая замена выполнена с целью унифицирования количества входов для всех операций. Кроме того, это позволяет избавиться от так называемых интронов (противоречивых программных кодов), в данном случае «двойное отрицание». Таким образом, функциональное множество состоит из 4 логических операций AND, OR, AND-NOT и OR-NOT.

Фитнесс-функция

В качестве фитнес-функции рассматривается доля пациентов с правильно поставленным диагнозом. Переменная диагноза принимает булевы значения 0 или 1. Единица соответствует положительному диагнозу (высокой степени риска СВСГР) и ноль отрицательному (низкой степени риска СВСГР). Значение фитнес-функции для особей с правильным диагнозом принимает значение 1, а для особей с неправильным диагнозом принимает значение 0.

Реализация метода

В контексте нашей задачи использовалось ГП для получения дерева, которое позволяет распознавать высокую степень риска СВСГР. Для получения такого дерева необходимо выполнить следующие этапы:

1. Установка параметров ГП. Возможные варианты приведены в таблице 7.12.
2. Генерация начальной популяции. Популяция представляет собой набор хромосом. Каждая хромосома соответствует определенному дереву, представляющее собой решение. Дерево (хромосома), на начальном этапе генерируется случайным образом и состоит из функциональных и терминальных узлов (множество которых описано выше).
3. По значению фитнес-функции оцениваются особи входящие в популяцию.
4. Применение генетических операторов.
5. Проверка критерия останова. При его выполнении переход на шаг 6, иначе шаг 3.

6. Лучшее дерево (максимум правильных результатов распознавания), полученное на любом этапе работы запоминается и является результатом работы ГП, а само дерево считается решением поставленной задачи.

Обобщенный алгоритм получения дерева с помощью ГП представлен на рисунке 7.20.

Параметры эволюционного алгоритма

В процессе поиска решений возможно использование параметров генетического алгоритма, приведенных в таблице 7.12.

Таблица 7.12.

Параметр	Допустимые значения
Мощность популяции	Задается
Максимальная глубина особи в начальной популяции	Задается
Максимальная глубина особи в эволюционирующей популяции	Задается
Метод генерации начальной популяции	<ul style="list-style-type: none"> – растущая $p_g = 100\%$ – полная $p_c = 100\%$ – смешенная $p_g + p_c = 100\%$
Вероятность функционального узла	50%
Вероятность терминального узла	50%
Вероятность кроссинговера	Задается
Вероятность мутации	Задается
Отбор родителей	<ul style="list-style-type: none"> – турнир – рулетка

Апробация метода

Для реализации поставленной задачи написана программа в среде C++ Builder 6, которая выполняет рассмотренный алгоритм. В таблице 7.13 приведены параметры, при использовании которых достигнут лучший результат.

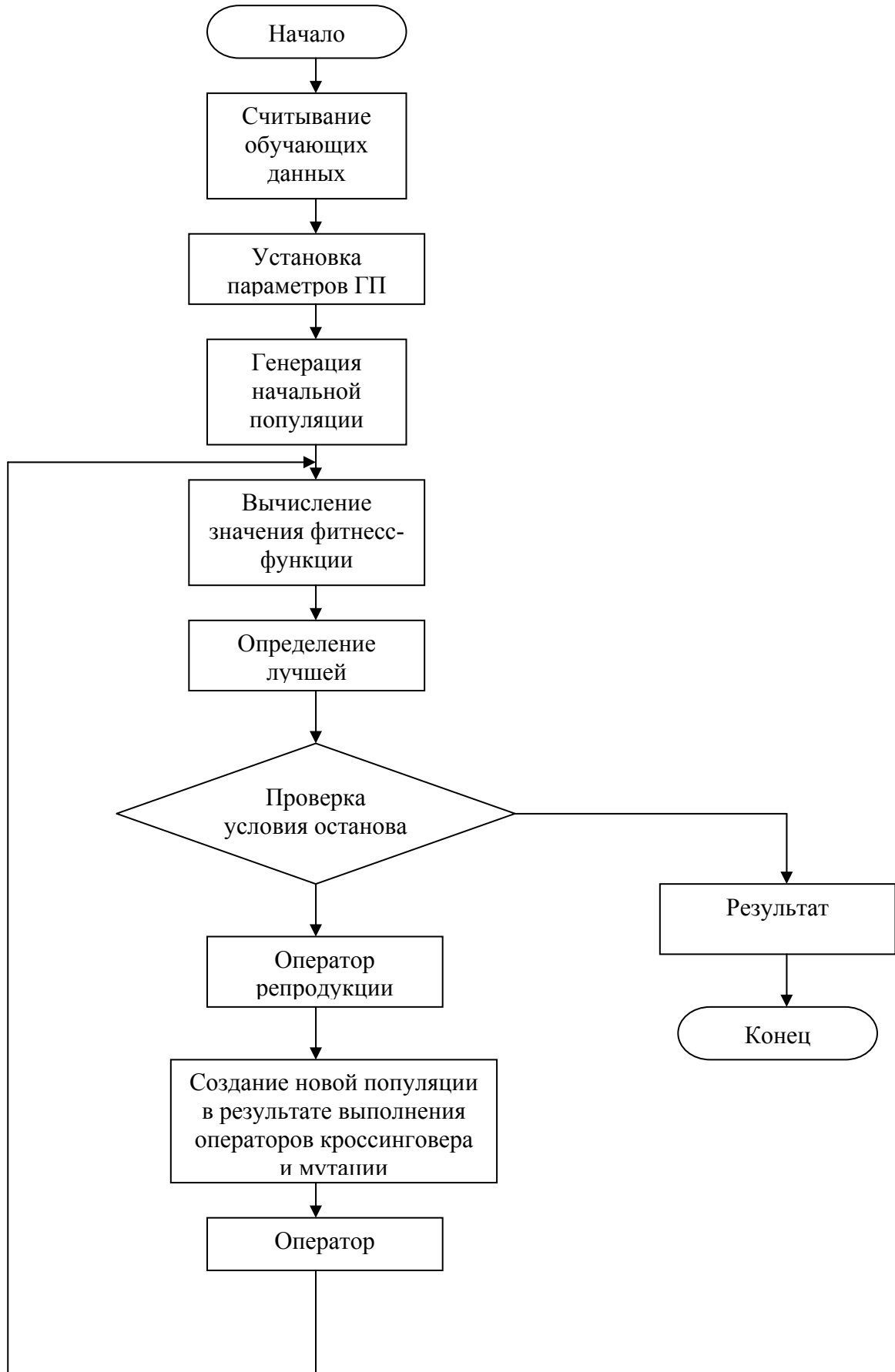


Рисунок 7.20. Обобщенный алгоритм получения дерева с помощью ГП.

Таблица 7.13

Параметр	Значения
Мощность популяции	200
Максимальная глубина особи	10
Метод генерации начальной популяции	– растущая $p_g = 50\%$ – полная $p_c = 50\%$
Вероятность функционального узла	50%
Вероятность терминального узла	50%
Вероятность кроссинговера	99%
Вероятность мутации	5%
Отбор родителей	– рулетка

В качестве критерия останова можно выбирать максимальное число итераций или определенное число повторений лучшего результата.

При тестировании на реальных медицинских данных получено 95,71% правильно распознанных диагнозов. Один из лучших вариантов решения в виде построенного дерева для определения высокой степени риска СВСГД, представлен на рисунке 7.21. Здесь левая ветвь А дерева перенесена на следующую страницу.

Разработанный аппарат ГП создан и протестирован на примере прогнозирования СВСГД, но может быть использован и при решении других задач медицинской диагностики и прогнозирования.

Таким образом, аппарат ГП может быть с успехом использован при решении задач классификации для решения задач медицинской диагностики и прогнозирования. По некоторым данным [39] он не уступает по качеству полученных результатов многослойным нейронным сетям прямого распространения.

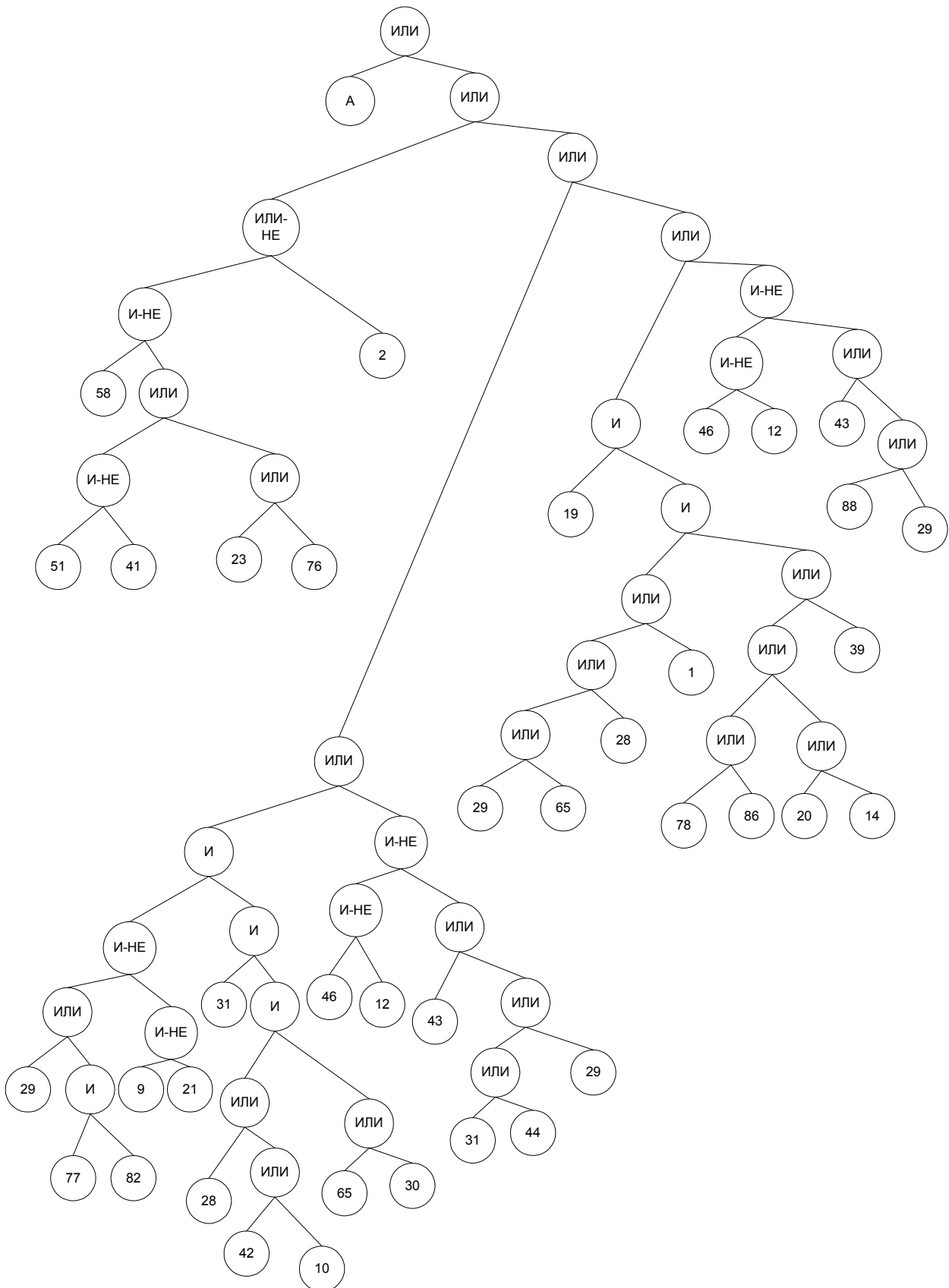


Рисунок 7.21 а) Дерево для прогнозирования высокой степени риска СВСГР

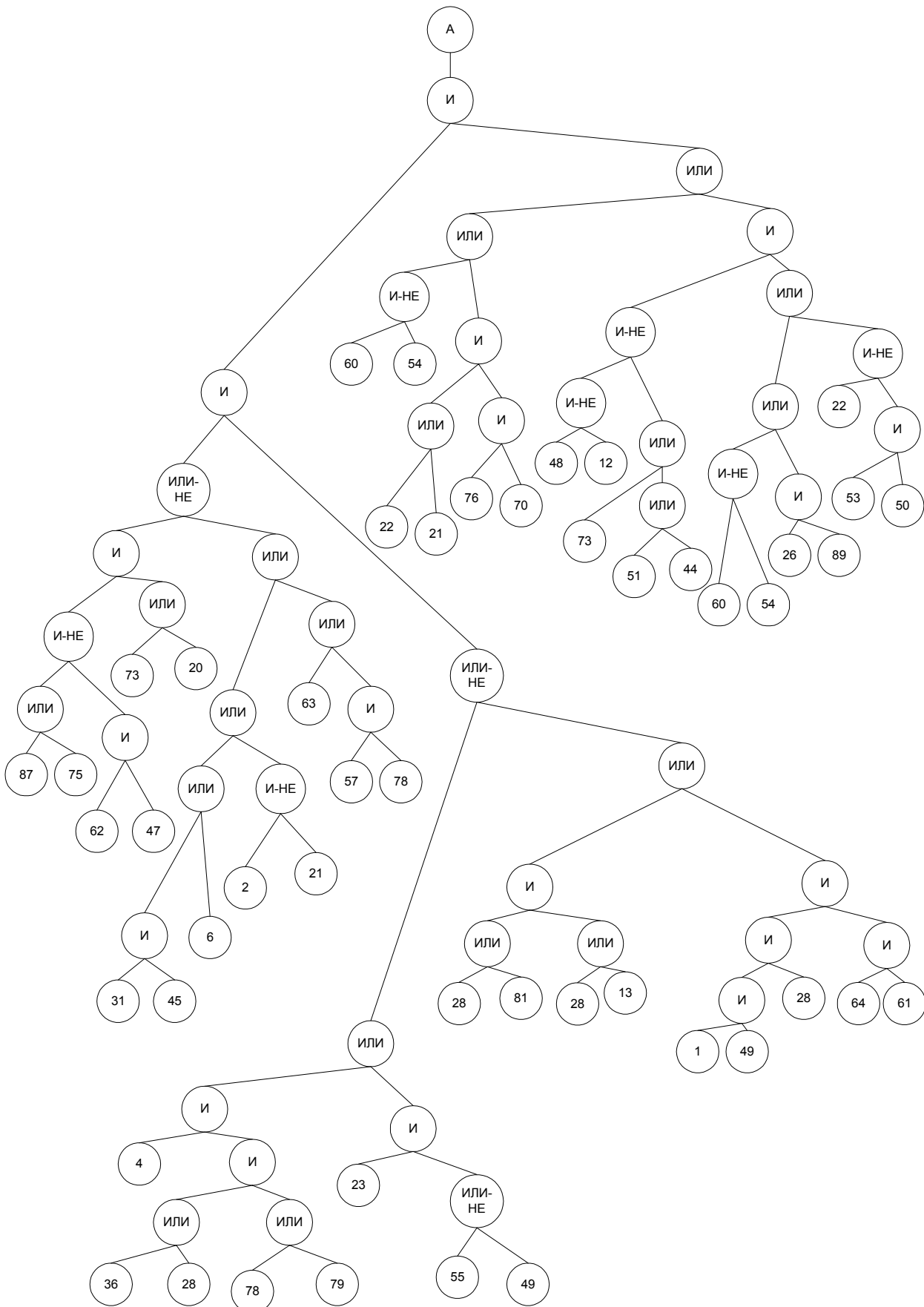


Рисунок 7.21 б) Дерево для прогнозирования высокой степени риска СВСГР

7.11 Модульное построение программ в генетическом программировании

Эволюционный подход в представленном выше виде порождает слабо структурированные программы, что подтверждают приведенные примеры. Напротив программист (человек) при проектировании программ, как правило, применяет модульный принцип построения, где решение разбивается на отдельные небольшие (часто используемые) участки кода, которые затем используются в разработке программы. Для этого широко используется аппарат функций и подпрограмм, что позволяет существенно упростить как процесс проектирования, так и сами разрабатываемые программы. Это подход является типичным примером применения стратегии «разделяй и властвуй», которая широко используется при проектировании программ. Одним из самых перспективных направлений повышения эффективности ГП является использование модульного принципа, в котором идентифицируются некоторые функциональные единицы программы, которые оформляются по определенным стандартам и сохраняются для дальнейшего использования.

Модулем называют последовательность (соседних) операторов программы, ограниченную некоторыми окаймляющими элементами, который имеет общий идентификатор. Модуль должен обладать, по крайней мере, следующими свойствами:

- 1) логическая замкнутость;
- 2) реализация по принципу «черного ящика»;
- 3) должен иметь интерфейс для взаимодействия с другими модулями (множество операций и типов данных).

Модульный принцип является средством инкапсуляции блоков кода. Такие блоки оформляются в виде подпрограмм, которые затем могут

многократно использоваться в главной программе или включаться в другие подпрограммы. Это позволяет существенно уменьшить сложность программы за счет оформления часто используемых идентичных участков кода в виде подпрограмм. Отметим, что меньшая сложность (длина) программ способствуют их «выживанию» в процессе эволюционного поиска. В ГП используется различная техника для реализации модульного принципа. Самыми известными методами являются автоматически определяемые функции (АОФ) (automatically defined functions –ADF), которые введены в [5] и детально исследованы во многих работах [5,27].

ADF построены по аналогии с функциями языка LISP. Поэтому программа, содержащая ADF, также представляется древовидной структурой, но при этом дерево состоит из двух частей (поддеревьев):

- 1) главной программы, которая, прежде всего оценивается с помощью фитнес-функции;
- 2) одно или несколько определений функций.

Пример такой структуры приведен на рис.7.22.

Эти два поддерева соответствуют структуре программы на языке программирования высокого уровня, подобном C++, Паскаль и т.п., где главная программа (тело) сопровождается «объявлениями», где описаны соответствующие функции. Отметим, что обе указанные компоненты должны участвовать в эволюции, которая в этом случае производит программу, построенную в соответствии с модульным принципом.

На рис.7.22 корень дерева “**программа**” соответствует программе в целом и объединяет ее различные части. Узел “**определение**” является корневым для поддерева, содержащего описание функции ADF0. В случае необходимости использования нескольких функций каждой из них необходимо выделить соответствующее поддерево ADF для описания функции.

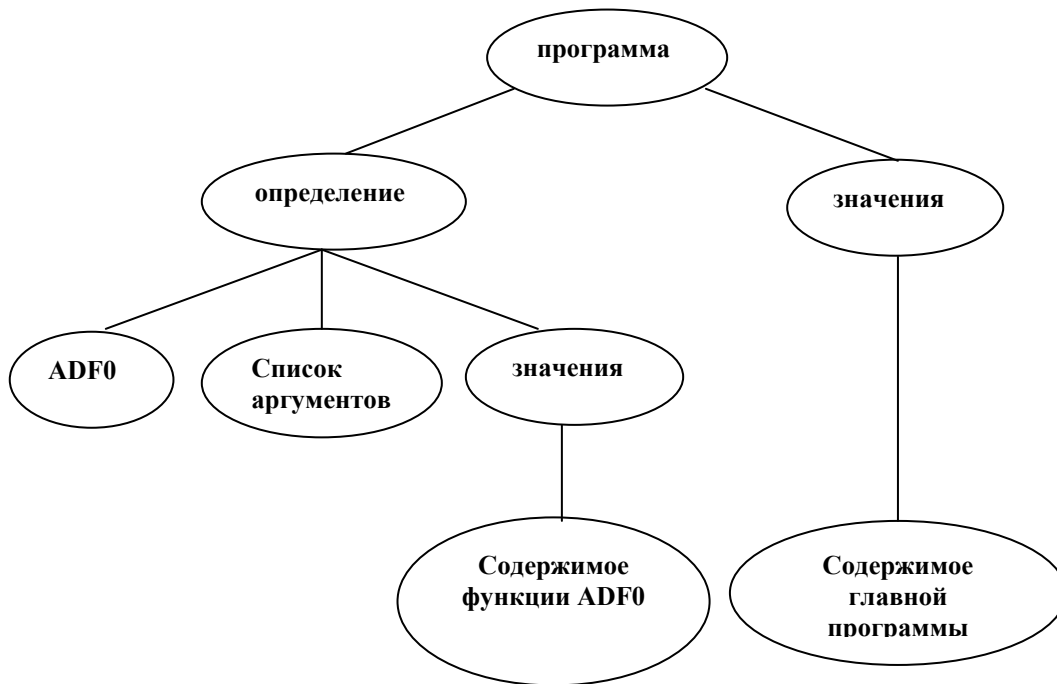


Рис.7.22 Дерево, содержащее ADF

Левая ветвь, исходящая из узла “**определение**”, содержит имя функции (ADF0), которое включается в функциональное множество главной программы и используется для вызова соответствующей функции. Справа от узла имени расположен узел, представляющий список аргументов данной функции. Этот список определяет имена входных переменных аргументов функции. Эти имена входят в терминальное множество тела соответствующей функции. Этот подход аналогичен определению функции в C++ или Паскале, где необходимо дать имя функции и определить входные переменные вместе с описанием тела функции. Следует отметить, что использование рекурсии, как правило, не допускается вследствие значительных трудностей реализации.

Очевидно, что данный подход требует применения специального, синтаксически ограниченного кроссинговера. Использование описанных выше видов кроссинговера для древовидных структур в этом случае невозможно. Отметим, что, различные поддеревья (главная программа и

описание функций) имеют свои (существенно отличающиеся) функциональные и терминальные множества. Очевидно, что при кроссинговере нельзя производить обмен некоторыми частями между различными типами поддеревьев. Поэтому, прежде всего, при выполнении кроссинговера производится выбор поддерева в одном из поддеревьев первого родителя (главной программе или функции). Соответственно точка кроссинговера (или поддерево) во втором родителе должна быть выбрана в поддереве того же типа, только в этом случае возможен обмен фрагментами деревьев.

До запуска программы поиска с использованием данного подхода необходимо определить число ADF функций и допустимое число аргументов для каждой функции. То есть мы, должны в каком то смысле задать структуру дерева. Естественно при инициализации мы должны придерживаться этой структуры. При этом различные поддеревья (главная программа и функции) генерируются независимо (случайными методами).

При использовании данного подхода необходимо дополнительно выполнить следующие действия:

- 1) выбрать число функций ADF;
- 2) фиксировать число аргументов для каждой ADF;
- 3) определить возможные связи между разными ADF, если используется больше одной функции;
- 4) определить терминальные функциональные множества для главной программы и ADF;
- 5) определить фитнес- функцию для оценки особей;
- 6) определить параметры эволюционного алгоритма.

Основным недостатком данного подхода является необходимость определения заранее структуры программы, которая включает число ADF функций, число аргументов и т.п.

Кроме ADF в ГП применяются и другие методы модульного построения программ [27]. К ним относится использование автоматически определяемых макросов (automatically defined macros), автоматическое определение циклов `do while`, `until` и рекурсии, применение сложных и абстрактных типов данных и т.д. С одной стороны они позволяют получать более компактные программы, но с другой их применение существенно осложняет сам процесс их построения.

7.12 Контрольные вопросы к разделу 7

1. Чем отличаются терминальное и функциональное множества?
2. Какие структуры используются для представления программ в ГП?
3. Опишите древовидное представление.
4. Какой тип памяти используется в древовидном представлении?
5. Опишите линейное представление программы.
6. Опишите представление программы в виде графа.
7. Какие два метода используются в инициализации древовидных структур?
8. Как производится инициализация линейных структур?
9. Какие виды кроссинговера вы знаете для древовидных структур?
10. Как выполняется кроссинговер на линейных структурах?
11. Какие виды кроссинговера вы знаете для графоподобных структур?
12. Какие виды мутации вы знаете для древовидных структур?
13. Как производится мутация на линейных структурах?
14. Как можно определить фитнес-функцию в ГП?
15. Что такое интроны?
16. Приведите общий алгоритм ГП.
17. Чем отличается символьная регрессия от обычной?

18. Как ГП может быть использовано в задачах диагностики?
19. Что такое дерево, содержащее противоречия и какие проблемы оно вызывает?
20. Как можно в ГП использовать принцип модульного построения программ?

8. МАШИННОЕ ОБУЧЕНИЕ

Машинное обучение (machine learning) первоначально применялось в разработке компьютерных программ, способных производить новые или обновлять накопленные знания, используя входную информацию в виде примеров. В большинстве случаев для этих целей ранее применялись не алгоритмические, а эвристические методы изучения. Одним из основных направлений исследований в этой области является применение ГА. При этом основным объектом исследования являются продукционные системы, то есть системы правил вида: ЕСЛИ <условие> ТО <действие> (которые используются в классических экспертных системах). Одной из самых серьезных проблем (и трудно формализуемых) при разработке экспертных систем является извлечение знаний у экспертов в виде системы правил (продукций). Поэтому была предпринята попытка автоматизировать этот процесс (или упростить и повысить эффективность уже имеющихся знаний в виде продукций). Здесь в качестве потенциального решения, особи популяции, рассматривается продукция (или система продукций). Как мы знаем для использования генетических алгоритмов, прежде всего, надо разработать: 1) эффективный способ кодирования решения; 2) основные операторы ГА; 3) определить целевую (фитнесс-) функцию. Как правило, здесь используется концепция обучения с учителем. При этом дано множество примеров событий (обучающая выборка) с принадлежностью каждого из них к определенной концепции (классу). Необходимо получить множество правил – продукций, представляющих (описывающих) данное множество событий.

Проблема состоит в том, чтобы создать систему, которая изучит концепции, то есть, определит решающие правила для всех положительных примеров и отрицательных. Мы можем оценивать и сравнивать потенциальные решения в терминах значений ошибок и

сложности построенных правил. Система должна быть способна выполнить классификацию заранее неизвестных примеров, или выполнять (возможно, более чем одну) классификацию частично определенных описаний.

При машинном обучении, основанном на ГА, применяются два основных подхода:

1) Мичиганский (разработанный в Мичиганском университете основоположником ГА Холландом), где в качестве особи используется отдельная продукция, а популяция состоит из множества продукций;

2) Питтсбургский (соответственно предложенный в университете Питтсбурга де Йонгом – учеником Холланда), где в качестве особи (потенциального решения) используется закодированное все множество продукций, а популяцию образуют различные множества продукций.

Исторически Холландом [1], основоположником ГА, первым был разработан Мичиганский подход, но в настоящее время на практике более распространен Питтсбургский подход, который мы рассмотрим в следующем разделе.

8.1. Питтсбургский подход

Здесь каждая особь в популяции представляет целый набор правил (а не отдельное правило, как в альтернативном подходе). Как обычно, особи конкурируют между собой, при этом слабые особи умирают, сильные живут и воспроизводятся. Здесь обычно при реализации ГА используется пропорциональный отбор родителей; операторы кроссинговера и мутации определяются над соответствующими структурами данных. Следует отметить, что Питтсбургский подход позволяет избежать тонкой проблемы оценки эффективности отдельного правила, для которой применяются эвристические методы.

Рассмотрим один из возможных способов кодирования [49] правила на примере продукции

$\text{if}(X \& Y) \text{ then } C.$

Пусть переменная X для определенности принимает три значения x_1, x_2, x_3 , а переменная Y – два значения y_1, y_2 . Тогда для кодирования значений переменной X будем использовать три бита, соответственно для переменной Y два бита (по одному биту для каждого возможного значения). При этом значение 1 в соответствующей позиции означает, что переменная может принимать соответствующее значение. Например, код (010) для переменной X означает, что эта переменная имеет значение x_2 . Более того, этот способ кодирования позволяет описывать ситуации, когда переменная может принимать несколько значений [49]. Так код (011) для той же переменной означает, что переменная может принимать значения x_2, x_3 . Отметим, что код (111) соответствует наименьшим ограничениям, когда переменная может принимать любые (из допустимых) значения. Тогда, например, гипотеза (концепция) $\text{if}(X = x_1 \vee X = x_2) \text{ then } C = \text{TRUE}$ кодируется следующим образом (110 1). Данный метод кодирования позволяет легко учитывать ограничения на значения нескольких атрибутов (переменных) путем конкатенации (сцепления) двоичных кодов этих переменных. Так, например, гипотеза $\text{if}(X = x_1 \vee X = x_3) \& (Y = y_1 \vee Y = y_2) \text{ then } C = \text{TRUE}$ кодируется двоичной строкой (101 11 1). Следует отметить, что двоичная строка, представляющая некоторое правило – продукцию, содержит для каждого атрибута (переменной) соответствующую подстроку (даже в том случае, когда на ее значения не накладываются никаких ограничений – атрибут может принимать любые значения). Это определяет фиксированный размер двоичной строки для кодирования правила – продукции, в котором под кодирование значений каждого атрибута выделяется поле в определенных позициях.

Данный метод кодирования легко распространяется на множество продукций путем конкатенации (сцепления) двоичных строк, которые представляют отдельные продукции [49]. Например, множество продукций (представляющих гипотезы)

1: $\text{if}(X = x_3) \& (Y = y_1 \vee Y = y_2) \text{ then } C = \text{TRUE} ;$

2: $\text{if}(X = x_1 \vee X = x_2 \vee X = x_3) \& (Y = y_2) \text{ then } C = \text{FALSE} ;$

3: $\text{if}(X = x_1 \vee X = x_2) \& (Y = y_1) \text{ then } C = \text{TRUE}$

можно представить следующей двоичной строкой

(001 11 1 111 01 0 110 10 1) в соответствии с тремя правилами

1: x y c, 2: x y c, 3: x y c.

При Питсбургском подходе одна двоичная строка представляет (как и в классическом ГА) потенциальное решение – множество продукций. В этом случае популяция, как обычно, содержит множество особей потенциальных решений (систем продукций). Далее для представленного метода кодирования продукций необходимо определить генетические операторы кроссинговера и мутации. Поскольку система продукций кодируется двоичной строкой, то возможно использование стандартных генетических операторов кроссинговера (например, 1-точечного, 2-точечного, однородного, которые рассмотрены в разделе 4 и классической мутации. Но для повышения эффективности при реализации данного метода в системе GABIL, в которой применяется данный подход [49, 50], используются следующие модификации генетических операторов.

При рекомбинации применяется двухточечный оператор кроссинговера, который учитывает специфику кодирования, в частности то, что родительские особи могут иметь различную длину. Это оператор выполняется следующим образом. Первая и вторая точка кроссинговера в первом родителе выбирается случайным образом. Пусть d_1 (d_2) обозначает расстояние от левого (правого) края двоичной строки кода правила до

первой (второй) точки кроссинговера родительской особи. Во второй родительской особи точки кроссинговера также выбираются случайно, но таким образом, чтобы расстояния d_1 (d_2) сохранились. Эти ограничения обусловлены семантикой и методом кодирования системы продукций и позволяет «разрывать» вторую особь так, чтобы сохранялся «формат» представления системы продукций. При этом точка кроссинговера должна попадать на тот же разряд кода той же переменной, но возможно другого правила данной системы. Рассмотрим выполнение двухточечного кроссинговера на примере следующих особей:

	Правило1			Правило2			Правило3		
	х	у	с	х	у	с	х	у	с
Родитель 1:	001	11	1	111	01	0	110	10	1
Родитель 2:	101	11	0	001	01	0	.		

Допустим, что для первого родителя выбраны точки кроссинговера 2, 10, что соответствует $d_1=2$ (номер позиции от левого края правила с первой точкой кроссинговера) и $d_2=2$ (номер позиции от правого края правила со второй точкой кроссинговера) показано ниже квадратными скобками []:

Родитель 1: 00[1 11 1 111 0]1 0 110 10 1.

Тогда возможны следующие варианты выбора пар точек кроссинговера во втором родителе: (2,4), (2,10), (8,10). При этом первая точка кроссинговера попадает между вторым и третьим разрядами двоичного кода переменной x , а вторая точка – между первым и вторым разрядами кода переменной y . Для определенности возьмем пару точек (2,4), что дает

Родитель 2: 10[1 1]1 0 001 01 0 .

Далее, как обычно, в двух-точечном кроссинговере выполняется

обмен фрагментами двоичных кодов между точками кроссинговера (скобками []), что дает следующих потомков (две новых систем продукций):

	Правило1			Правило2			Правило3		
	х	у	с	х	у	с	х	у	с
Потомок 1:	00	[1 1]	1 0	110	10	1;			
Потомок 2:	10	[1 1 1 1	1 111	0]	1 0	001	01	0 .	

Следует отметить, что в результате выполнения такого кроссинговера число правил в системе продукций может изменяться.

Далее к полученным потомкам с небольшой вероятностью применяется один из следующих операторов мутации:

- 1) стандартный оператор мутации, инвертирующий один двоичный разряд;
- 2) оператор, изменяющий в случайно выбранном разряде нулевое значение на единичное $0 \rightarrow 1$;
- 3) оператор, который устанавливает все разряды кода выбранной переменной правила в единицу (то есть фактически исключает ее из рассмотрения, поскольку приписывает ей неопределенное значение и она не влияет на результат).

Следует особо отметить, что при Питсбургском подходе проблема построения фитнес-функции решается гораздо проще, чем в Мичиганском, поскольку здесь оценивается вся система продукций в целом. Одним из самых распространенных видов фитнес-функции [50] является следующий

$$F(h) = (y(h))^2,$$

где $y(h)$ – процент правильно классифицируемых примеров обучающей выборки с помощью гипотезы (системы продукций) h . При этом каждое потенциальное решение – система продукций оценивается на обучающей выборке «естественным образом».

8.2. Мичиганский подход

Здесь системы классификации используют структуру, в которой популяция правил закодирована в строки битов и развивается и совершенствуется на основе меняющихся входных данных, поступающих из внешней среды [1,2]. Система “обучается” на представленных входных данных по методу обучения с учителем, где для каждого набора входных данных известны правильные значения выходов. Правила в системе классификации формируют популяцию из особей, развивающихся во времени. Система классификации, представленная на рис. 8.1 состоит из следующих компонентов:

- датчик и исполнительный элемент;
- система обмена сообщениями (входные, выходные и внутренние списки сообщения);
- система правил (популяция классификаторов);
- система оценки эффективности и отбора правил («бригадный алгоритм»);
- генетический алгоритм (репродукция классификаторов).

Среда (внешнее окружение системы классификации) посылает сообщение, которое принимается датчиками системы классификации и помещается во входной список сообщений. Датчики декодируют сообщение в одно или более (декодированных) сообщений и размещают его во внутренний список сообщений. Эти сообщения активизируют классификаторы. Наиболее сильные из активизированных классификаторов размещают сообщения в списке внутренних сообщений. Эти новые сообщения могут активизировать другие классификаторы, или послать некоторые сообщения в выходной список сообщений. В последнем случае, исполнительные элементы системы классификации кодируют их в выходные сообщения, которые возвращаются во внешнюю среду. Среда оценивает действие системы посредством обратной связи с помощью

"бригадного алгоритма", который модифицирует "силу" классификаторов [1,2].

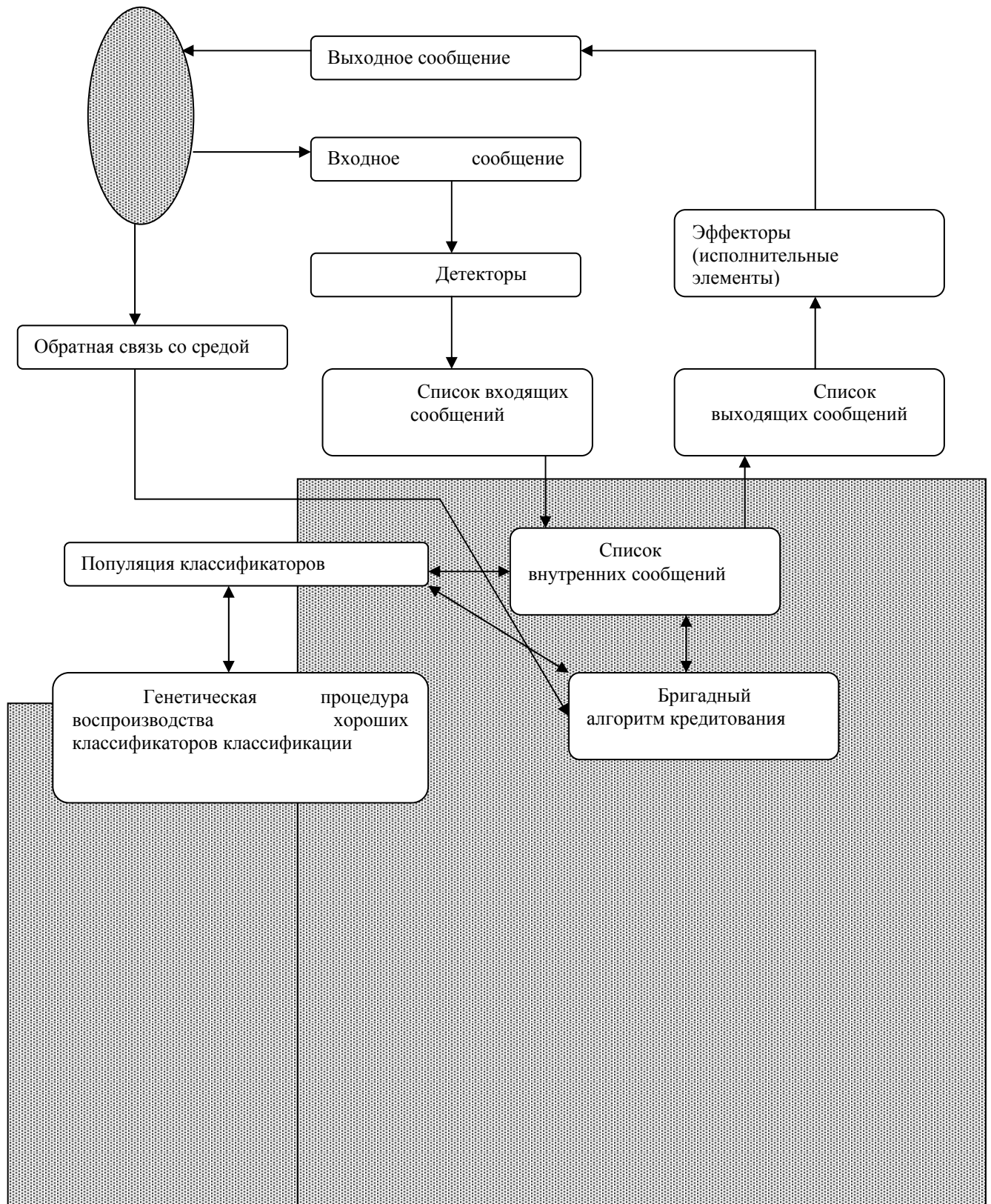


Рис. 8.1. Система классификации.

Далее рассмотрим более подробно некоторые из этих действий. Сначала определим некоторые базовые понятия. Каждый классификатор состоит из двух частей: (1)-я часть условие, и (2)-я сообщение. "Условная часть" правила представляет собой конечную строку символов из некоторого алфавита. Здесь алфавит включает "неопределенный" символ "*". Часть, представляющая сообщение, является конечной строкой из символов того же самого алфавита, кроме символа "*".

Далее мы будем использовать (шуточный) пример классификации роботов [17]. Пусть каждый робот описывается шестью атрибутами, которые могут принимать следующие значения, представленные в таблице 8.1.

Таблица 8.1

Атрибуты	Значение Атрибутов:
Форма Головы	Округлая, Квадратная, Восьмиугольная
Форма Тела	Округлая, Квадратная, Восьмиугольная
Улыбка	Да, Нет
Держит в руках	Сабля, Шарик, Флаг
Цвет куртки	Белый, Жёлтый, Зелёный, Синий, Красный
Шарф	Да, Нет

Здесь жирные буквы используются для идентификации атрибутов и их значений. Например, (**Ц=Ж**) означает “**Цвет_Куртки = Жёлтый**”.

Приведем примеры описаний концепций C_i (классов или видов) роботов :

C_1 - голова круглая и куртка белая, или голова квадратная и держит шарик;

C_2 - улыбается и держит шарик, или голова круглая;

C_3 - улыбается и не держит саблю;

C_4 - куртка белая и не носит шарф, или голова круглая и улыбается;

C_5 - улыбается и держит шарик или саблю.

Здесь каждая концепция C_i описана в терминах этих шести атрибутов и их значений. Формальное описание концепций представлено на языке VL_1 (упрощенная версия распространенного языка **Variable Valued Logic Systems**), описывающем входные события в пространстве атрибутов.

Описание концепции C представляется в виде дизъюнкции комплексов

$$C_1 \vee \dots \vee C_k \Rightarrow C.$$

При этом каждый комплекс C_i выражен посредством конъюнкции селекторов, которые являются триплетами (например, (**Ц=Ж**) для “**Цвет куртки = Желтый**”).

Концепции $C_1 - C_5$ могут быть выражены следующим образом:

$$\langle G = O \rangle \wedge \langle C = B \rangle \vee \langle G = K \rangle \wedge \langle D = Ш \rangle \Rightarrow C_1$$

$$\langle Y = Д \rangle \wedge \langle D = Ш \rangle \vee \langle G = O \rangle \Rightarrow C_2$$

$$\langle Y = Д \rangle \wedge \langle D \neq С \rangle \Rightarrow C_3$$

$$\langle C = B \rangle \wedge \langle Ш = Н \rangle \vee \langle G = O \rangle \wedge \langle Y = Д \rangle \Rightarrow C_4$$

$$\langle Y = Д \rangle \wedge \langle D = \{Ш, С\} \rangle \Rightarrow C_5$$

Каждому классификатору приписывается его "сила",

характеризующая его "важность". "Сила" важна в процессе "торговли", где классификаторы конкурируют за право послать сообщения. Мы можем представить решающее правило с помощью одного или более классификаторов. Каждый классификатор имеет следующую форму $(p_1, p_2, p_3, p_4, p_5, p_6) : d$, где p_i обозначает значение i -го атрибута ($1 \leq i \leq 6$) для областей значений, описанных выше.

Например, классификатор, $(O * * * B *)$: C_1 представляет следующее правило: "Если голова **Округлая** и куртка **Белая**, то робот соответствует концепции C_1^* . Здесь концепция фактически соответствует классу, к которому принадлежит робот.

Чтобы упростить пример, предположим, что система обучается единственной концепции C_1 . Но рассматриваемый метод может быть легко обобщен для обработки множественных концепций. В случае одной концепции каждый классификатор имеет следующую форму $(p_1, p_2, p_3, p_4, p_5, p_6) : d$, где $d=1$ (принадлежность к концепции C_1) или $d=0$ (в противном случае).

Предположим, что на некоторой стадии процесса обучения в системе имеется небольшая (случайная) популяция классификаторов q . При этом каждый классификатор имеет свою силу s . Пусть для определенности в нашем примере на текущий момент присутствуют следующие классификаторы:

$$\begin{aligned} Q1 &= (* * * C K *) : 1, & s_1 &= 12.3, \\ Q2 &= (* * D * * H) : 0, & s_2 &= 10.1, \\ Q3 &= (B K * * * *) : 1, & s_3 &= 8.7, \\ Q4 &= (* B * * * *) : 0, & s_4 &= 2.3. \end{aligned}$$

Предположим далее, что из внешней среды поступает новое входное сообщение m : (ООДСБН). Оно представляет описание одного робота с округлой головой (О), округлым телом (О), который улыбается (Д), держит саблю (С) и одет в белую (Б) куртку без шарфа (Н). Очевидно, этот робот

вписывается (соответствует) в концепцию C_1 из-за его округлой головы и белой куртки.

Анализ показывает, что это сообщение активизирует три классификатора: q_1 , q_2 и q_4 . Эти классификаторы "торгуются": предложение каждого классификатора в торге пропорционально его силе ($bid_i = b * s_i$). Самый сильный классификатор q_1 выигрывает и посылает свое сообщение. Так как сообщение дает правильную классификацию, этот классификатор получает премию $r > 0$. Тогда сила классификатора становится равной:

$$s_1 := s_1 - bid_1 + r.$$

Если бы сообщение дало неправильный ответ, "премия" r была бы отрицательна. Конкретно для коэффициентов $b = 0.2$ и $r = 4.0$, новая сила классификатора q_1 составляет $s_1 = 12.3 - 2.46 + 4.0 = 13.84$.

Одним из основных параметров системы классификаторов является период ГА t_{ga} , который определяет число временных шагов (число циклов описанных выше) между запросами ГА. Конечно, t_{ga} может быть константой, генерируемой произвольно (со средним числом, равным t_{ga}), или вообще не определено, и этот выбор может быть сделан исходя из характеристик работы системы. Так или иначе, предположим, что настало время для применения генетического алгоритма в классификации.

В данном подходе сила классификаторов рассматривается в качестве значений фитнес-функции и при выборе родителей здесь используется пропорциональный отбор (колесо рулетки).

Далее используются стандартные генетические операторы: репродукция, мутация и кроссинговер. Однако, их необходимо несколько модифицировать. Рассмотрим, например, первый атрибут. Его областью (форма головы) является $\{O, K, B, *\}$. Поэтому, при мутации, мы заменяем изменяемое значение на любое из трех других значений (с равной вероятностью):

$$\begin{aligned}
K &\rightarrow \{B, O, *\}, \\
B &\rightarrow \{K, O, *\}, \\
O &\rightarrow \{K, B, *\}, \\
* &\rightarrow \{K, B, O\}.
\end{aligned}$$

После выполнения оператора мутации сила потомка обычно остаётся такой же, как и у родительской особи. Оператор кроссинговера в данном случае не требует никакой модификации (то есть применяется классический ОК). Этому способствует также тот факт, что все классификаторы имеют равную длину. Далее применяем кроссинговер к двум родителям, например, q_1 и q_2 :

$$\begin{aligned}
(* * * \mid C K *) : 1, \quad \text{и} \\
(* * Д \mid * * Н) : 0.
\end{aligned}$$

Случайным образом генерируется номер позиции для кроссинговера (например, как показано, после третьего символа), и получаем следующий результат (особи - потомки):

$$\begin{aligned}
(* * * * * Н) : 0, \quad \text{и} \\
(* * Д C K *) : 1.
\end{aligned}$$

При кроссинговере сила полученных классификаторов определяется как среднее значение (возможно взвешенное) от значений силы родителей.

Далее процесс обучения продолжается: принимаются новые положительные и отрицательные сообщения из внешней среды, производится "торг" и модифицируются «силы» классификаторов. Можно показать, что в конечном счете популяция классификаторов сходится к некоторому числу сильных особей (классификаторов), например,

$$\begin{aligned}
(K * * * K *) : 1, \\
(B * * Ш * *) : 1, \\
(O * * * * *) : 0, \\
(* * * C Ж *) : 0.
\end{aligned}$$

Приведенный пример является, конечно искусственным и предназначен для иллюстрации основных принципов обучения, используемых в Мичиганском подходе. Следует отметить, однако, что при "торге" в примере использовалась самая простая система оценок эффективности классификаторов. Существуют и более сложные (и эффективные) методы оценки классификаторов.

8.3 Применение ГА в задачах прогнозирования

В [18,51] исследованы возможности применения ГА к некоторым проблемам анализа данных и прогнозирования. Общая проблема может быть сформулирована в следующем виде: серия наблюдений некоторого процесса имеет вид $\{(x_1, y_1), \dots, (x_n, y_n)\}$, где $x_i = (x_{i1}, \dots, x_{in})$ – независимые и y_i – соответственно зависимые переменные. В прогнозировании, например, течения некоторого заболевания независимые переменные могут представлять характеристики состояния больного в настоящем или прошлом (пусть для определенности, факторы риска атеротромбогенного инсульта). Зависимая переменная соответствует некоторым признакам будущего состояния здоровья или рекомендациям по лечению (например, 1) дальнейшее наблюдение врача; 2) медикаментозное лечение; 3) рекомендация на операцию - удаление атеросклеротической бляшки).

В качестве зависимой переменной используется прогнозируемая величина, например, возможный исход заболевания. В качестве зависимой может быть и векторная переменная, содержащая несколько компонент.

Таким образом, множество данных представляет набор аналогичных входных воздействий (влияющих факторов), произошедших в прошлом, в сочетании со значениями прогнозируемой величины, которые были получены в результате действия влияющих факторов.

При прогнозировании в данном методе в качестве особи используется некоторое условие на независимые переменные. В конечном счете, необходимо найти такое множество условий (популяцию), которое дает хорошее предсказание для зависимых переменных. Например, условие (особь) может иметь следующий вид:

$$C=(20<X_1<31)\wedge(25<X_6<28)\wedge(19<X_9<27),$$

где \wedge представляет логический оператор «И». Такое условие определяет некоторое подмножество из множества данных процесса. Фактически это условие является более формализованной формой представления правил-продукций, рассмотренных выше. Переменные X_i здесь соответствуют зависимым переменным, например, факторам риска. Таким образом, популяцию составляют множество условий на независимые переменные.

При данном подходе целью является выбор с помощью генетических алгоритмов таких наблюдений из множества данных, которые имеют сходные тенденции изменений независимых переменных, и близкие значения зависимых переменных. Эти зависимые переменные собственно и определяют прогнозируемые значения. Условие также должно однозначно удовлетворять и такому наблюдению, для которого выполняется прогноз. Таким образом, генетические алгоритмы выбирают из тех фактов, которые уже состоялись в прошлом, те, которые имеют достаточно много общего с фактом, который присутствует в настоящем. Это соответствует гипотезе локальной компактности. Можно предположить, что если из похожих фактов можно сделать близкие выводы, подобный же вывод можно сделать и из текущего наблюдения на основании гипотезы линейных зависимостей, допустим, простой операцией усреднения зависимой(ых) переменной(ых).

Фитнесс-функция для каждой особи (условия) вычисляется на основании всех данных наблюдений в обучающем множестве, которые удовлетворяют этому условию. Следует отметить, что с одной стороны,

чем больше данных, тем лучше, но с другой, большое количество данных неизбежно приводит к большому разнообразию во множестве зависимых переменных, на основании которых, собственно, и строится прогноз.

В данном подходе обычно используется следующая фитнес-функция:

$$f(\tilde{N}) = -\log \frac{\sigma}{\sigma_0} - \frac{\alpha}{N_c} + \Delta, \text{ где:}$$

σ – стандартное отклонение (дисперсия) для зависимых переменных из множества данных, удовлетворяющих заданному условию C ;

σ_0 – стандартное отклонение (дисперсия) для зависимых переменных всего множества обучающих данных;

N_c – количество наблюдений, которые удовлетворяют данному ограничению C ;

α и Δ - константы.

Как видно из приведенного выражения, фитнес-функция имеет три составляющие (слагаемых).

Первая составляющая оценивает разброс данных, отобранных данным условием. Чем ближе между собой данные в пространстве независимых переменных, тем меньше дисперсия для условия C , и тем большее значение имеет первая составляющая.

Вторая составляющая оценивает размер выборки, которую представляет данное условие. Чем большее число наблюдений удовлетворяет данному условию, тем меньше значение этой составляющей и тем больше значение целевой функции. Таким образом, вторую составляющую можно назвать "штрафом для условий с бедной статистикой". Для того, чтобы вторая составляющая была соизмерима с первой, введен масштабный коэффициент α .

Третья составляющая Δ введена для того, чтобы существовала возможность регулирования величины фитнес-функции относительно

нуля.

Представление особи и инициализация популяции

Каждая особь популяции представляется линейной структурой (унарным деревом). Каждый узел данного дерева имеет атрибуты: «имя переменной», «левая граница», «правая граница», «потомок». Таким образом, каждый узел определяет диапазон для какой-либо одной переменной. К нему может быть присоединен (или нет) узел – потомок. Таким образом, особь представляется унарным деревом, которое имеет максимум N узлов (которые соответствует N независимым переменным), и для каждой из них определен диапазон изменения (верхняя и нижняя граница).

Отбор особей для размножения.

Используются стандартные оператор репродукции, например, "колесо рулетки".

Оператор кроссинговера.

В данном методе представления особи чаще всего применяется версия оператора кроссинговера, основанная на равновероятном случайном выборе узлов от каждого родителей. Каждый ген (ограничение на значения переменной) потомка наследуется у одного из родителей с вероятностью $P_c \approx 0,5$.

Оператор мутации.

Для данного представления особей возможны следующие операторы мутации:

1. Добавление ограничения, которое добавляет в дерево новое ограничение.
2. Удаление ограничения, которое удаляет один случайный узел из дерева.
3. Расширение или сужение диапазона изменения некоторой переменной.

4. Сдвиг диапазона изменения переменной вверх или вниз:
5. Полная перестройка дерева, при которой особь полностью удаляется и вместо нее заново случайным образом генерируется новая особь.

Использование алгоритмов такого вида по сравнению со стандартными имеет следующие преимущества:

- вместо того чтобы искать функциональную зависимость прогнозируемой величины от набора влияющих факторов производится отбор аналогичных ситуаций, выбранных из множества наблюдений. Поскольку функциональная зависимость может не быть постоянной, а также зачастую весьма сложно выражается, данное преимущество весьма существенно.
- может использоваться сколь угодно большое количество влияющих переменных, а не только прошлые значения из временного ряда. Это снижает недостаток малой длины временного ряда.

Для решения задачи прогнозирования можно использовать стандартный ГА, который приведен в разделе 1.

Отбор особей для размножения

При отборе особей для размножения обычно используется стандартный оператор репродукции – "колесо рулетки", который реализует пропорциональный отбор в промежуточную популяцию. При этом особи-родители выбираются случайным образом с вероятностью, зависящей от величины их целевой функции. Для повышения эффективности целевая функция должна плавно увеличиваться от нулевого значения. Для соблюдения этого правила очень важен подбор величины смещения Δ для фитнес-функции. Построенные в процессе кроссинговера потомки замещают особи, целевая функция которых хуже средней по популяции.

Оператор кроссинговера.

При получении потомства может быть использована следующая

версия оператора кроссинговера:

Пусть имеется два родителя

$$A = \{(3.2 \leq x_6 \leq 5.5) \wedge (0.2 \leq x_8 \leq 4.8) \wedge (3.4 \leq x_9 \leq 9.9)\},$$

$$B = \{(4.5 \leq x_2 \leq 6.8) \wedge (1.4 \leq x_4 \leq 4.8) \wedge (1.2 \leq x_9 \leq 1.7) \wedge (4.8 \leq x_{16} \leq 5.1)\}$$

Каждый ген (ограничение на значения переменной) потомка наследуется у одного из родителей с вероятностью $P_c \approx 0,5$. Если у родителей есть узлы с одинаковыми переменными, то они в любом случае помещаются в разные потомки. При этом добавлено правило, которое не позволяет записывать все узлы всех родителей к одному потомку, и не поощряет передачу двух узлов подряд к одному и тому же потомку. Таким образом, возможные потомки для данных родителей могут быть, например, следующие:

$$C = \{(1.4 \leq x_4 \leq 4.8) \wedge (3.2 \leq x_6 \leq 5.5) \wedge (3.4 \leq x_9 \leq 9.9)\}$$

$$D = \{(4.5 \leq x_2 \leq 6.8) \wedge (0.2 \leq x_8 \leq 4.8) \wedge (1.2 \leq x_9 \leq 1.7) \wedge (4.8 \leq x_{16} \leq 5.1)\}$$

Здесь потомок C имеет два гена от родителя A и один ген от родителя B. Потомок D имеет один ген от родителя A, и три – от родителя B.

Оператор мутации

В дополнение к оператору кроссинговера использовались следующие операторы мутации:

1. Добавление ограничения, которое вводит в дерево новое ограничение. Например,

$$\{(3.2 \leq x_6 \leq 5.5) \wedge (0.2 \leq x_8 \leq 4.8) \wedge (3.4 \leq x_9 \leq 9.9)\}$$

$$\rightarrow \{(3.2 \leq x_6 \leq 5.5) \wedge (0.2 \leq x_8 \leq 4.8) \wedge (3.4 \leq x_9 \leq 9.9) \wedge (5.2 \leq x_{16} \leq 9.1)\}$$

2. Удаление ограничения, которое удаляет один случайный узел из дерева. Например,

$$\{(3.2 \leq x_6 \leq 5.5) \wedge (0.2 \leq x_8 \leq 4.8) \wedge (3.4 \leq x_9 \leq 9.9)\}$$

$$\rightarrow \{(3.2 \leq x_6 \leq 5.5) \wedge (3.4 \leq x_9 \leq 9.9)\}$$

3. Расширение или сужение диапазона.

$$\{(3.2 \leq x_6 \leq 5.5) \wedge (0.2 \leq x_8 \leq 4.8) \wedge (3.4 \leq x_9 \leq 9.9)\}$$

$$\rightarrow \{(3.2 \leq x_6 \leq 5.5) \wedge (0.4 \leq x_8 \leq 4.6) \wedge (3.4 \leq x_9 \leq 9.9)\}$$

4. Сдвиг диапазона вверх или вниз:

$$\{(3.2 \leq x_6 \leq 5.5) \wedge (0.2 \leq x_8 \leq 4.8) \wedge (3.4 \leq x_9 \leq 9.9)\}$$

$$\rightarrow \{(3.6 \leq x_6 \leq 5.9) \wedge (0.2 \leq x_8 \leq 4.8) \wedge (3.4 \leq x_9 \leq 9.9)\}$$

5. Полная перестройка дерева, где особь полностью удаляется, и вместо нее заново случайным образом генерируется новая особь.

Таким образом, глобальный оператор мутации выбирает с равной вероятностью один из этих операторов.

В качестве критерия останова алгоритма обычно используется выполнение условие окончания: повторение лучшего результата заданное количество поколений подряд.

При сокращении промежуточной популяции применяется стратегия элитизма: особь с наилучшим значением целевой функции обязательно переходит в следующее поколение. Для замедления вырождения выборки к одной особи, все особи, которые совпадают с наилучшей, подвергаются мутации.

Определение параметров алгоритма

Подбор параметров α и Δ .

Как указано выше, параметр α определяет размер выборки для каждого условия. Чем ниже этот фактор, тем меньшее значение имеет размер выборки, и тем большее – близость между собой значений зависимых переменных. Параметр Δ определяет величину смещения фитнес-функции на положительную полуплоскость. Этот параметр должен быть таким, чтобы минимальное значение целевой функции было как можно ближе к 0, но имело положительное значение. Это способствует эффективному функционированию оператора репродукции. Для подбора параметров фитнес-функции необходимо оценить зависимость точности результата от этих параметров. Например, для различных значений α и Δ

можно выполнить поиск решений, среди которых найти лучшее. Затем выполнено сравнение лучшего решения с действительным, и найдена погрешность результата.

Подбор размера популяции и критерия остановки программы

Аналогично предыдущему пункту производится подбор таких параметров, как размер популяции и количество поколений, в течение которых не улучшается результат (погрешность). Чем меньше размер популяции, тем быстрее выполняется программа, но тем меньше шансов найти оптимальное решение. Аналогично большое число шагов программы до остановки способствует нахождению хорошего решения и замедляет работу программы.

Подбор вероятности мутации

После выполнения оператора кроссинговера все потомки могут быть подвергнуты мутации с заданной вероятностью. Чем выше эта вероятность, тем больше ГА превращается в алгоритм случайного поиска. Соответственно должно увеличиваться и количество поколений до достижения результата.

Таким образом, в данном разделе рассмотрены возможности применения генетических алгоритмов и генетического программирования к задачам прогнозирования, которые при грамотной реализации дают результаты, сопоставимые с результатами, полученными с помощью многослойных нейронных сетей прямого распространения, но имеют при этом более ясную интерпретацию.

Напомним, что основной проблемой при разработке классических экспертных систем является формирование базы знаний – множества правил-продукций. Для этого привлекаются, как правило, квалифицированные эксперты. Определенную проблему представляет также и обновление знаний в процессе эксплуатации экспертной системы. Преимуществом является, как правило, «прозрачность» правил-продукций

и возможность проследить сам процесс вывода – заключения экспертной системы.

В эволюционном подходе предпринята попытка объединить преимущества обоих указанных методов создания экспертных систем – классического и нейросетевого. Фактически этот подход является развитием классического метода построения экспертных систем. Знания хранятся здесь в виде формализованных правил-продукций (почти также как и в обычных экспертных системах). Но здесь есть возможность автоматически строить эти правила-продукции по имеющейся обучающей выборке. Суть данного подхода заключается в том, что из обучающей выборки автоматически выводится множество правил-продукций, которое позволяет с минимальной (в некотором смысле) ошибкой решать поставленную задачу. Для медицинских приложений такой подход является чрезвычайно перспективным – фактически здесь из имеющихся статистических данных по некоторому заболеванию можно фактически автоматически получить методику его диагностирования и прогноза течения заболевания. Это становится возможным благодаря использованию методов эволюционных вычислений.

8.4 Контрольные вопросы к разделу 8

1. В чем суть машинного обучения?
2. Какие два основных подхода применяются в машинном обучении ?
3. На чем основан Питсбургский подход ?
4. Как можно закодировать одну продукцию?
5. Приведите возможный вариант кодирования системы продукций.
6. Опишите двух точечный кроссинговер, применяемый в системе

GABIL.

7. Какие операторы мутации могут использоваться при Питсбургском подходе?
8. Какой вид имеет фитнес-функция в Питсбургском подходе?
9. Опишите систему классификации, которая используется в Мичиганском подходе.
10. Что такое концепция и как она представляется?
11. Что такое «сила классификатора» и как она используется?
12. Опишите кроссинговер, который применяется в Мичиганском подходе.
13. Как можно использовать ГА в прогнозировании ?
14. Что представляет особь в случае использования ГА для прогнозирования?
15. Какую фитнес-функцию можно использовать при прогнозировании?
16. Опишите оператор кроссинговера, который может быть использован при прогнозировании.
17. Опишите оператор мутации, который может быть использован при прогнозировании.
18. Какие параметры ГА необходимо настроить для эффективной работы ГА при прогнозировании?

ЧАСТЬ 4

ЭВОЛЮЦИОННЫЕ СТРАТЕГИИ И ПРОГРАММИРОВАНИЕ, РЕАЛИЗАЦИЯ

9.ЭВОЛЮЦИОННЫЕ СТРАТЕГИИ

Эволюционные стратегии (ЭС), также как и предыдущие парадигмы основаны на эволюции популяции потенциальных решений, но в отличие от них здесь используется генетические операторы на уровне фенотипа, а не генотипа, как это делается в ГА. Разница в том, что ГА работают в пространстве генотипа – кодов решений, в то время как ЭС производят поиск в пространстве фенотипа – векторном пространстве вещественных чисел. В ЭС учитываются свойства хромосомы «в целом», в отличие от ГА, где при поиске решений исследуются отдельные гены. В природе один ген может одновременно влиять на несколько свойств организма. С другой стороны одно свойство особи может определяться несколькими генами. Естественная эволюция основана на исследовании совокупности генов, а не отдельного (изолированного) гена.

В эволюционных стратегиях целью является движение особей популяции по направлению к лучшей области ландшафта фитнес-функции. ЭС изначально разработаны для решения многомерных

оптимизационных задач, где пространство поиска – многомерное пространство вещественных чисел. Иногда при решении задачи накладываются некоторые ограничения, например, вида $g_i(x) > 0$.

Ранние эволюционные стратегии (ЭС) основывались на популяции состоящей, из одной особи, и в них использовался только один генетический оператор – мутация. Здесь для представления особи (потенциального решения) была использована идея, не представленная в классическом генетическом алгоритме, которая заключается в следующем.

Здесь особь представляется парой действительных векторов

$$v = (\bar{x}, \bar{\sigma}), \quad (9.1)$$

где \bar{x} - точка в пространстве решений и $\bar{\sigma}$ - вектор стандартных отклонений (вариабельность) от решения. В общем случае особь популяции определяется вектором потенциального решения и вектором «стратегических параметров» эволюции. Обычно это вектор стандартных отклонений (дисперсия), хотя допускаются (и иногда используются) и другие статистики.

Единственным генетическим оператором в ЭС является оператор мутации, который выполняется путем сложения координат вектора-родителя со случайными числами, подчиняющихся закону нормального распределения, следующим образом:

$$\bar{x}^{t+1} = \bar{x}^t + N(0, \bar{\sigma}), \quad (9.2)$$

где $N(0, \bar{\sigma})$ - вектор независимых случайных чисел Гаусса с нулевым средним значением и стандартным отклонением σ . Как видно из приведенной формулы величина мутации управляется нетрадиционным способом. Иногда эволюционный процесс используется для изменения и самих стратегических параметров σ , в этом случае величина мутации

эволюционирует вместе с искомым потенциальным решением. Это соответствует адаптивному ГА с изменяемым шагом мутации.

Интуитивно ясно, что увеличение отклонения подобно увеличению шага поиска на поверхности ландшафта. Высокая вариабельность способствует расширению пространства поиска и эффективна при нахождении потенциальных зон (суб)оптимальных решений и соответствует высоким значениям коэффициента мутации. В тоже время малые значения вариабельности позволяют сфокусироваться на поиске решения в перспективной области. В данном случае стратегические параметры стохастически определяют величину шага поиска: большая вариабельность ведет к большим шагам. Отметим, что поскольку отклонения генерируются стохастически (по нормальному закону), то большая вариабельность может давать маленький шаг и наоборот. Известно, что 68,26% случайных чисел при нормальном распределении попадают в интервал, определяемый стандартным отклонением σ ; 95% чисел попадают в интервал $1,96\sigma$ и т.д.

9.1. Двукратная эволюционная стратегия

Здесь потомок принимается в качестве нового члена популяции (он заменяет своего родителя), если значение фитнес функции (ЦФ) на нем лучше, чем у его родителя и выполняются все ограничения. Иначе, (если значение фитнес-функции на нем хуже, чем у родителей), потомок уничтожается и популяция остается неизменной.

Рассмотрим выполнение оператора на конкретном примере следующей функции [17]:

$$f(x_1, x_2) = 21,5 + x_1 \cdot \sin(4\pi x_1) + x_2 \cdot \sin(20\pi x_2)$$

$$\begin{aligned} -3.0 \leq x_1 \leq 12.1 & \quad \bar{x} = (x_1, x_2) \\ 4.1 \leq x_2 \leq 5.8 & \quad \bar{\sigma} = (\sigma_1, \sigma_2)' \end{aligned} \quad (9.3)$$

Для определенности предположим, что в t -поколении текущая особь имеет вид:

$$(\bar{x}', \sigma) = ((5.3; 4.9), (1.0; 1.0)) \quad (9.4)$$

Тогда потомок определяется следующим образом:

$$\left. \begin{aligned} x_1^{t+1} &= x_1^t + N(0; 1.0) = 5.3 + 0.4 = 5.7 \\ x_2^{t+1} &= x_2^t + N(0; 1.0) = 4.9 - 0.3 = 4.6 \end{aligned} \right\} \text{потомок} \quad (9.5)$$

Поскольку $f(x') = f(5.3; 4.9) = 18.383705 < 24.849532 = f(5.7; 4.6) = f(x^{t+1})$ (значение ЦФ потомка лучше, чем у родителя), то полученный потомок заменяет родителя.

В целом алгоритм процесса эволюции двукратной (1+1) эволюционной стратегии можно сформулировать следующим образом.

1. Выбрать множество P параметров X , необходимых для представления решения данной проблемы и определить диапазон допустимых изменений каждого параметра:

$$\{x_{1\min}, x_{1\max}\}, \{x_{2\min}, x_{2\max}\}, \dots, \{x_{P\min}, x_{P\max}\},$$

установить номер поколения (итерации) $t=0$;

задать стандартное отклонение σ_i для каждого параметра, функцию f , для которой необходимо найти оптимум и максимальное число поколений k .

2. Для каждого параметра случайным образом выбрать начальное значение из допустимого диапазона: множество этих значений составляет начальную популяцию (из одной особи) $X^t = (x_1, x_2, \dots, x_P)$.

3. Вычислить значение оптимизируемой функции f для родительской особи $F^p = f(X^t)$.

4. Создать новую особь –потомка в соответствии с (9.2)

$$\bar{x}^* = \bar{x}^t + N(0, \bar{\sigma}).$$

5. Вычислить значение f для особи-потомка $F^0=f(X^*)$.

6. Сравнить значения функций f для родителя и потомка; если значение потомка F^0 лучше, чем у родительской особи, то заменить родителя на потомка

$$\bar{x}^t = \bar{x}^*,$$

иначе оставить в популяции родителя.

7. Если не достигнуто максимальное число поколений $t < k$, то переход на шаг 4, иначе выдать найденное решение X^t .

Несмотря на то, что фактически здесь популяция состоит из одной особи, рассмотренная стратегия называется двукратной ЭС. Причина в том, что здесь фактически происходит конкуренция потомка и родителя. Обычно вектор стандартных отклонений σ остается неизменным в течении всего процесса эволюции. Если все его компоненты одинаковы и оптимизационная задача регулярна, то можно доказать следующую теорему сходимости [17,3]. *Теорема.* Для $\sigma > 0$ и регулярной оптимизационной задачи с $f_{opt} > -\infty$ (минимизация), либо $f_{opt} < +\infty$ (максимизация), имеет место

$$P\{\lim_{t \rightarrow \infty} f(\bar{x}^t) = f_{opt}\} = 1. \quad (9.6)$$

Эта теорема формулирует, что оптимальное решение регулярной оптимизационной задачи находится с вероятностью, равной единице при $t \rightarrow \infty$, но при этом совершенно не говорится, как и каким образом двигаться к этому оптимальному решению. Поэтому, чтобы

оптимизировать скорость сходимости этого процесса Решенберг [3] (основоположник ЭС) предложил правило успеха «1/5» .

Смысл его заключается в следующем - правило применяется после каждых k поколений процесса (где k – параметр этого метода):

$$\sigma^{t+1} = \begin{cases} c_d \cdot \sigma^t, & \text{если } \varphi(k) < 1/5 \\ c_i \cdot \sigma^t, & \text{если } \varphi(k) > 1/5 \\ \sigma^t, & \text{если } \varphi(k) = 1/5 \end{cases}, \quad (9.7)$$

где: $\varphi(k)$ - отношение числа успешных мутаций к общему числу произведенных мутаций k (число успехов деленное на k), которое называется коэффициентом успеха для оператора мутации в течении k последних поколений; величина $c_i > 1$, $c_d < 1$ – регулирует увеличение/уменьшение отклонения мутации.

Обычно на практике оптимальные значения полагают: $c_d = 0.82$; $c_i = 1/0.82 = 1.22$. Смысл этого правила в следующем:

- если коэффициент успеха $\varphi(k) > 1/5$, то отклонение σ^{t+1} увеличивается (мы идем более крупными шагами);
- если коэффициент успеха $\varphi(k) < 1/5$, то отклонение σ^{t+1} уменьшается (шаг поиска уменьшается).

Идеи Решенберга получили дальнейшее развитие в концепции «эволюция окна» [52] при которой результат применения оператора мутации принимается только в том случае, если он лежит в пределах некоторой окрестности (окна) родительской особи в пространстве решений. Динамическое изменение шага мутации в сочетании с эволюцией величины окна ведет к метаэволюции [52].

Иногда рекомендуется устанавливать коэффициент мутации обратно пропорционально числу переменных в потенциальном решении (особи) и прямо пропорционально расстоянию от точки оптимального решения.

Конечно, в реальных приложениях точное расположение оптимума неизвестно. Однако, иногда может быть известна априорная информация об оптимуме (например, порядок величины). Даже ограниченная информация может быть полезна в процессе поиска в ЭС.

9.2. Многократная эволюционная стратегия

По сравнению с двукратной отличается размером популяции ($N > 2$) и имеет некоторые дополнительные отличия:

- все особи в поколении имеют одинаковую вероятность выбора для мутации;
- имеется возможность введения оператора рекомбинации (типа – однородного ОК в ГА, рассмотренного в разделе 4), где два случайно выбранных родителя производят потомка по следующей схеме:

$$\begin{aligned}
 (\bar{x}^1, \bar{\sigma}^1) &= ((x_1^1, \dots, x_n^1), (\sigma_1^1, \dots, \sigma_n^1)) \\
 (\bar{x}^2, \bar{\sigma}^2) &= ((x_1^2, \dots, x_n^2), (\sigma_1^2, \dots, \sigma_n^2)) \\
 (\bar{x}, \bar{\sigma}) &= ((x_1^{q_1}, \dots, x_n^{q_n}), (\sigma_1^{q_1}, \dots, \sigma_n^{q_n}))
 \end{aligned}
 \tag{9.8}$$

где $q_i=1$ или $q_i=2$, $i=1, \dots, n$ (т.е. каждая компонента потомка копируется из первого или второго родителя).

Имеется еще одно сходство между двукратными и многократными эволюционными стратегиями. При обоих видах ЭС производится только один потомок. В двукратных стратегиях потомок соревнуется со своим родителем. В многократной стратегии самая слабая особь уничтожается.

В современной литературе используются следующие обозначения:

(1+1) – ЭС – двукратная стратегия (1 родитель производит 1 потомка);

$(\mu+1)$ – ЭС – многократная стратегия (μ родителей производят 1 потомка);

$(\mu+\lambda)$ – ЭС, где μ -родителей производят λ -потомков и отбор μ лучших представителей производится среди объединенного множества ($\mu+\lambda$) особей) родителей и потомков;

(μ,λ) – ЭС, где μ -особей родителей порождает λ -потомков; причем $\lambda>\mu$ и процесс выбора лучших производится только на множестве потомков.

Следует подчеркнуть, что в обоих последних видах ЭС обычно число потомков существенно больше числа родителей $\lambda>\mu$ (иногда полагают $\lambda/\mu=7$).

В кратных стратегиях часто используется двухуровневое обучение, где параметр σ не является постоянным, но изменяется по некоторому детерминированному алгоритму (например, по правилу 1/5).

При рекомбинации потомок генерируется в несколько этапов (стадий):

1) *выбрать 2 родительских особи*

$$\begin{aligned}(\bar{x}^1, \bar{\sigma}^1) &= ((x_1^1, \dots, x_n^1), (\sigma_1^1, \dots, \sigma_n^1)) \\ (\bar{x}^2, \bar{\sigma}^2) &= ((x_1^2, \dots, x_n^2), (\sigma_1^2, \dots, \sigma_n^2))\end{aligned}\tag{9.9}$$

2) *выполнить оператор рекомбинации.*

Обычно используется два типа ОК:

а) дискретный ОК, для которого потомок строится следующим образом

$$(\bar{x}, \bar{\sigma}) = ((x_1^{q_1}, \dots, x_n^{q_n}), (\sigma_1^{q_1}, \dots, \sigma_n^{q_n})),\tag{9.10}$$

где $q_i=1$ или $q_i=2$, $i=1, \dots, n$.

б) промежуточный ОК, для которого потомок генерируется так

$$(\bar{x}, \bar{\sigma}) = (((x_1^1 + x_1^2)/2, \dots, (x_n^1 + x_n^2)/2), ((\sigma_1^1 + \sigma_1^2)/2, \dots, (\sigma_n^1 + \sigma_n^2)/2)) \quad (9.11)$$

Каждый из этих операторов может применяться также в глобальном режиме, где новая пара родителей будет выбрана для каждого компонента вектора потомка.

3) *применить оператор мутации ОМ к полученному потомку*

$$\begin{aligned} (\bar{x}^1, \bar{\sigma}^1) \quad \bar{\sigma}^1 &= \bar{\sigma} \cdot e^{N(0, \Delta \bar{\sigma})}; \\ \bar{x}^1 &= \bar{x} + N(0, \sigma^1) \end{aligned} \quad (9.12)$$

Чтобы улучшить степень сходимости ЭС, часто добавляется дополнительный параметр управления θ . Этот новый параметр корректирует параметр σ мутации. Для ЭС с собственным σ_i для каждой переменной x_i , выделенное направление поиска может быть установлено только по осям системы координат. Теперь каждая особь в поколении представляется тройкой (x, σ, θ) .

Операторы рекомбинации подобны тем, которые были показаны ранее, но оперируют с тремя компонентами (в отличие от стандартного представления, использующего две компоненты). Например, оператор мутации производит потомка (x', σ', θ') из (x, σ, θ) следующим образом:

$$\begin{aligned} \sigma' &= \sigma \cdot e^{N(0, \Delta \sigma)}, \\ \theta' &= \theta + N(0, \Delta \theta), \\ x' &= x + C(0, \sigma', \theta'), \end{aligned} \quad (9.13)$$

где $\Delta\theta$ - дополнительный параметр метода; $C(0, \sigma', \theta')$ - вектор независимых случайных чисел Гаусса с нулевым средним значением и стандартным отклонением σ .

В заключение приведем укрупненный алгоритм решения задачи с помощью ЭС.

1. Инициализировать популяцию.
2. Выполнить рекомбинацию, используя μ родительских особей для производства λ потомков.
3. Выполнить мутацию для всех особей – потомков.
4. Вычислить значение целевой функции f для λ или $\mu + \lambda$ особей в зависимости от типа ЭС (соответственно (μ, λ) или $(\mu + \lambda)$ ЭС).
5. Выбрать μ особей для новой популяции.
6. Если не выполнен критерий останова (например, максимальное число поколений), то переход на шаг 2, иначе выдать найденное решение

Эволюционные стратегии применяются, в основном, в численной оптимизации, так как они были (по крайней мере, первоначально) посвящены функциональным задачам оптимизации в вещественной области. ЭС позволяет эффективно решать многие задачи стандартной и сложной нелинейной оптимизации с ограничениями (и без них).

Они являются примерами эволюционных программ, которые используют соответствующие структуры данных (вещественные векторы, расширенные параметрами стратегии управления) и "генетические" операторы, ориентированные на решение определенных задач.

9.3 Сравнение эволюционной стратегии и генетических алгоритмов

Основная разница между этими методами состоит в кодировании особей. Эволюционная стратегия была создана и развивалась для решения задач численной оптимизации. Поэтому решение (особь) представляется в виде вектора вещественных чисел. Хотя в настоящее время известны новые работы, в которых ЭС применяется для решения задач дискретной оптимизации.

С другой стороны генетические алгоритмы были разработаны в качестве общего метода решения оптимизационных задач. В простом классическом ГА особь представляется в виде двоичного вектора.

Поэтому, может быть, не совсем корректно сравнивать эти два направления, так как они были разработаны для разных целей. Но их объединяет принцип отбора лучших решений (отбор по Дарвину сильнейших особей). Однако имеются существенные различия между этими подходами.

Первое различие между ЭС и ГА - это форма представления решений. ЭС использует вектор вещественных чисел, ГА – двоичный.

Второе различие между ГА и ЭС скрыто в процедуре выбора. В ЭС μ -особей родителей порождают промежуточную популяцию, которая состоит из λ -потомков, путем оператора мутации и репродукции. На промежуточной популяции выполняется процедура отбора, которая сокращает эту популяцию обычно до исходной. В простейшем случае оставляются μ лучших особей – решение (причем все они разные).

В ГА оператор репродукции ОР (аналог процедуры выбора) генерирует промежуточную популяцию, причем число представителей каждой особи зависит от значений целевой функции для этих особей. Т.е. сильнейшие представители промежуточной популяции имеют нескольких

представителей, и наоборот, наиболее слабые особи промежуточной популяции могут быть не представлены. Далее случайным образом производится выбор пар для выполнения ОК, ОМ.

Третье различие заключается в относительном порядке выполнения процедур отбора и рекомбинации. В ЭС процедура отбора выполняется после выполнения оператора репродукции. В ГА наоборот, ОР работает перед ОК и ОМ.

Следующее различие в том, что в классическом ГА параметры операторов рекомбинации обычно остаются постоянными (вероятность ОМ, ОК), а в ЭС параметры $\bar{\sigma}$ изменяются.

ЭС и ГА по разному учитывают ограничения: в ЭС есть множество неравенств $g_1(\bar{x}) \geq 0, \dots, g_q(\bar{x}) \geq 0$, которое рассматривается как часть оптимизационной задачи. В ГА ограничения обычно учитываются в виде штрафных функций, т.е. в неявном виде.

Из выше сказанного следует, что ЭС и ГА, хотя и имеют много общего, но и имеют существенные отличия. Но в настоящее время есть явная тенденция сближения этих двух направлений. С одной стороны современные ГА часто используются для представления решения векторами вещественных чисел, с другой стороны ЭС в качестве ОР использует не только ОМ, но и операторы типа ОК.

Недавно были введены в ГА и ЭС следующие ОК:

Два родителя \bar{x}_1, \bar{x}_2 производят двух потомков \bar{y}_1, \bar{y}_2 , которые являются линейной комбинацией родителей.

$$\begin{aligned}\bar{y}_1 &= \alpha \bar{x}_1 + (1 - \alpha) \bar{x}_2 \\ \bar{y}_2 &= (1 - \alpha) \bar{x}_1 + \alpha \bar{x}_2\end{aligned}\tag{4.1}$$

Такой ОК в ГА называют арифметическим кроссинговером или средним ОК при $\alpha=1/2 \rightarrow \bar{y}_1 = \frac{\bar{x}_1 + \bar{x}_2}{2}$ (фактически мы их рассматривали в

разделе 4.3). В ЭС этот оператор называется промежуточным кроссинговером.

Элементы самоадаптации, характерные ранее для ЭС (изменение вектора отклонения $\bar{\sigma}$), в настоящее время используется и в ГА, например, разрабатываются адаптивные ГА, в которых изменяются вероятности ОМ и ОК (которые также были рассмотрены в разделе 4.7).

9.4 Контрольные вопросы к разделу 9

1. Как представляется потенциальное решение в ЭС?
2. Какой генетический оператор применяется в ЭС?
3. Как выполняется мутация в ЭС?
4. Опишите двукратную ЭС.
5. Сформулируйте правило успеха в ЭС.
6. Что такое многократная ЭС?
7. Какие операторы рекомбинации могут быть использованы в ЭС?
8. Сформулируйте общий алгоритм решения задачи с использованием ЭС.
9. Что общего между ГА и ЭС?
10. Какие различия между ГА и ЭС?
11. Зачем и как вводится третья компонента для представления решения в ЭС?

10. ЭВОЛЮЦИОННОЕ ПРОГРАММИРОВАНИЕ

Эволюционное программирование (ЭВ) было основано в ранних работах Фогеля [4]. Фогель считал, что в основе интеллектуальных систем лежит адаптивное поведение в изменяющейся окружающей среде. ЭВ, также как и ЭС использует подход «сверху вниз» т.е. эволюцию на уровне фенотипа. При этом в ЭВ представлен, возможно, самый гибкий подход по сравнению с другими парадигмами эволюционных вычислений, где форма представления потенциального решения и генетические операторы адаптируются в достаточно широких пределах к рассматриваемой проблеме. В классическом ЭВ используется только один оператор мутации (как и в классической ЭС), но в современных модификациях ЭВ (также как и ЭС) допускается также оператор рекомбинации (кроссинговера). В отличие от ЭС, где обычно потомков производится существенно больше, чем родителей, в ЭВ число потомков, как правило, равно числу родительских особей. При выборе родителей чаще используется ранговый или турнирный отбор. Когда в результате порождения потомков (путем мутации) популяция удваивается, ее особи (родители и потомки) ранжируются и лучшая половина образует популяцию следующего поколения. Укрупненный алгоритм эволюции в ЭС представлен ниже.

1. Инициализировать популяцию, номер поколения $k=1$.
2. Подвергнуть популяцию влиянию окружающей среды.
3. Вычислить значение фитнес-функции для каждой особи.
4. Выполнить оператор мутации для каждой особи популяции.
5. Оценить каждую особь (родителя и потомка).
6. Выбрать особи популяции следующего поколения.
7. Если не выполнено условие останова ($k < k_{\max}$ и т.п.), то переход на шаг 2, иначе выдать полученное решение.

Здесь, как обычно, популяция инициализируется случайным образом. При решении конкретных задач для каждой компоненты особи генерируется реальное значение в пределах некоторого динамического диапазона. Число особей в популяции, как и в ГА, обычно составляет от нескольких десятков до нескольких сотен.

В классическом ЭВ для простоты и общности описания воздействие окружающей среды описывается в виде последовательности символов конечного алфавита. Поэтому здесь для представления потенциального решения используется модель конечного автомата.

10.1. Конечный автомат в качестве генома

Итак, в классическом ЭВ особь представляется в виде конечного автомата. В определенном смысле конечные автоматы являются подмножеством машин Тьюринга, которые являются базовой моделью в теории вычислительной сложности. Известно, что машины Тьюринга способны, в принципе, решить любые задачи. Конечные автоматы, конечно, не обладают такой «вычислительной мощностью», но они способны решать достаточно сложные задачи.

Напомним, что конечный автомат является совокупностью пяти объектов $A=(Y,X,Z,\delta,\lambda,y_0)$, где Y, X, Z - конечные множества состояний, входных и выходных сигналов соответственно; $\delta:Y \times X \rightarrow Y$ - функция переходов, определяющая следующее состояние автомата; $\lambda:Y \times X \rightarrow Z$ - функция выхода, определяющая выходной сигнал, y_0 - начальное состояние. Обычно автомат представляется таблицей или графом переходов-выходов.

Например, табл. 10.1 представляет конечный автомат [4] с входным алфавитом из двух символов $\{0,1\}$, выходным алфавитом $\{\alpha,\beta,\gamma\}$ и тремя состояниями (A,B,C). Здесь на пересечении строки (текущего

состояния) и столбца (входного сигнала) приводятся следующее состояние и выходной сигнал автомата.

Таблица 10.1.

Y	X	
	0	1
A	B, β	A, β
B	B, γ	C, α
C	B, β	A, γ

Кроме приведенной табличной формы автомат также часто представляется графом переходов и выходов. Для примера на рис.10.1 показан граф переходов – выходов автомата, представленного табл.10.1.

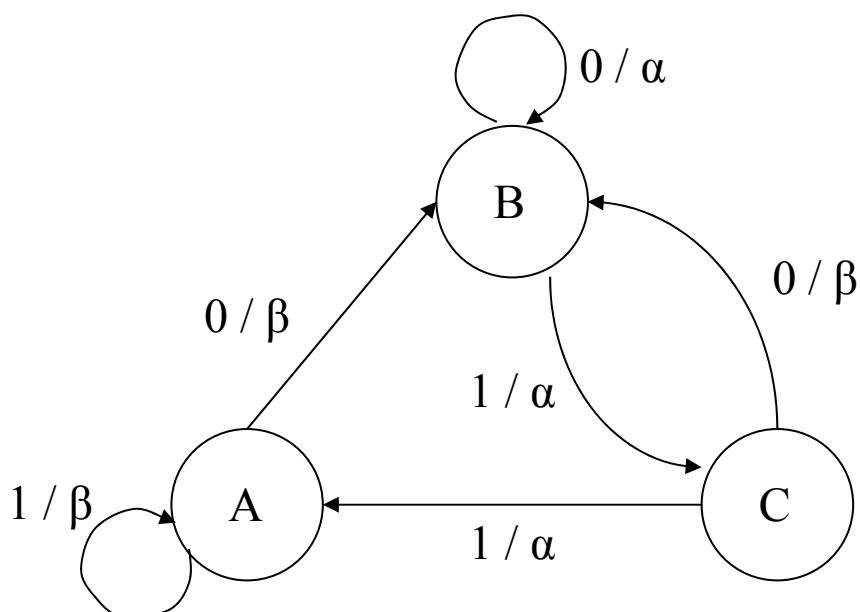


Рис. 10.1. Граф переходов-выходов

Очевидно, автомат на входную последовательность в зависимости от текущего состояния выдает выходную последовательность. Например, если автомат табл.10.1 находится в состоянии A , и на его вход поступает последовательность $0, 1, 1$, то он выдает выходную последовательность β, α, γ . При этом входная последовательность моделирует воздействие окружающей среды, а выходная последовательность – реакцию на это воздействие. Таким образом, рассматриваются последовательности событий, которые отмечаются символами x_1, x_2, \dots, x_n , и алгоритм должен прогнозировать следующее событие x_{n+1} на основе известных предыдущих n символов (событий). Цель эволюции состоит в поиске особи, которая позволяет в каком-то смысле наилучшим образом решать данную проблему.

Таким образом, множество потенциальных решений составляет популяцию конечных автоматов. Как правило, автоматы имеют относительно небольшое число состояний. В процессе эволюции применяется оператор мутации (кроссинговер здесь не используется). Обычно используются пять возможных операторов мутации: изменение выходного сигнала (для данного перехода), изменение следующего состояния в переходе, изменение начального состояния, добавление одного состояния, сокращение одного состояния. Эти операторы мутации выбираются в соответствии с некоторым распределением вероятности. При этом возможно применение не одного, а нескольких операторов мутации. Например, на рис.10.2 показан результат мутации автомата рис.10.1, где изменен выходной сигнал перехода из состояния $C \xrightarrow{0/\beta} B$ (на $C \xrightarrow{0/\alpha} B$).

После применения мутации к текущей популяции, лучшие n особей переносятся в следующее поколение. Отметим, что в отличие от ГА (где

сначала используется оператор отбора, а затем кроссинговер и мутация) в ЭВ (также как и ЭС) мутация применяется до отбора.

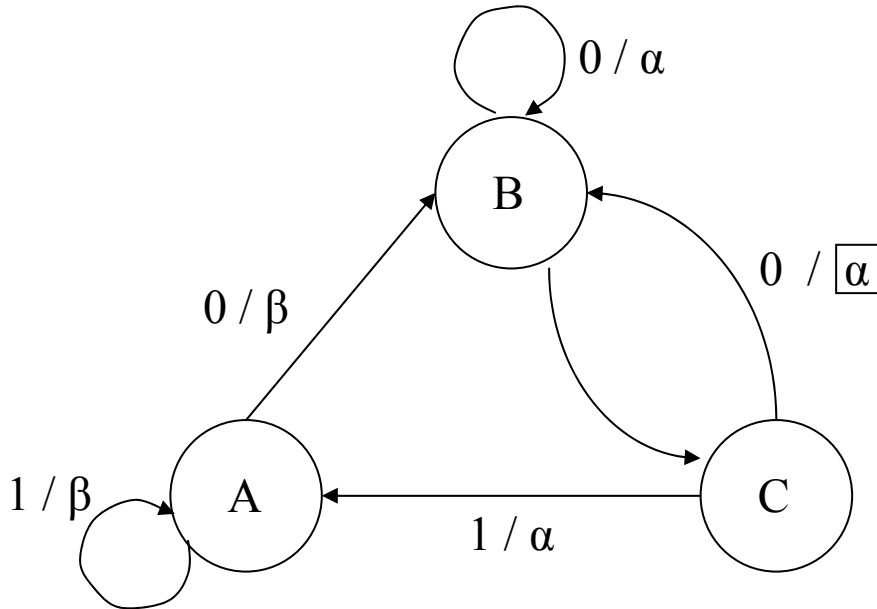


Рис.10.2. Автомат – «мутант»

Рассмотрим особенности выполнения основных этапов алгоритма эволюции в эволюционном программировании.

1) Инициализация. Начальная популяция состоит из m автоматов $A_i = (Y_i, X_i, Z_i, \delta_i, \lambda_i, y_{0i}) \quad \forall i \in \{1, 2, \dots, m\}$, которые генерируются случайным образом. Число состояний в автоматах $N_i = |Y_{ii}|$ выбирается также случайно с равной вероятностью из $\{1, 2, \dots, N_{\max}\}$. Начальные состояния y_{0i} и переходы в следующее состояние присваиваются случайно в зависимости от числа состояний данного (случайного) автомата, как и выходные символы z_i из заданного конечного алфавита. При этом с каждым автоматом ассоциируется важный параметр λ_i , который представляет среднее число мутаций, применяемых к родительской особи

при производстве потомков (мутантов). Обычно первоначально полагается $\lambda_i=5$, и счетчику поколений присваивается $k=1$.

2) Фитнесс-функция. Вид фитнес-функции существенно зависит от рассматриваемой проблемы (ее целевой функции) и для некоторых задач (прогнозирования и управления) будет описан ниже.

3) Мутация. Каждый из родительских автоматов A_i мутирует и производит потомка A_i' путем применения (возможно нескольких) операторов мутации. В задачах прогнозирования их число M_i обычно определяется заранее экспериментальным путем. В экспериментах по управлению M_i получается в результате вероятностного эксперимента со случайной переменной, значения которой для потомка λ_i' определяются из значения родителя λ_i согласно распределению Пуассона: $\lambda_i' = \lambda_i + 0.5N(0,1)$, где $N(0,1)$ представляет распределение Гаусса с нулевым средним значением и единичной дисперсией. Если λ_i' или M_i превышает N_i или меньше 1, то им присваивается другое значение. Аналогично, если родительская особь имеет только одно состояние, то уменьшение числа состояний или назначение нового состояния запрещено. Используются следующие операторы мутации.

а) Добавление состояния: в родительский автомат добавляется новое состояние. Переходы из этого состояния и выходные сигналы назначаются случайным образом. Переходы в предшествующее состояние родительского автомата разрываются и перебрасываются в новое состояние, что повышает «активность» этого фрагмента автомата.

б) Сокращение состояния: случайно выбирается одно из состояний родительского автомата и сокращается. Все переходы в это состояние случайным образом перебрасываются в оставшиеся состояния. Если сокращаемое состояние было начальным, то случайно выбирается другое начальное состояние для нового автомата – «мутанта».

в) Изменение начального состояния: в родительском автомате случайно выбирается другое начальное состояние.

г) Изменение перехода: случайно выбранный переход перебрасывается в случайно выбранное состояние.

д) изменение выходного сигнала: в случайно выбранном переходе случайным образом изменяется значение выходного сигнала на другое значение из заданного алфавита.

Эти пять операторов мутации способны генерировать множество разнообразных генотипов из родительских автоматов. Отметим, что первые три оператора, как правило, приводят к значительному изменению функционирования родительского автомата, в отличие от двух последних, которые производят меньшие изменения.

4) Оценка фитнес-функции: для каждого «мутанта» (вновь порожденного потомка) выполняется оценка значения фитнес-функции. Как отмечалось, вид фитнес-функции существенно зависит от задачи и ниже приведены некоторые наиболее распространенные типы.

5) Отбор родителей: для популяции с числом особей меньше десяти особи сортируются согласно их значениям фитнес-функции и лучшая половина особей используется для генерации следующего поколения. Для больших популяций обычно используется турнирный отбор родителей. Часто применяется попарное сравнение между множеством родителей $\{P_i\} \forall i \in \{1, \dots, m\}$ и потомков $\{P'_i\} \forall i \in \{1, \dots, m\}$. Последующий выбор особей производится с равной вероятностью из множества родителей и потомков. Потомок признается победителем, если он имеет ошибку прогнозирования не больше, чем у конкурирующей особи. При этом в качестве родителей для генерации следующего поколения отбирается m особей-победителей.

6) Процедура заканчивается при выполнении критерия останова, в противном случае наращивается номер поколения $k=k+1$ и

осуществляется переход на шаг 3 алгоритма эволюции. Для прогнозирования часто число поколений ограничивается сверху ($k=5$).

10.2. Применение эволюционного программирования в прогнозировании

Как уже отмечалось, в ранних работах ЭВ использовалось, в основном, при решении задач прогнозирования временных рядов, где целью эволюции популяции конечных автоматов является построение автомата, который способен предсказывать значение следующего входного сигнала. Рассмотрим суть задачи прогнозирования на примере автомата, представленного на рис. 10.3

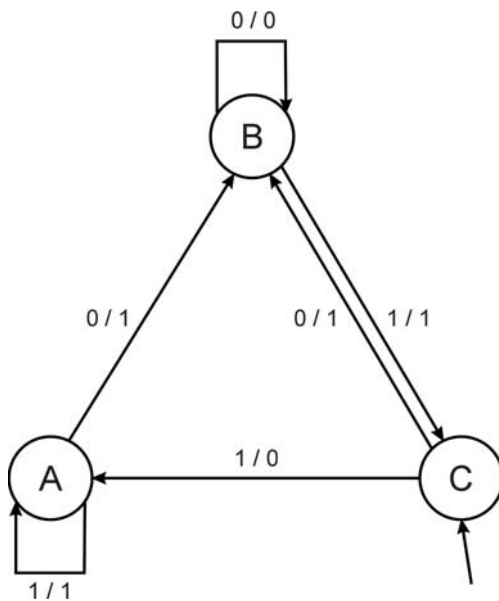


Рис.10.3. Пример автомата для задачи прогнозирования

Задача заключается в определении следующего символа входной последовательности. Допустим приведенный автомат рис.10.3 находится в начальном состоянии С, и на его вход подается последовательность $X=011101$. В этом случае согласно графу переходов-выходов автомат

выдает выходную последовательность $Y=110111$. Автомат правильно предсказывает если его выходной сигнал z_i равен следующему входному сигналу x_{i+1} . В случае равенства $z_i = x_{i+1}$ значение фитнес-функции увеличивается на 1. Для нашего примера автомат правильно предсказывает 3 из 5 символов входной последовательности. Иногда в фитнес-функцию вводится штраф за превышение заданного числа состояний автомата.

В качестве примера рассмотрим предсказание следующего символа периодической последовательности, один период которой представляется $(101110011101)^*$ [53]. Здесь входной и выходной алфавит конечных автоматов равен $\{0,1\}$. В качестве обучающего множества используется множество подпоследовательностей, содержащих первые 20 символов указанной циклической последовательности. Популяция случайно синтезированных конечных автоматов проходит пять поколений эволюции. В качестве фитнес-функции в данном случае используется средняя абсолютная ошибка (число неправильно предсказанных символов). После короткого периода эволюции в популяции определялась лучшая особь (с минимальной ошибкой), которая далее использовалась для предсказания следующего символа входной последовательности. Такая процедура повторялась 300 раз, и процесс эволюции в итоге состоял из 1500 поколений. Процедура ЭВ оценивалась при каждом прогнозе с использованием совокупной доли корректных прогнозов.

Эксперименты по прогнозированию выполнялись с популяциями, содержащими $m=\{3,5,10,0,100\}$ особей. Число мутаций выбиралось случайно из $M_i=\{1,2,3\}$. Было выполнено 30 попыток для каждой комбинации m и M_i . Число состояний автоматов варьировалось от 1 до 10. Во всех экспериментах максимальное число состояний ограничивалось сверху 10.

Согласно [53] получены следующие результаты экспериментов для задачи прогнозирования, которые представлены на рис.10.4. Здесь показано, как изменяется доля правильных прогнозов для лучшего автомата в популяции, усредненная по 30 попыткам для популяций с числом особей 3,5,10,50 и 100 и выполнением от 1 до 3 операторов мутации. Из графика видно, что при первом прогнозе число правильных прогнозов примерно равно 80% (после 30 попыток первого эксперимента). Далее следует небольшой спад, и затем постепенное повышение доли правильных прогнозов примерно до 90%.

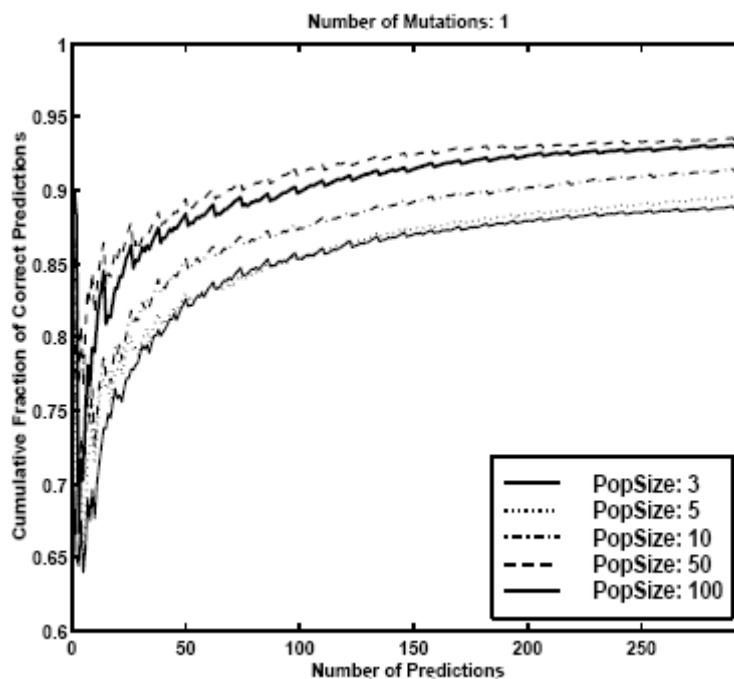


Рис.10.4. Изменение доли правильных прогнозов для различных мощностей популяций

10.3. Применение эволюционного программирования в задачах управления

Кроме задач прогнозирования ЭВ используется при решении задач управления [54]. В этом случае целью является построение в процессе эволюции автомата, который выдает на выход некоторое управляющее воздействие. В дальнейших экспериментах, для определенности, автомат должен выдать выходной сигнал, равный 5. Здесь особи популяции представляют модели целевого устройства управления, которые видоизменяются для того, чтобы получить необходимое выходное управляющее воздействие. Пусть для простоты алфавит автоматов состоит из целых чисел $\{1, 2, \dots, 9\}$. В первой серии экспериментов [54] в качестве целевого устройства использовался циклический автомат с 3-мя состояниями, представленный на рис.10.5. Во второй серии экспериментов применялись случайные автоматы с N состояниями.

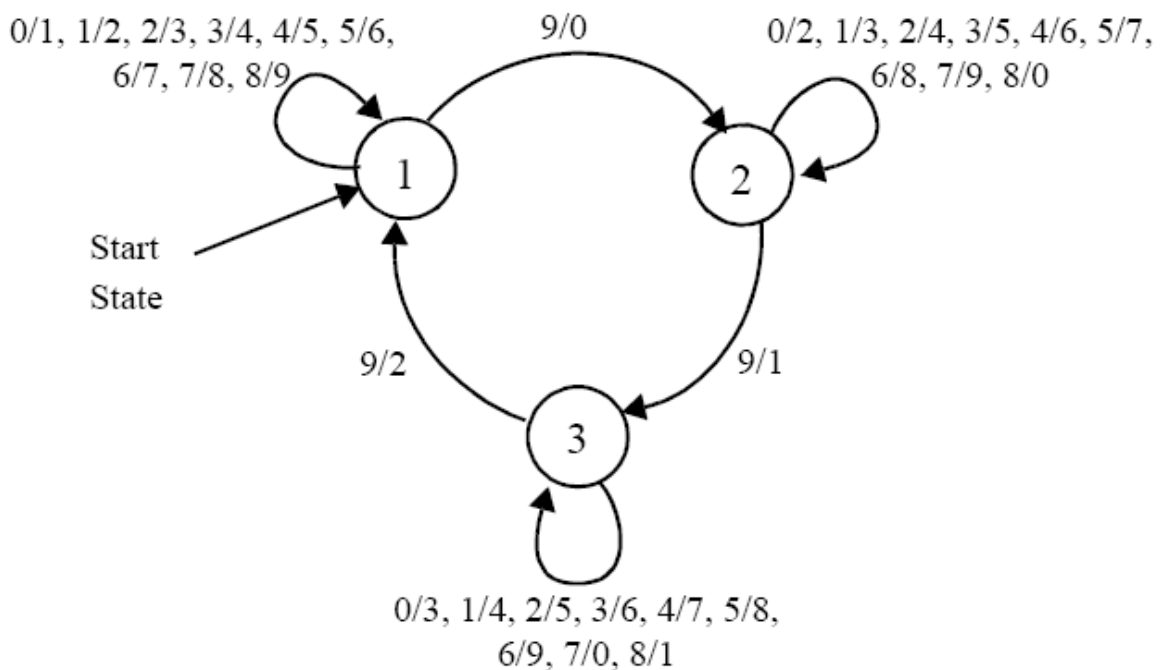


Рис.10.5. Циклический автомат с 3 состояниями

Как видно из рис.10.5, в качестве начального используется первое состояние автомата. Данный автомат, находясь в 1-м состоянии при подаче входного сигнала $i=\{1,...,8\}$ на выходе генерирует значение $i+1$ (по модулю 10). При подаче на вход 9 автомат выдает на выход 0 и переходит в следующее состояние $n+1$. Аналогично во втором состоянии входные символы 0,1,2,...,8 генерируют выходы 2,3,...,9,0 соответственно. И наконец, в последнем третьем состоянии входы 0,1,2,...,8 производят выходы 3,4,...,0,1. Аналогично может быть построен подобный циклический автомат для произвольного числа состояний. Отметим, что этот автомат полностью контролируем – то есть для каждого состояния существует входной сигнал, который выдает необходимое управляющее воздействие 5. С другой стороны, случайный автомат с N состояниями может быть либо частично контролируемым (для некоторых состояний существует входной сигнал, выдающий управляющее воздействие), либо неконтролируемым (ни для одного состояния не существует входного сигнала, которые генерирует управляющее воздействие).

10.4. Современные направления эволюционного программирования

В настоящее время известны многочисленные модификации ЭВ, где в качестве фитнес-функции используются не только среднеквадратичная ошибка, а более сложные функции. Кроме этого известны работы с большей глубиной прогнозирования (больше чем на один шаг). В середине восьмидесятых годов в ЭВ в качестве генома стали использоваться не только конечные автоматы, но и произвольные структуры данных [54]. Это позволяет с успехом применять ЭВ при численной оптимизации и решении задач комбинаторной оптимизации.

Более развитые версии ЭВ используют адаптивную мутацию. В настоящее время по некоторым источникам (например [55]), можно выделить три направления ЭВ: 1) стандартное ЭВ, 2) мета-ЭВ, 3) R-мета ЭВ, которые отличаются различным уровнем самоадаптации. По сути, это направление тесно переплетается с ЭС. Здесь широко используется представление потенциального решения в виде вектора действительных чисел, операторы мутации на основе сложения со случайными числами, подчиняющимися нормальному закону распределения, и операторы рекомбинации, которые рассмотрены в предыдущем разделе 9, посвященном ЭС.

С другой стороны, адаптивные автоматные модели достаточно широко используются в работах по мультиагентным системам («рой пчел», «колония муравьев» и т.п.), которые тесно примыкают к эволюционным вычислениям [56]).

10.5. Контрольные вопросы к разделу 10

1. Как представляется потенциальное решение в ЭВ?
2. Какой генетический оператор применяется в ЭВ?
3. Чем отличается эволюция на уровне фенотипа от эволюции на уровне генотипа ?
4. Какие виды мутации применяются в ЭВ?
5. Сформулируйте общий алгоритм решения задачи с использованием ЭВ.
6. Что общего между ГА и ЭВ?
7. Какие различия между ГА и ЭВ?
8. Что общего между ЭВ и ЭС ?
9. Как можно использовать ЭВ в прогнозировании ?
10. Какую фитнес-функцию можно использовать при прогнозировании?

11. Как можно использовать ЭВ при решении задач управления?
12. Какие формы генома используются в современных разделах ЭВ?
13. Какие генетические операторы используются в современных разделах ЭВ?

11. РЕАЛИЗАЦИЯ ЭВОЛЮЦИОННЫХ АЛГОРИТМОВ

При реализации эволюционных алгоритмов для решения прикладных задач используются два основных подхода:

- 1) программное – все компоненты ЭА реализуются программно на универсальных компьютерах;
- 2) аппаратное – (почти) все компоненты реализуются на аппаратном уровне.

Существуют различные реализации каждого из приведенных двух направлений.

11.1 Программная реализация

В настоящее время программная реализация эволюционных алгоритмов получила наибольшее распространение в силу чрезвычайно широкого распространения универсальных компьютеров. Программный пакет должен позволять выполнять следующие основные этапы при реализации решения задачи эволюционными методами:

- 1) выбор эволюционного алгоритма (ГА, ГП, ЭС, ЭВ);
- 2) выбор параметров ЭА (мощность популяции, вероятности кроссинговера и мутации и т.п.);
- 3) при необходимости (например, для символьной регрессии или машинного обучения) ввод и формирование обучающей выборки;
- 4) реализация необходимого эволюционного алгоритма решения прикладной задачи;
- 5) сохранение полученных результатов в необходимом формате;
- 6) формирование тестовых входных воздействий;
- 7) удобный контроль функционирования пакета.

Далее приводятся краткие сведения об основных программных пакетах, в которых реализованы эволюционные алгоритмы.

11.1.1 Пакет Evolver фирмы Palisade Corp

Данный пакет реализован в виде дополнения к MS Excel версий 5.0 и 7.0 [23]. Здесь оболочка Excel может быть использована в качестве средства описания исходных данных для решения прикладной задачи и расчетов в процессе функционирования алгоритма. При установке Evolver добавляет в Excel дополнительную панель инструментов для обеспечения доступа непосредственно к пакету. Исходные данные для решения прикладной задачи формируются стандартными средствами Excel. Особенностью является необходимость присутствия ячейки для задания формулы целевой (фитнесс-) функции. На рис. 11.1 показано основное окно пакета, в котором предусмотрены:

- ячейка для описания целевой функции;
- ячейка для описания хромосомы;
- ячейки для ввода ограничений (жестких и мягких) и диапазона возможных значений.

Кроме основного, предусмотрено окно для установки расположения исходных данных и их свойств (рис.11.2), к которым относятся:

- тип изменения значений генов в хромосоме;
- диапазон, тип (целые или действительные числа) и расположение в ячейках значений генов;
- позиция точки кроссинговера;
- значения вероятностей кроссинговера и мутации.

Тип оптимизации целевой функции – минимальное, максимальное или заданное значение

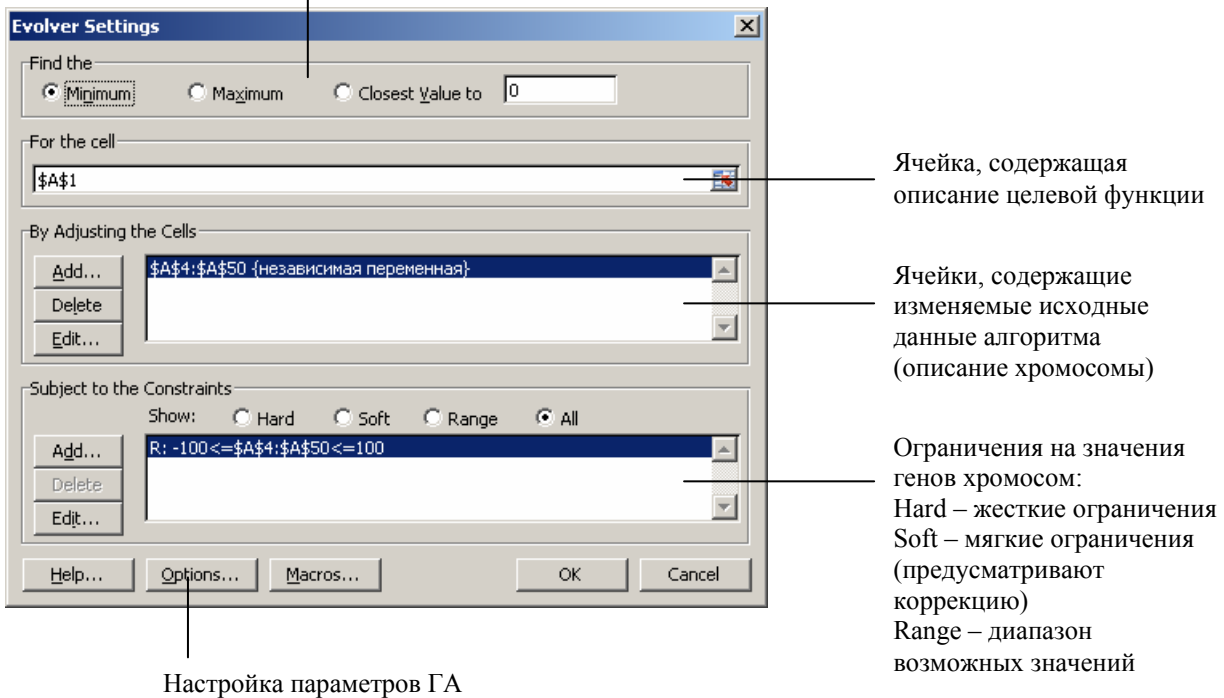


Рис. 11.1 Основное окно Evolver

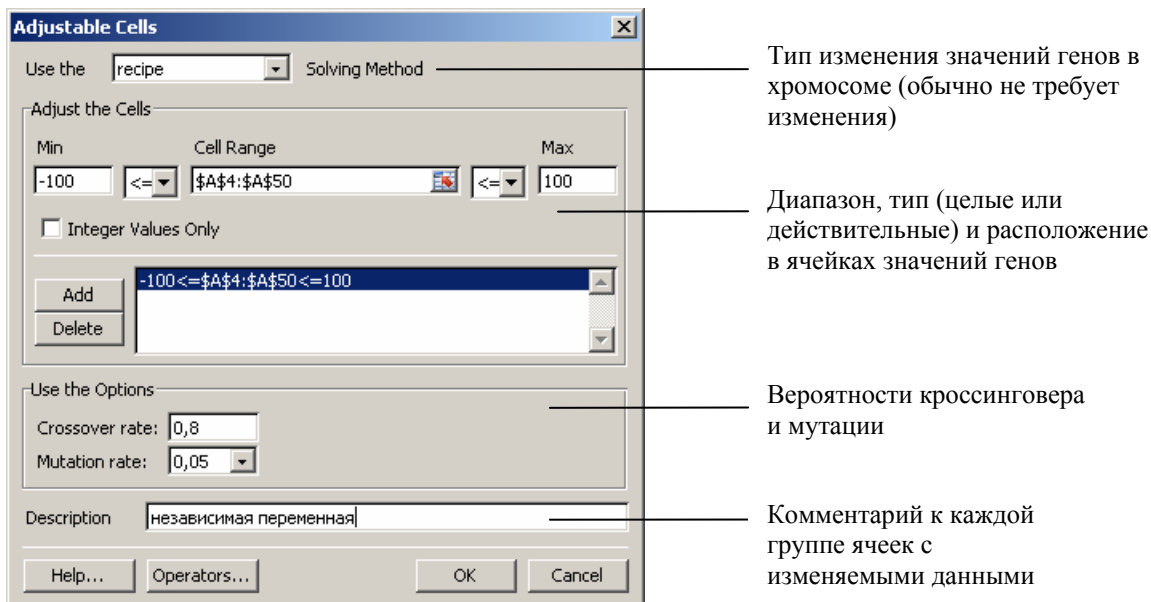


Рис.11.2 Окно установки расположения данных

Установка остальных параметров ГА выполняется также с помощью окна, представленного на рис.11.3, которое позволяет задавать:

- мощность популяции;
- способ формирования начальной популяции (случайный или детерминированный);
- критерии останова (заданное число поколений, времени, изменение лучшего решения на заданную величину на протяжении определенного числа поколений)
- другие параметры.

В случае необходимости к стандартному пакету поставляются средства расширенной обработки ГА которые позволяют разрабатывать собственные операторы репродукции, кроссинговера и мутации.

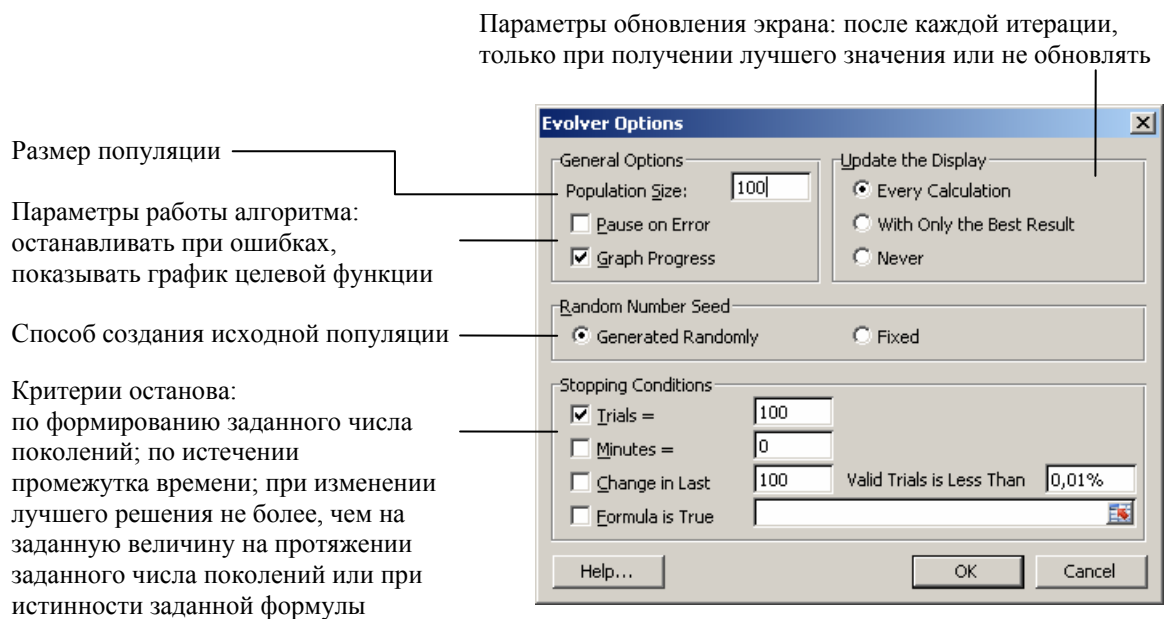


Рис.11.3 Окно установки параметров ГА

После ввода данных и необходимых параметров необходимо закрыть окно и запустить оптимизацию на панели инструментов. В процессе

поиска решения на экране отображается график изменения целевой (фитнесс-) функции, в строке состояния Excel значение лучшей в текущей популяции особи и номер поколения. По окончании работы полученные результаты по желанию пользователя формируются в файл отчета на отдельном листе Excel. Отметим, что пакет Evolver поставляется со справочной системой и обучающим курсом.

Данный пакет предназначен, в основном, для решения двух основных типов задач:

- 1) оптимизационных задач для функций вещественных переменных;
- 2) задач комбинаторной оптимизации.

В первом случае используется ГА с вещественным кодированием особи (см. раздел 1.3) и асинхронный ГА с частичной заменой популяции (раздел 4.6), где в каждый момент времени заменяется только одна особь. Отбор родителей производится преимущественно с помощью метода ранжирования (раздел 4.2.2). Скрещивание выполняется с помощью оператора однородного кроссинговера (раздел 4.3.1.3), где задается показатель скрещивания (crossover rate), который определяет процент генов, который потомок наследует от каждого родителя. Этот параметр вводится пользователем и представляется числом от 0.01 до 1.00. Например, показатель 0.7 означает, что потомок получит 70% генов от первого родителя, а остальные 30% заимствует от второго родителя. Отметим, что в вырожденном случае при показателе скрещивания 1.00 никакое скрещивание не выполняется, а производятся так называемые клоны – особи, идентичные родителям. По умолчанию в программе используется значение 0.5, которое дает возможность примерно одинакового количества генов от каждого родителя.

Кроме этого, задается показатель мутации из диапазона 0.00 до 1.00, который определяет процент генов, подвергающихся мутации. Если

показатель равен 0.00, то, очевидно, мутация не выполняется вовсе. Наоборот, при значении 1.00 мутация выполняется всегда.

При решении задач комбинаторной оптимизации используются генетические операторы такого же типа, что описаны в разделе 3 для задачи коммивояжера (в частности, упорядоченный кроссинговер).

11.1.2 Пакет GeneHunter фирмы Ward System Group

Этот пакет, также как и предыдущий, реализован в виде надстройки над MS Excel версий 5.0 и 7.0. Его запуск производится из меню сервис, что показано на рис. 11.4. [57]. Пакет русифицирован и имеет некоторых дополнительные настройки для ГА, например, включение стратегии элитизма и разнообразия.

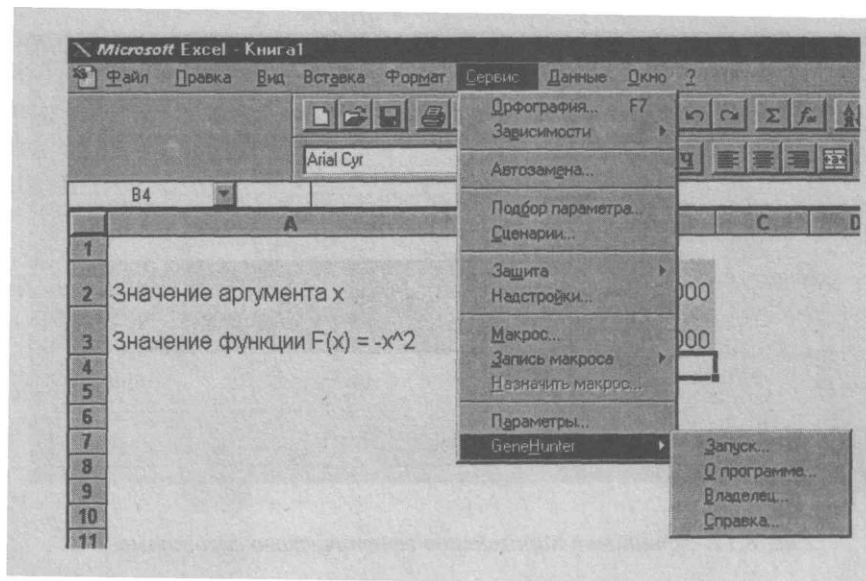


Рис.11.4 Запуск пакета GeneHunter

Окна этого пакета похожи на окна предыдущего, поскольку задается та же информация. Параметры ГА здесь определяются с помощью

соответствующей кнопки «Параметры», представленной на рис.11.5. Отметим, что здесь параметры не сохраняются автоматически в файле Excel. Для этого используется кнопка «Модель», после нажатия которой появляется необходимое окно. В рассмотренных пакетах по-разному реализовано задание параметров оператора кроссинговера. В пакете Evolver можно определять положение точки кроссинговера, но его вероятность автоматически принимает значение 1. Наоборот, в GeneHunter можно задать вероятность кроссинговера, но положение точки скрещивания определяется случайно по равномерному закону. В процессе поиска решения отображается номер текущего поколения и значение целевой функции его лучшей особи. Результаты работы помещаются на лист Excel. Интерфейс GeneHunter уступает по удобству Evolver, но превосходит по быстродействию. В комплект поставки входит несколько популярных приложений, включая реализацию численной оптимизации функций 1- и 2- переменных и задачу коммивояжера. Имеется также файл примеров в формате Excel решения прикладных задач: оптимизации функций, коммивояжера, управления пакетом ценных бумаг, обучения нейронных сетей и других, что конечно упрощает его освоение.

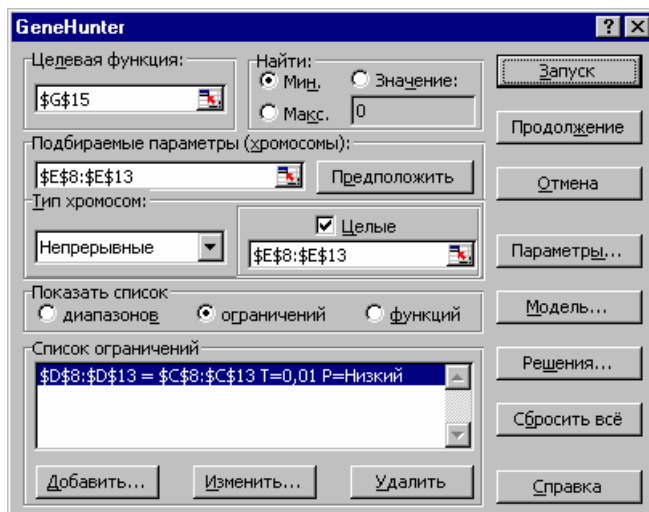


Рис.11.5 Главное окно GeneHunter

11.1.3 Пакет Genetic Training Option (GTO) фирмы California Scientific Software

Данный пакет является специализированным и представляет дополнительную утилиту, которая поставляется к известному нейросетевому пакету BrainMaker и применяется, в основном, для синтеза и обучения нейронных сетей. Пакет GTO имеет два основных режима работы:

- 1) вычисление основных характеристик искусственной нейронной сети – таких, как число слоев и нейронов при ее синтезе;
- 2) вычисление значений матрицы синаптических коэффициентов при обучении сети. Отметим, что тип активационных функций нейронов сети не подбирается. Этот пакет независимо от BrainMaker использоваться не может.

11.1.4 Программа FlexTool

Данная программа взаимодействует с пакетом **MATLAB** и запускается из окна этого пакета [23]. Для задания оптимизируемых функций используются m-файлы пакета **MATLAB**. В рабочем окне **MATLAB** можно просматривать значения различных переменных, которые используются программой **FlexTool** [23]. В данной программе реализованы следующие модификации ГА: классический (синхронный) ГА; асинхронный ГА с частичной заменой популяции; генетический

микроалгоритм (раздел 4.11), которые пользователь может выбрать для решения своей задачи. Кроме этого предоставляется возможность выбора метода отбора родителей (пропорциональный, ранговый или турнир), метода кодирования особи (двоичное или логарифмическое), число точек скрещивания в операторе кроссинговера. В программе предусмотрена одновременная оптимизация нескольких функций и создание для этого различных ниш. Для работы ГА, как обычно, необходимо задать его параметры: размерность популяции, вероятности кроссинговера и мутации, условия останова алгоритма и т.п.

11.1.5 MATLAB 7 – Genetic algorithm toolbox

Широко распространенный математический пакет MATLAB, начиная с версии 7.1, содержит инструментарий, позволяющий работать с ГА (Genetic Algorithm and Direct Search Toolbox). Запуск ГА производится из окна, представленного на рис.11.6.

Как видно из рисунка, при запуске задаются: фитнес-функция и число ее аргументов (переменных), характеристики популяции – тип кодирования особи, мощность популяции, ее инициализация, типы генетических операторов (репродукции, кроссинговера, мутации), критерий останова, список и форма выводимой в процессе поиска решения информации и другие параметры. Следует отметить, что при решении реальных сложных задач допускается импорт и экспорт информации в (из) пакет. Допускаются различные виды кодирования потенциального решения-особи как в виде двоичных строк, так и вектора реальных чисел. Предусмотрено масштабирование фитнес-функции, задание вероятностей кроссинговера и мутации, вид отбора родителей (в том числе, элитного, рулетки, однородного, турнирного).

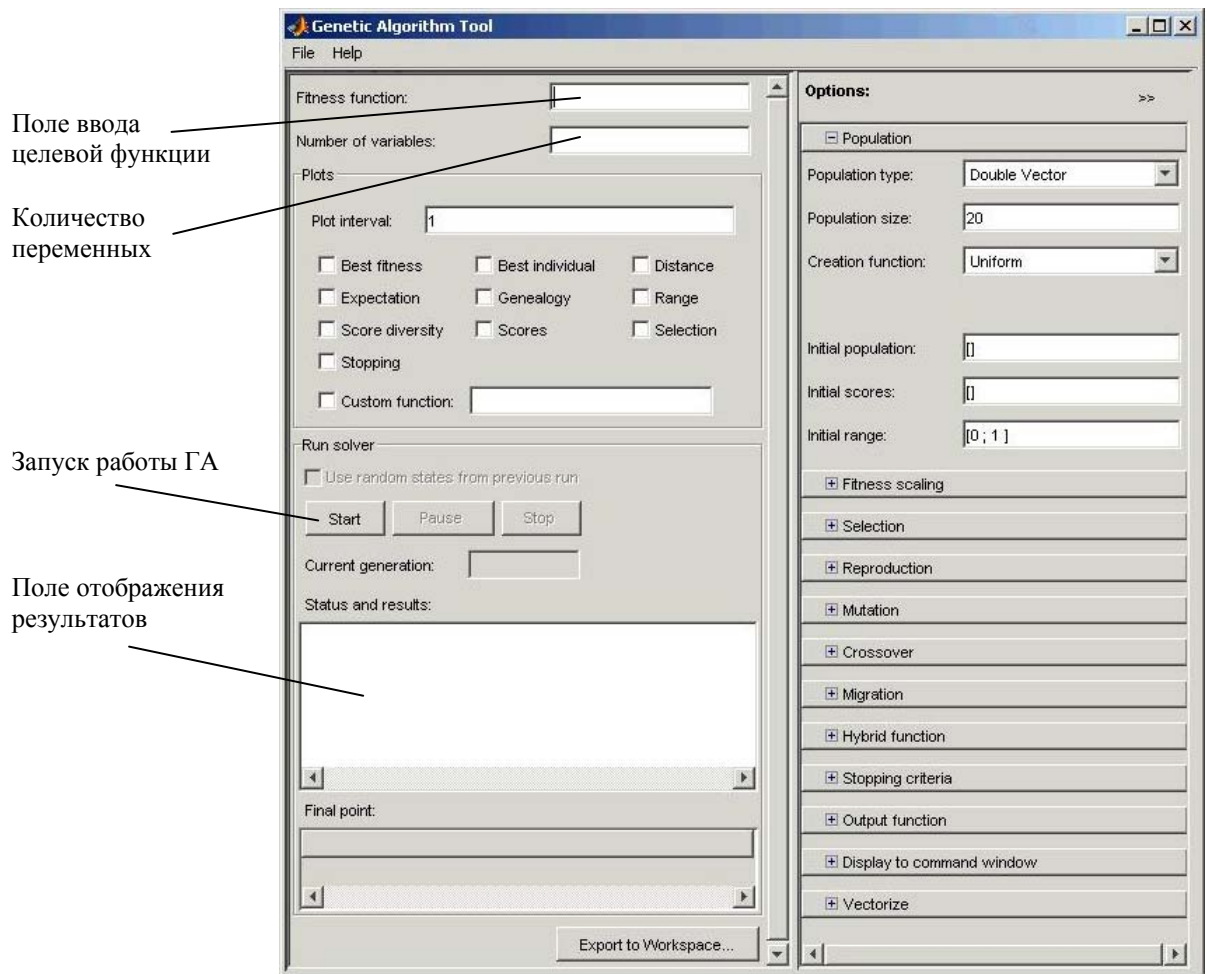


Рис.11.6 Окно запуска Genetic algorithm toolbox

Допускается использовать одноточечный и 2-точечный кроссинговер для двоичных особей. В случае вещественного кодирования особей есть возможность использовать промежуточный кроссинговер и некоторые другие (арифметические) операторы рекомбинации. Кроме ГА, допускается использование “гибридной функции”, когда после генетического поиска решения используется направленный поиск оптимального решения. При этом лучшее решение, найденное ГА, используется в качестве начального для последующего поиска. В данном Toolbox имеется достаточно много демонстрационных примеров, что облегчает его освоение. Данный пакет с успехом можно использовать при решении задач численной (в том числе и с ограничениями) и комбинаторной оптимизации.

11.1.6 Некоммерческое программное обеспечение

В заключение приведем некоторое некоммерческое программное обеспечение по генетическим алгоритмам, представленное в табл. 11.1.

Таблица 11.1

Название пакета	Язык программирования, операционная система	Характеристика пакета	Вид кодирования особи	ftp-Адрес
Genesis	C	Первый широко признанный пакет		ftp.aic.nrl.navy.mil/pub/galist/src/ga/genesis.tar.z
Evolution Machine	C/Dos	Пакет решает оптимизационные задачи, хорошо развиты графические средства	вещественный	ftp-bionic.fbio.tu-berlin.de:/pub/software/Evolution-Machine/
GAGA (GA for General Application)	C/Dos	Пакет минимизирует «тяжелые» целевые функции	-	ftp://cs.usl.fc.uk/darpa/gaga.shar
GANNET	C/UNIX	Пакет для эволюционного синтеза нейронных сетей	Двоичный	ftp:/famt.gmu.tdu:/gannet/source/
Genitor	C/UNIX	Решает задачи комбинаторной	Вещественный, двоичный	ftp:/cs.colostate.edu:/pub/GENITO

		оптимизации на основе параллельного ГА		R.tar
GENOCOP	C/UNIX	Выполняет численную оптимизацию с линейными ограничениями	Вещественный	ftp.uncc.edu:/ cor/evol/genoc op2.tar.z
GENlib	C/Dos,UNIX	Многоцелевой пакет, включая синтез нейронных сетей	Вещественный , двоичный	ftp.neuro.informa tic.uni- kassel.de/pub/Ntu ralNets/GA-and- NN/
PGAPack	Fortran и C	Библиотека модулей для параллельного ГА	Любой тип	http://www.m cs.anl.gov/pga pack.html
WOLF?	C/UNIX/Sun	Строит модели на основе сплайнов	Вещественный	David Rogers, Tmail:drogers@ msi.com
GALOPPS 3.2	C/Dos,UNIX	Реализованы различные модели параллельного ГА	Вещественный	http://isl.msu.edu/ GA/software/soft ware-index.html

11.2 Аппаратная реализация генетических алгоритмов

Работы по аппаратной реализации ГА ведутся достаточно давно [58,59], поскольку время решение реальных задач высокой (и даже средней) размерности эволюционными методами требует значительных временных ресурсов. Но ГА общего назначения обычно содержат

некоторые изменяемые модули (например, вид функции, которую необходимо оптимизировать), что создает проблемы при аппаратной реализации. Появление новых технологий, в первую очередь FPGA, которые позволяют перепрограммировать аппаратную часть, в значительной степени способствовало активизации работ в этом направлении. Основой эффективной аппаратной реализации ГА является его разбиение на части, которые реализуются аппаратно отдельными модулями и выполняются параллельно. Основными приемами, которые позволяют повысить эффективность аппаратной реализации ГА, являются параллелизация и конвейеризация выполнения отдельных модулей. При этом используются различные архитектуры и технологии, некоторые из которых представлены ниже.

11.2.1 Аппаратная реализация классического генетического алгоритма

Специализированный процессор, реализующий ГА на аппаратном уровне должен удовлетворять следующим требованиям.

- 1) Главной целью аппаратной реализации является существенное сокращение процессорного времени решения задачи по сравнению с программной реализацией.
- 2) Не менее важной является стоимость аппаратной реализации ГА спецпроцессора. Возможна архитектура с использованием нескольких FPGA, в которой параллельно выполняются различные генетические операторы (отдельный оператор реализуется в своей FPGA), что повышает быстродействие, но ведет к большей стоимости.

- 3) Спецпроцессор должен легко реконфигурироваться (настраиваться) для решения различных задач в процессе синтеза. Это дает возможность использовать один и тот же RTL код (на языке проектирования аппаратуры) для многих приложений. В RTL коде это можно выполнить путем настройки параметров в процессе синтеза, которые изменяются в зависимости от приложений.
- 4) Проект должен быть модульным. Это означает, он должен быть разбит на меньшие модули, каждый из которых проектируется для выполнения хорошо определенной и также «разделяемой» функции. Это делает проект более понятным, облегчает его отладку и помогает использовать технологию «design-reuse».
- 5) Проект должен иметь управляющие регистры с простым интерфейсом (запись-чтение) для изменения операционных параметров. К таким изменяемым параметрам относятся параметры общего назначения, например, мощность популяции, счетчик поколений, вероятности кроссинговера и мутации с одной стороны, и, с другой стороны, проблемно ориентированные, такие как, например, число контактов и элементов в схеме при решении задачи разбиения, которая рассматривается далее в качестве примера. Обычно проблемно-ориентированными являются 16-разрядные регистры, что позволяет обрабатывать для указанной задачи достаточно большие схемы (с числом контактов или элементов не более 65536). Как правило, остальные регистры могут быть 8-разрядными. При этом мощность популяции не должна превышать 256, вероятности кроссинговера и мутации могут изменяться от 0 до 255/256 с шагом 1/256.
- 6) Для хранения данных (например, машинного описания схемы, промежуточной популяции) должны использоваться модули памяти. Они могут быть либо внешними относительно ядра процессора, либо

внутренними, которые реализуются в виде RAM (ОЗУ) и доступны FPGA.

- 7) Концепция этой архитектуры должна допускать расширение программно-аппаратного проектирования с использованием ARM процессора с FPGA модулем.
- 8) Вычисление значений фитнес-функции занимает большую часть процессорного времени, поэтому оно должно быть реализовано аппаратно.

Далее мы рассмотрим аппаратную реализация ГА на одной FPGA на примере ГА спецпроцессора для решения задачи разбиения (размещения) схемы (circuit-partitioning problem) [60], которая является одной из самых сложных в проектировании цифровых схем и NP- полной (т.е. не имеет алгоритма решения полиномиальной сложности). Архитектура ГА спецпроцессора определяется в результате идентификации основных шагов (этапов) ГА и отображения их в аппаратуру, что показано на рис.11.7. Рассматриваемая архитектура для реализации ГА использует конвейерную обработку при обработке наиболее сложных (и ресурсоемких) частей (этапов) алгоритма. Здесь используется общий (главный) контроллер, который генерирует управляющие сигналы для остальных блоков. Отдельные блоки применяются для выполнения операторов отбора, кроссинговера и мутации, вычисления значений фитнес-функции. В отличие от ранних архитектур, в данном проекте значения фитнес-функции вычисляются аппаратно.

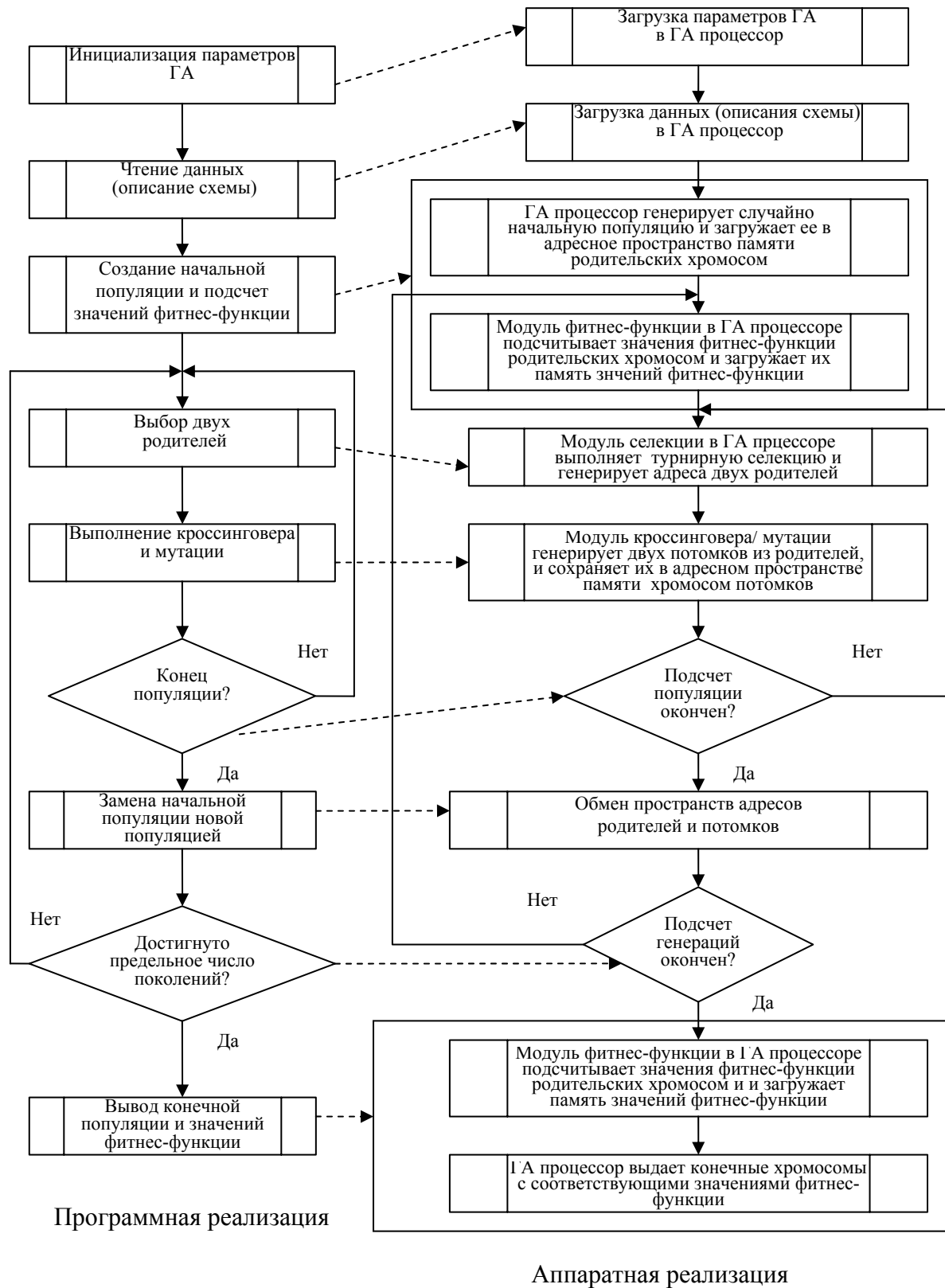


Рис.11.7 Сравнение программной и аппаратной реализации ГА

Внутренняя архитектура ГА спецпроцессора представлена на рис.11.8.

Здесь модуль отбора (selection) выбирает родителей с хорошими значениями фитнес-функции из фитнес-памяти и посылает адреса выбранных родителей в модули кроссинговера и мутации. Модули кроссинговера и мутации выполняют соответствующие операции над родительскими особями. Фитнес-модуль генерирует значения фитнес-функции для каждой производимой хромосомы. Главный контроллер генерирует управляющие сигналы для всех остальных блоков.

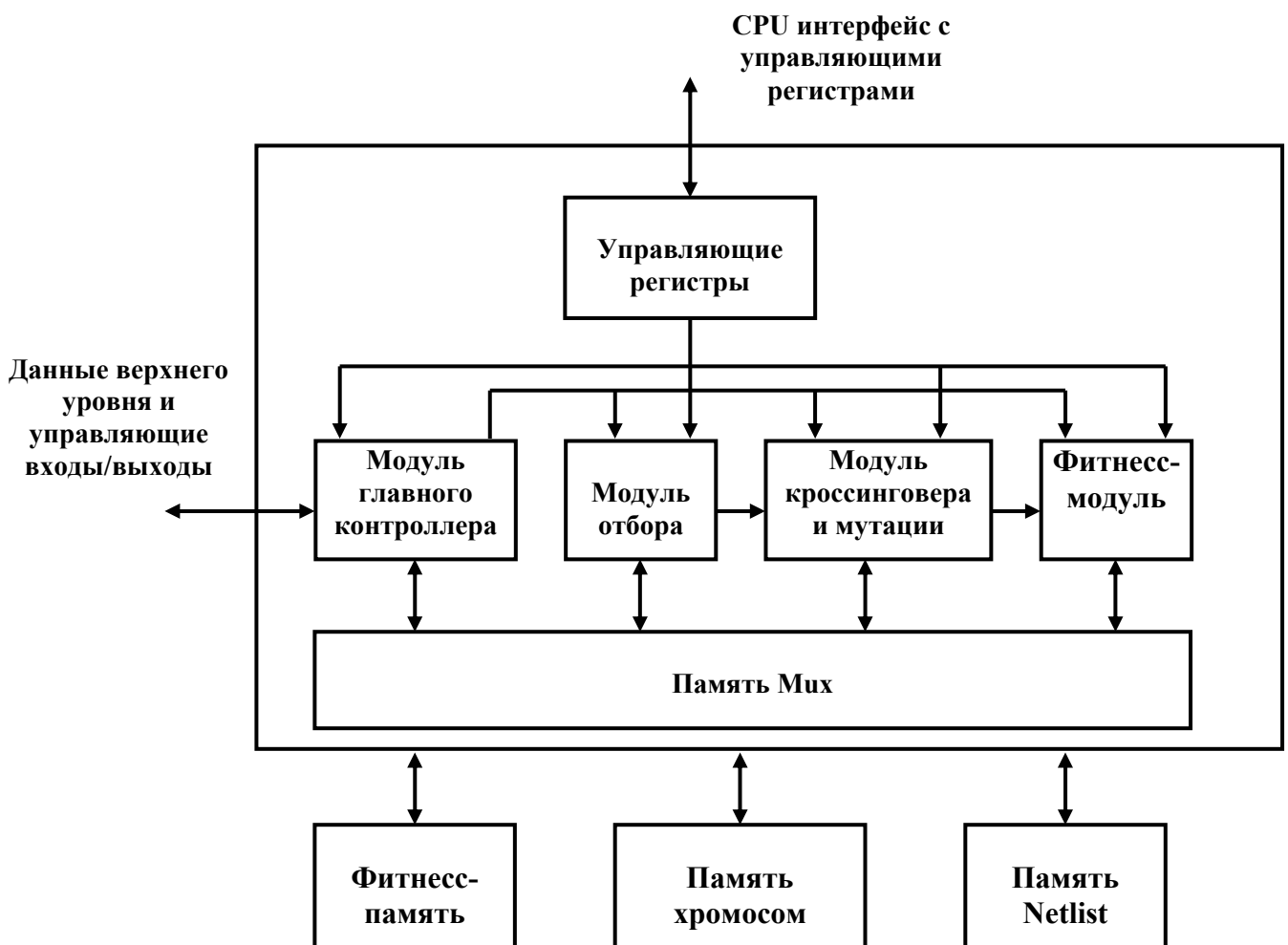


Рис.11.8 Архитектура спецпроцессора ГА

После получения сигнала “старт” ядро процессора (core) принимает описание схемы (в виде списка связей элементов), который запоминается в памяти связей, из которой схема многократно считывается в ядро для вычисления значений фитнес-функции. Далее ядро случайно генерирует начальную популяцию и запоминает ее в памяти хромосом. Модуль отбора использует турнирный отбор для выбора двух родителей. Их адреса (в памяти хромосом) используются модулями кроссинговера и мутации для выполнения генетических операторов. Модули кроссинговера и мутации запоминают двух произведенных потомков в память хромосом. После генерации новой популяции фитнес-модуль вычисляет значение фитнес-функции для каждой особи популяции и запоминает их в фитнес-памяти. Новая популяция заменяет родительскую, но лучшая особь из родительской популяции переносится в следующее поколение. После выполнения заданного числа поколений (которое хранится в управляющем регистре) ядро выдает конечную популяцию со значениями фитнес-функции для каждой особи.

Общее описание взаимодействия ГА спецпроцессора с базовым компьютером (host) показано на рис.11.9. Значения управляющих регистров и список связей схемы загружается из базового компьютера и запоминается в управляющих регистрах и памяти связей. Рассмотрим детальней основные модули ГА спецпроцессора.

Модуль отбора после приема активного управляющего сигнала из главного контроллера выполняет следующие функции.

- 1) Генерирует четыре случайных адреса для фитнес-памяти и читает из этой памяти четыре значения фитнес-функции. При этом используется генератор случайных чисел, реализованный на линейном регистре сдвига (LFSR).

- 2) Выполняет сравнение двух пар значений фитнес-функции и выбирает лучшие из каждой пары.
- 3) Адреса двух лучших хромосом выставляются на внешние выходы модуля в качестве адресов родительских особей, которые используются в модуле кроссинговера и мутации.
- 4) По завершению операции модуль генерирует сигнал готовности.

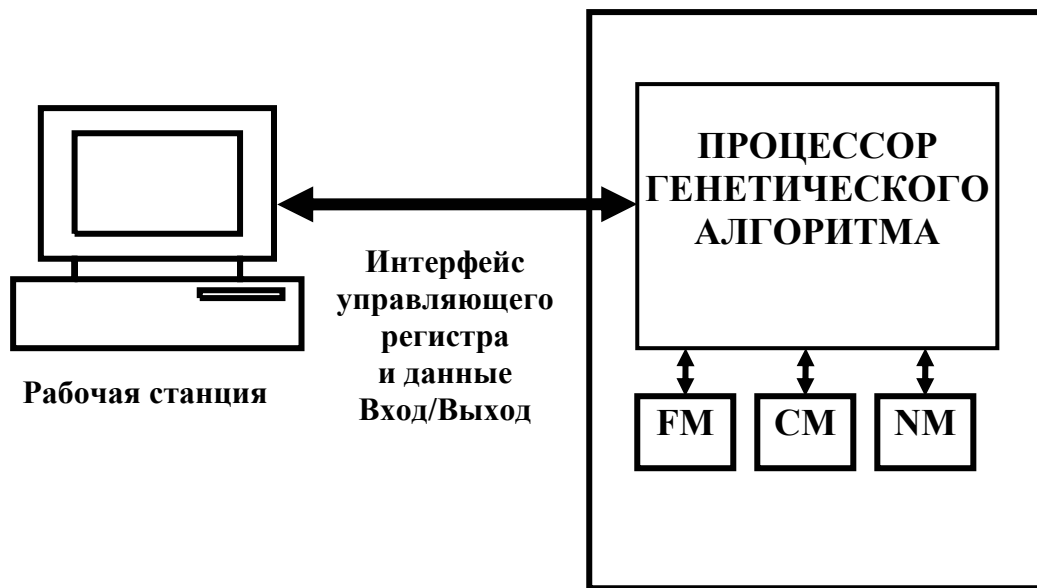


Рис.11.9 Взаимодействие спецпроцессора с базовым компьютером

Таким образом, при аппаратной реализации модуль отбора содержит генератор случайных чисел, компаратор, ряд регистров для хранения адресов и управляющий автомат, который генерирует управляющие сигналы для различных блоков модуля.

Модуль кроссинговера и мутации выполняет операторы кроссинговера и мутации над двумя родительскими хромосомами, адреса которых генерируются модулем отбора. Память для хромосом разбита на две части. Родительская популяция хранится в одной из них, а потомки

соответственно в другой. После приема активного управляющего сигнала из главного контроллера модуль выполняет следующие функции.

- 1) Для каждой родительской особи читается одно слово из памяти хромосом согласно адресам, генерируемым модулем отбора. После чтения слова счетчик хромосом наращивается.
- 2) Генерируется случайно маска кроссинговера для каждого слова – родительской особи. Вероятности кроссинговера и мутации, которые хранятся в управляющих регистрах, сравниваются с генерируемым (в этом модуле) 8-разрядным случайным числом. Если генерируемое случайное число меньше хранимого значения, то соответствующий оператор выполняется, иначе родительская особь просто копируется в качестве потомка. Для мутации генерируется случайное число, которое определяет случайный бит (номер разряда) в хромосоме потомка. Значение этого бита инвертируется в зависимости от числа единиц и нулей в хромосоме. Результаты выполнения кроссинговера и мутации слово за словом запоминаются в памяти хромосом.
- 3) Шаги 1 и 2 повторяются, пока счетчик хромосом не достигает заданной величины, которая хранится в управляющем регистре. По завершению операции модуль генерирует сигнал готовности в главный контроллер.

Итак, внутренняя архитектура модуля должна включать счетчик хромосом, несложную комбинационную схему для выполнения кроссинговера и мутации и генератор случайных чисел. Этот же генератор используется для генерации масок однородного кроссинговера.

Как только новая популяция полностью генерируется модулем кроссинговера и мутации, *фитнесс-модуль* вычисляет значения фитнес-функции для каждой ее особи. Этот процесс повторяется для каждого

слова памяти хромосом, пока значение счетчика хромосом не достигнет заданного значения – мощности популяции.

Результирующее значение фитнес-функции модифицируется с учетом числа элементов в разбиении схемы. Поскольку данный ГА спецпроцессор предназначен для решения задачи разбиения (размещения) в САПР, то он является проблемно-ориентированным и определяет качество (значение фитнес-функции) рассматриваемого разбиения, которое представляется особью популяции и выполняет следующие функции.

- 1) Для каждой схемы определяет – генерирует ли текущая хромосома сечение. Для каждой хромосомы аккумулятор фитнес-значения сбрасывается в нуль. Хромосома и схема считываются слово за словом из памяти хромосом и соединений схемы соответственно.
- 2) Для пары машинных слов, представляющих хромосому и схему выполняется поразрядное логическое И с последующим поразрядным ИЛИ. В используемом кодировании хромосомы каждое сечение представляется 1 в соответствующем разряде, поэтому при наличии 1 аккумулятор значения фитнес-функции увеличивается на 1.
- 3) Этот процесс повторяется для каждого слова памяти хромосом пока значение счетчика хромосом не достигнет заданного значения – мощности популяции.
- 4) Результирующее значение фитнес-функции модифицируется с учетом числа элементов в разбиении схемы. Это выполняется путем подсчета единиц в словах памяти хромосом.
- 5) По достижению значения счетчика мощности популяции модуль генерирует сигнал готовности в главный контроллер.

Итак, при аппаратной реализации фитнес-модуль содержит счетчик слов, счетчик схем, счетчик особей в популяции, фитнес-аккумулятор, и управляющий автомат, который генерирует управляющие сигналы в эти блоки. Кроме этого, используется регистр признаков для каждого разбиения, который показывает наличие хромосомы в данном разбиении.

Модуль главного контроллера генерирует управляющие сигналы для остальных модулей и выполняет следующие функции.

- 1) После приема активного сигнала «старт ГА» главный контроллер запускает чтение описания схемы (список связей).
- 2) После чтения схемы в память связей главный контроллер генерирует случайные хромосомы и инициализирует память хромосом начальной случайной популяцией.
- 3) Далее выполняется основной цикл ГА, в котором последовательно выполняются вычисление значений фитнес-функции, отбор хромосом, кроссинговер и мутация в соответствующих модулях, пока счетчик поколений не достигнет заданного числа, хранимого в регистре поколений. В каждом поколении счетчик поколений наращивается.
- 4) В конце последнего поколения главный контроллер вызывает фитнес-модуль последний раз и выводит конечную популяцию с окончательными значениями фитнес-функции для каждой особи.

Данный ГА спецпроцессор спроектирован на VHDL и его характеристики проверены путем моделирования. Данные представленные в [60] показывают, что аппаратная реализация позволяет ускорить решение задачи (в данном случае размещения схемы) на тестовых задачах (benchmarks) примерно в 100 раз. Хотя архитектура разработана для решения задачи размещения, основные модули могут быть с успехом

использованы и при решении других задач. Это легко сделать путем модификации соответствующих блоков кода VHDL.

11.2.2 Аппаратная реализация компактного ГА.

Заметим, что псевдокод компактного ГА, представленный в разделе 6 содержит основные следующие операции: сложение, вычитание и сравнение. Значение вероятности каждого разряда $p[i]$ можно изменять параллельно. Кроме этого, в компактном ГА операции частично перекрываются, поэтому применение при аппаратной реализации конвейерной архитектуры существенно улучшает его характеристики. Рассматриваемая аппаратная реализация компактного ГА [61] представлена на рис.11.10 и содержит 5 основных модулей: генератор случайных чисел, регистр вероятностей, компаратор, буфер и модуль оценки фитнес-функции. Рассмотрим каждый из них детальнее.

Генератор случайных чисел реализуется на основе одномерного клеточного автомата (КА) с двумя состояниями. Увеличение размерности КА дает лучшие характеристики, однако 8-разрядного КА достаточно для практических целей. Согласно рис.11.10 каждый разряд генерируется индивидуально случайно и параллельно. Поэтому число генераторов равно числу разрядов хромосомы.

В регистре вероятностей вероятность $p[i]$ (которая, в общем случае представляется числом с плавающей запятой) может быть заменена представлением целым числом, поскольку операции над $p[i]$ сводятся к сложению и вычитанию с $1/n$. В этом случае при $n=256$ для представления $p[i]$ достаточно 8-разрядного целого числа, а операции сложения и

вычитания заменяются инкрементом и декрементом. По этой же причине n должно быть степенью 2.

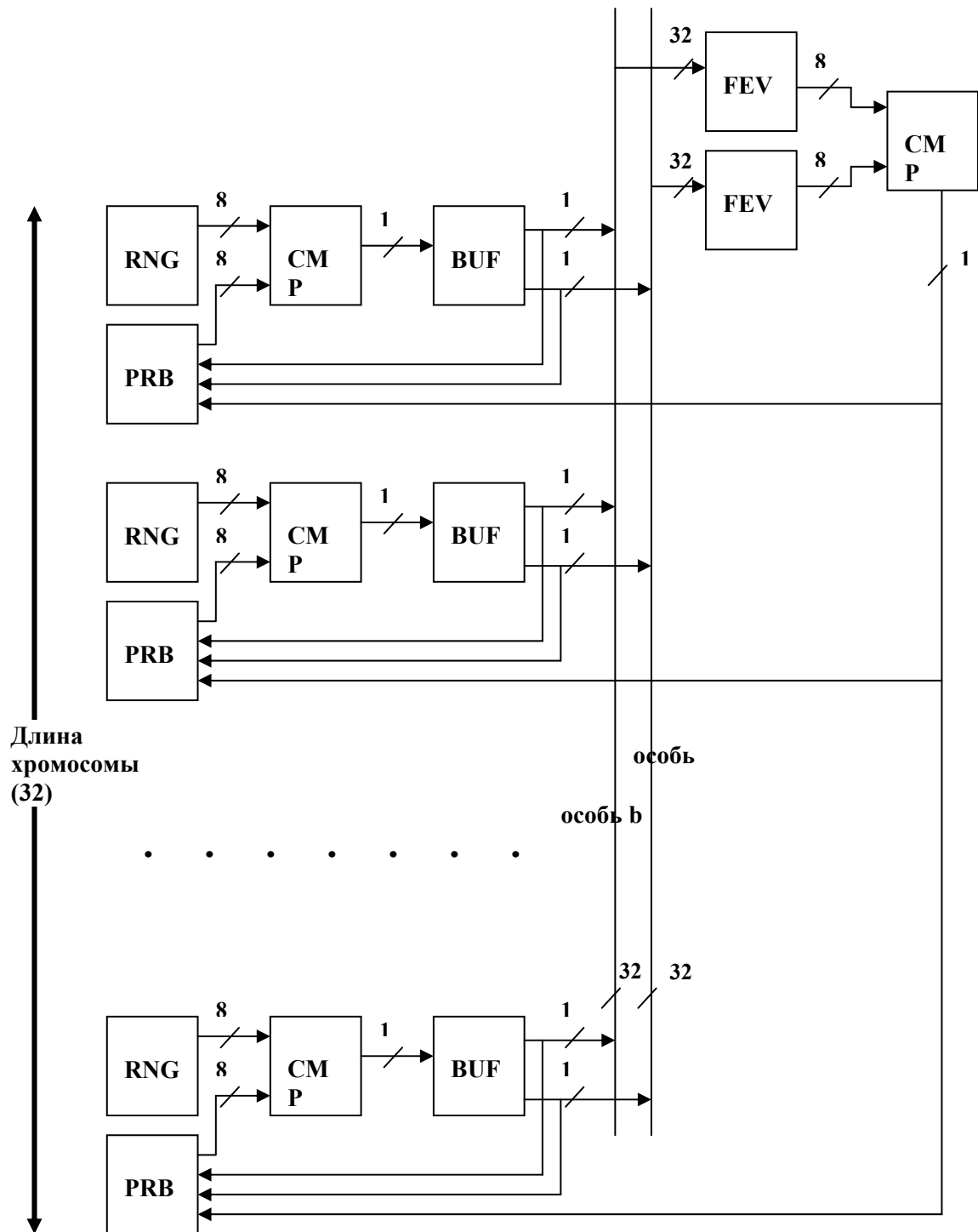


Рис.11.10 Аппаратная реализация компактного генетического алгоритма (размер популяции = 256, длина хромосомы = 32)

Компаратор является комбинационной схемой, которая сравнивает два числа m и n . Если $m \geq n$, то выход схемы должен быть равен 1. В противном случае выход равен 0.

Буфер является последовательностной схемой, в которой содержатся значения каждого i -го разряда особей a и b . Каждый буфер должен хранить особи, пока выполняется оценка значений фитнес-функции.

Модуль оценки фитнес-функции должен выполнять параллельно оценку двух особей a и b . В [15] модуль реализован для решения задачи максимизации числа 1 в хромосоме (one-max problem) и сводится к подсчету числа единичных разрядов в двоичном числе. Число требуемых циклов естественно зависит от решаемой задачи (для задач оптимизации может быть достаточно большим).

Итак, спецпроцессор – компактный ГА должен выполнять операции над вектором вероятностей p . Для каждого гена $p[i]$ обрабатывается параллельно. Генератор случайных чисел, регистр вероятностей и компаратор используются для генерации двух особей и сохранения их в буфере. Компаратор определяет победителя и побежденного, и, в соответствии с результатом, корректирует вектор вероятностей p .

При аппаратной реализации по получению сигнала сброса, генераторы устанавливаются в определенные значения (в векторе вероятностей для каждого гена устанавливается значение 0.5), буфера сбрасываются в начальное состояние. Далее повторяются последовательно следующие шаги до тех пор, пока для каждого гена в векторе вероятностей не получим значения 0 или 1.

- 1) Результат оценки фитнес-функции определяет, какую операцию (инкремент или декремент) надо выполнить над регистром вероятностей. Далее выполняется сравнение сгенерированных случайных чисел и регистра вероятностей.

- 2) В буферах запоминаются результат сравнения. Если случайное число больше чем $p[i]$, то i -ый разряд особи a устанавливается в 0. В противном случае – в 1. Пока выполняется синхронизация буферов, одновременно генерируются новые случайные числа.
- 3) В буферах выполняется эта же операция для особи b . На этом шаге особи передаются в фитнес-модуль. Результаты сравнения используются для корректировки регистра вероятностей на шаге 1.

Очевидно, что каждый шаг может быть выполнен за один такт. Поэтому спецпроцессор компактный ГА выполняет одно поколение в эволюции за 3 такта (для one-max problem). Для более сложных задач для одного поколения необходимо $3+e$ тактов, где e – число тактов, необходимых для оценки значения фитнес-функции.

Согласно данным [61] аппаратная реализация компактного ГА содержит 15000 вентилях и позволяет увеличить быстродействие в 1000 раз (для one-max problem). Таким образом, компактный ГА допускает более эффективную аппаратную реализацию по сравнению с классическим ГА, но сам метод для некоторых задач может давать результаты хуже.

11. Контрольные вопросы к разделу 11

1. Какие виды реализации эволюционных алгоритмов существуют?
2. Чем отличается программная реализация ГА от аппаратной?
3. Что должен позволять программный пакет, реализующий эволюционный алгоритм?
4. Приведите основные программные пакеты, реализующие ГА.
5. В какой среде работают пакеты Evolver и GeneHunter?
6. В какой среде работает пакет FlexTol?
7. Каковы преимущества аппаратной реализации ГА?

8. Приведите пример аппаратной реализации простого ГА и опишите его основные модули.
9. Приведите пример аппаратной реализации компактного ГА и опишите его основные модули.

ЛИТЕРАТУРА

1. Holland J.P. Adaptation in Natural and Artificial Systems. An Introductory Analysis With Application to Biology? Control and Artificial Intelligence. University of Michigan, 1975, 210 p.
2. Goldberg D.E. Genetic Algorithms in Search, Optimization and Machine Learning. - Addison-Wesley, Reading, MA. - 1989.
3. Rechenberg I. Evolutionstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Stuttgart: Frommann-Holzboog, 1973.
4. Fogel L.J., Owens A.J., Walsh M.J. Artificial Intelligence through Simulated Evolution. Wiley, New York, 1966.
5. Koza J.R. Genetic Programming. Cambridge: MA: MIT Press, 1992.
6. Ивахненко А.Г. Самообучающиеся системы распознавания и автоматического управления. Киев: Техника, 1969.
7. Цыпкин Я.З. Основы теории обучающихся систем. М.: Наука. - 1970.
8. Расстригин Л.А. Статистические методы поиска. М.: Наука. - 1968.
9. Букатова И.Л. Эволюционное моделирование и его приложения. М.: Наука, 1979.
10. Букатова И.Л., Михеев Ю.И., Шаров А.М. Эволюционная информатика: теория и практика эволюционного моделирования. - М.: Наука. - 1991. - 206 с.
11. В.М. Курейчик. Генетические алгоритмы. - Тагарог: Изд-во ТРТУ, 1998.
12. Л.А. Гладков, В.В. Курейчик, В.М. Курейчик. Генетические алгоритмы. - М.: Физматлит. - 2006. - 319 с.
13. Батищев Д.И. Генетические алгоритмы решения экстремальных задач. Воронеж: Воронежский технический университет. - 1995. - 65 с.

- 14.Верлань А.Ф., Дмитренко В.Д., Корсунов Н.И., Шорох В.А. Эволюционные методы компьютерного моделирования.Киив: Наукова Думка.-1992.-256с.
15. Скурихин А.Н.Генетические алгоритмы. Новости искусственного интеллекта.- N.4., 1995.- С. 6-17.
- 16.Вороновский Г.К.,Махотило К.В.Петрашев С.Н.,Сергеев С.Л. Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности.Харьков:Основа,1997,112с.
- 17.Z.Michalevich. Genetic Algorithms + data structures=Evolution Programs. Springer.-1999.
- 18.M.Mitchel. An introduction to genetic algorithms.The MIT Press, Cambridge, Massachusetts.-1998.-210 p.
- 19.Vose M.D.,Liepins G.E.Punctuated equilibria in genetic search.Complex Systems,1991.-5.-p.31-44.
20. Nix A.E.,Vose M.D. Modelling genetic algorithms with Markov chains.Annals of mathematics and artificial intelligence,1991.-5.-p.79-88.
21. Шапиро Д.,Рэттрэй М., Прюнель-Беннет А. Применение методов статистической механіки для изучения динамики генетического алгоритма.//Обозрение прикладной и промышленной математики.- Москва:ТВП.-1996ю-Т.3, вып.5, серия «Методы оптимизации».- с.670-687.
22. Скобцов Ю.А., Скобцов В.Ю. Современные модификации и обобщения генетических алгоритмов//Таврический вестник компьютерных наук и математики.Симферополь: 2004, N1. – с.60-71.
23. Рутковская Д.,Пилинский М.,Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы.-М:Горячая линия, 2004.- 452с.

- 24.L. M.Patnaik,S.Mandavilli.Adaptation in genetic algorithms. In: Genetic algorithms for pattern recognition.-CRC Press.-1996.-P.45-64.
- 25.Beasley D.,Bull D.R., Martin R.R. A sequential niche technique for multimodal function optimization//Evolutionary computation,1993.- Vol.1,N2.-pp.101-125.
- 26.Mahfoud S.W. A comparison of parallel and sequential niching methods.//Proceedings of VI International conference on genetic algorithms.-1995.-p.136-143.
27. W.Banzhaf, P.Nordin, R.E.Keller, F.D.Francone. Genetic programming: introduction. Morgan Kaufman, Inc. Sun Francisco, USA, 1998.
28. Ю.А.Скобцов, А.И.Эль-Хатиб.Параллельные генетические алгоритмы.Наукові праці Донецького національного технічного університету серія:“Обчислювальна техніка та автоматизація”, Випуск 90.-Донецьк : ДонНТУ. – 2005.-с.137-144.
- 29.E.Alba, J.M.Troya.. A survey of parallel distributed genetic algorithms//Complexity,vol.4,no.4,pp.31-52,1999.
- 30.E.Alba, M.Tomassini. Parallelism and evolutionary algorithms//IEEE trans. On evolutionary computation.-2002.-vol.6.N5.-pp.443-462.
- 31.Ю.А.Скобцов, В.Ю.Скобцов. Логическое моделирование и тестирование цифровых устройств.-Донецк:ИПММ НАНУ, ДонНТУ, 2005.-436с..
- 32.Ari Juels, Shumeet Baluja, Alistair Sinclair. The Equilibrium Genetic Algorithm and the Role of Crossover. – 1993.
- 33.Shumeet Baluja. Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning (Tech. Rep. No. CMU-CS-94-163). Pittsburgh, PA: Carnegie Mellon University. – 1994.
- 34.G. Harik, F. G. Lobo, D. E. Goldberg. The Compact Genetic

- Algorithm. // IEEE Trans. Evolutionary Computation. – 1999. - vol. 3. – pp. 287-297.
- 35.F. Corno, M. Sonza Reorda, G. Squillero. The Selfish Gene Algorithm: a new Evolutionary Optimization Strategy. Proceedings of the ACM Symposium on Applied Computation. – 1998.
- 36.Скобцов Ю.А., Дубов Я.В. Вероятностные и компактные генетические алгоритмы// Искусственный интеллект.-2006.-№2.- С.108-116.
- 37.Friedberg R.M. A learning machine: part1// IBM J. research and development,1958.-V.2.
- 38.Кнут Д. Искусство программирования.Т.1.-М.:Мир,1976.
- 39.Brameier M., Banzhaf W. A comparison of linear genetic programming and neural networks in medical data mining// IEEE Transactions on Evolutionary Computation 5 (2001).- P. 17 – 26.
- 40.Kantschik W., Banzhaf W. Linear-graph GP – a new GP structure//Proceedings of the 4th European Conference on Genetic Programming, EuroGP 2002, Kinsale, Ireland, E. Lutton, J. Foster, J. Miller, C. Ryan and A. Tettamanzi (Eds.), Springer LNCS 2278, Berlin, 2002, pp. 83-92
- 41.Скобцов Ю.А., Скобцов В.Ю. Модификации генетического программирования//Труды конференций "Интеллектуальные системы" и "Интеллектуальные САПР".-Москва:Физматлит.- 2004.- с.76-81.
- 42.Iba H. and de Garis H. Extended genetic programming with recombinative guidance. – In P.J. Angeline&L.E.Kinnear, Jr.(Eds) Advances in Genetic Programming 2(pp.69-88),Cambridge,MA:MIT Press.

43. Gruau F. Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm. PhD thesis, Laboratoire de l'Informatique du Parallelisme, Ecole Normale Supérieure de Lyon, France.
44. Pruiainkevich P. and Lindermayer A. The Algorithmic Beauty of Plants. – Springer-Verlag, New York, 1990.
45. Whigham P.A. Grammatically-based genetic programming. // In Roska J.P., editor, Proceedings of the workshop on Genetic Programming: From Theory to Real-World Application, pp.64-75, Tahoe City, CA.
46. S.Kent. Diagnosis of oral cancer using genetic programming. Technical Report CSTR-96-14.-1996.-20p.
47. Скобцов Ю.А., Васяева Т.А., Лобачева М.В. Формирование знаний для медицинских экспертных систем на основе генетического программирования//Информационные технологии и информационная безопасность в науке, технике и образовании "ИНФОТЕХ - 2007" Материалы международной научно-практической конференции Часть 2. г. Севастополь, 10-16 сентября 2007г. 38-42с.
48. Skobtsov Y.A., Vasyaeva T.A. Diagnosis of SIDS using Genetic Programming. // Advanced Computer Systems and Networks: Design and Application. Proceedings of 3-st International Conference ACSN-2007. 20-22 September, 2007, Lviv, Ukraine 92-93с
49. T. Mitchel. Machine Learning. McGraw-Hill, New York.-1996.
50. DeJong K.A., Spears W.H., Gordon D.F. Using genetic algorithms for concepts learning// Machine Learning.-1993.-13.-P.161-188.
51. Скобцов Ю.А., Хмелевой С.В. Генетический подход к задачам прогнозирования//Наукові праці Донецького національного технічного університету:серія:“Обчислювальна техніка та автоматизація”.- Випуск 90.-Донецьк : ДонНТУ. – 2005.-с.127-136

52. Rechenberg I. Evolution strategy. In J. Zurada, R. Marks I, C. Robinson. Computational Intelligence – Imitating Life, Piscataway, NJ: IEEE Press.-1994.-p.147-159.
53. D.B. Fogel, K. Chellaila. Revisiting Evolutionary Programming // <http://vision.ucsd.edu/~kchellp/>.
54. D.B. Fogel. Evolutional computation: toward a new philosophy of machine intelligence. New York: IEEE Press.- 1995.
55. Beantley P.J. Evolutionary Design be Computers. San Francisco: Morgan Kaufmann.-1999.
56. Chambers L.D. Practical Handbook of Genetic Algorithms. Complex Coding Systems.-volume III.-CRC PRESS,1999.
57. В.В.Корнеев, А.Ф.Гарев,С.В.Васютин,В.В.Райх. Базы данных. Интеллектуальная обработка информации.-М.: “Нолидж”,200.-352с.
58. S.Scott,A.Seth.HGA: A hardware-based genetic algorithm//ProceedingsACM/SIGDA 3-rd Int. Symp.Field-Programmable Gate Arrays,1995.-pp.53-59.
59. D.Scott, S.Seth,A.Samal. A hardware Engine for Genetic Algorithms. Technical Report UNL-CSE-97-001,University of Nebraska-Lincoln,Dept Computer Science and Engineering, University of Nebraska-Lincoln 4 July 1997.
60. G.Koonar,S.Areibi,M.Moussa. Hardware implementation of genetic algorithms for VLSI CAD design//
61. C.Apporntewan,P.Chongstitvatana. A hardware implementation of the compact genetic algorithm//Proceedings 2001 IEEE Congress Evolutionary computation,Seul,Korea,2001,pp.624-629.