

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

Государственное образовательное учреждение
высшего профессионального образования

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ

М. В. Бураков

ГЕНЕТИЧЕСКИЙ АЛГОРИТМ: ТЕОРИЯ И ПРАКТИКА

Учебное пособие

Санкт-Петербург
2008



УДК 004.4
ББК 22.12
Б91

Рецензенты:

кафедра интегрированных систем управления Санкт-Петербургского
государственного политехнического университета;
доктор технических наук, профессор кафедры автоматики и
процессов управления Санкт-Петербургского государственного
электротехнического университета «ЛЭТИ» им. В. И. Ульянова (Ленина)
С. Е. Душин

Утверждено
редакционно-издательским советом университета
в качестве учебного пособия

Бураков М. В.

Б91 Генетический алгоритм: теория и практика: учеб. пособие / М. В. Бураков. – СПб.: ГУАП, 2008. – 164 с.: ил.

ISBN 978-5-8088-0298-8

В учебном пособии рассматриваются теоретические и прикладные вопросы использования генетического алгоритма – мощного метода глобальной оптимизации, который в последние годы активно используется в инженерной практике.

Приводится описание генетического алгоритма и его основных модификаций, рассмотрены принципы использования генетического алгоритма для решения конкретных задач. Описываются возможности пакета математического моделирования MatLab для организации генетических вычислений.

Учебное пособие предназначено для подготовки специалистов, бакалавров и магистров по направлению 220200 «Автоматизация и управление», а также для широкого круга других специальностей, в которых актуальными являются задачи оптимизации в больших поисковых пространствах.

УДК 004.4
ББК 22.12

ISBN 978-5-8088-0298-8

© ГУАП, 2008
© М. В. Бураков, 2008

Содержание

Введение.....	5
1. Основы генетического алгоритма	9
1.1. Глобальная оптимизация.....	9
1.1.1. Задачи локальной и глобальной оптимизации	9
1.1.2. Классификация методов глобальной оптимизации...	11
1.1.3. Алгоритм имитации отжига	12
1.1.4. Генетический алгоритм и другие методы оптимизации	14
1.2. Общие сведения о генетическом алгоритме.....	16
1.2.1. Основные представления об эволюции.....	16
1.2.3. Классический генетический алгоритм.....	22
1.2.4. Пример работы классического генетического алгоритма	27
Вопросы для самопроверки.....	31
2. Механизмы работы генетического алгоритма.....	33
2.1. Строительные блоки в генетическом алгоритме	33
2.2. Кодирование параметров в генетическом алгоритме	35
2.3. Варианты оценки пригодности хромосом	38
2.4. Операция селекции.....	41
2.4.1. Варианты стохастического отбора.....	41
2.4.2. Турнирная селекция.....	44
2.4.3. Селекция отсечением	44
2.4.4. Селекция генов	45
2.5. Варианты операции скрещивания.....	48
2.5.1. Бинарный кроссовер.....	48
2.5.2. Всеобщий кроссовер (uniform crossover)	50
2.6. Операция мутации.....	51
2.7. Миграционная модель генетического алгоритма	52
Вопросы для самопроверки	53
3. Приложения генетического алгоритма	55
3.1. Генетический синтез регуляторов	55
3.1.1. Общие принципы синтеза регуляторов	55
3.1.2. Синтез ПИД-регуляторов	58
3.1.3. Синтез нечетких регуляторов.....	60
3.1.4. Нейроконтроллеры.....	67
3.2. Мультиагентные системы	74
3.2.1. Понятие интеллектуального агента.....	74
3.2.2. Архитектура и обучение интеллектуального агента	76
3.3. Генетическое программирование	82
3.3.1. Автоматическое составление программ	82

3.3.2. Механизмы языка ЛИСП	83
3.3.3. ЛИСП и генетическое программирование	85
3.4. Генетическое тестирование программного обеспечения .	88
3.4.1. Проблемы тестирования программ	88
3.4.2. Принципы тестирования программ.....	89
3.4.3. Количественная оценка надежности программы.....	92
3.4.4. Генетическое тестирование программ.....	93
Вопросы для самопроверки	96
4. Генетический алгоритм в MatLab	99
4.1. Общие сведения.....	99
4.2. Использование графического окна GATool	100
4.2.1. Общий вид графического окна.....	100
4.2.2. Описание задачи и ограничений	102
4.2.3. Визуализация результатов работы алгоритма.....	104
4.2.4. Параметры запуска генетического алгоритма.....	111
4.2.5. Панель Options	111
4.3. Запуск из командной строки и М-файла.....	121
Вопросы для самопроверки	124
5. Решение задач в MatLab	126
5.1. Минимизация функций.....	126
5.2. Решение диофантова уравнения	128
5.3. Настройка ПИД-регулятора	131
5.4. Настройка нечеткого регулятора	134
5.4.1. Настройка масштабирующих коэффициентов	134
5.4.2. Настройка регулятора Сугэно	142
5.5. Настройка нейронной сети.....	146
5.6. Задачи для самостоятельной работы.....	148
5.6.1. Поиск экстремума функции одной переменной.....	148
5.6.2. Поиск экстремума функции двух переменных.....	149
5.6.3. Задачи линейного программирования	150
5.6.4. Решение задачи нелинейного программирования	154
5.6.5. Генетический синтез параметров регулятора.....	156
Заключение	158
Библиографический список	160

ВВЕДЕНИЕ

Генетический алгоритм (ГА) является мощным инструментом для эволюционного решения сложных задач.

Генетический алгоритм – это стохастический поисковый алгоритм, который итеративно трансформирует множество математических объектов (популяцию), представляющих собой кодированные решения некоторой задачи. С каждым объектом (хромосомой) связывается оценка качества решения задачи, на основании которой выполняется селекция, имитирующая процесс естественного отбора по Дарвину. Отобранная популяция потомков подвергается операциям скрещивания и мутации, имитирующим генетический процесс передачи и преобразования наследственной информации. Итерации продолжаются до получения приемлемого качества решения. Таким образом, ГА опирается на современные представления о механизмах эволюции и генетики.

Считается, что стандартный ГА впервые описан в работе [1]. Всеобщий интерес к ГА вызвала работа [2], в которой была сделана попытка его математического обоснования. В последующие годы развитию ГА было посвящено множество зарубежных публикаций, в том числе ряд обобщающих работ [3–6].

В Советском Союзе также происходило развитие идей эволюционного поиска решений сложных задач. Так, в 70-х годах в рамках теории случайного поиска Л. А. Растригиным был предложен ряд алгоритмов, использующих идеи бионического поведения особей [7]. Развитие этих идей нашло отражение в ряде работ И. Л. Букатовой по эволюционному моделированию [8, 9]. Ю. И. Неймарк предложил осуществлять поиск глобального экстремума на основе коллектива независимых автоматов [10].

В России в последние годы были изданы несколько учебников и монографий, посвященных ГА [11–14]. Продолжает расти поток научных публикаций, посвященных теоретическим и прикладным аспектам использования ГА ([15–18] и др.).

Генетический алгоритм предназначен для поиска экстремума сложных функций. Сложность оптимизируемой функции определяется как количеством переменных, по которым ведется оптимизация, так и наличием локальных экстремумов. В задачах оптимизации пространство поиска может быть практически безграничным, поэтому невозможно доказать, что найденное с помощью ГА решение является наилучшим. Однако такое доказательство обычно и не требуется, важно лишь, чтобы найденное решение в достаточной степени удовлетворяло смыслу решаемой задачи. Поэтому говорят, что с помощью ГА можно получить квазиоптимальное решение.

Общая задача поиска квазиоптимального решения в разных предметных областях получает свою специфику, поэтому конкретные сферы использования ГА весьма разнообразны. К ним относятся:

- аппроксимация функций и регрессионный анализ;
- поиск кратчайших путей (задачи коммивояжера);
- комбинаторная оптимизация;
- параметрический дизайн;
- задачи размещения и составления расписаний;
- задачи автоматического программирования и тестирования программ;

– техническое проектирование.

В системах управления ГА используется для:

- выбора структуры и параметров искусственных нейронных сетей;
- оптимизации параметров регуляторов (в том числе – нейронных и нечетких);
- проектирования мультиагентных систем и клеточных автоматов;
- оптимизации траекторий робота, в том числе обучение робота ходьбе.

Последняя задача является одним из наиболее наглядных приложений ГА. В Японии был разработан робот Pino (сокращение от Pinokkio), отличающийся относительно невысокой стоимостью при простоте конструкции (рисунок). Pino уже поступил в массовую продажу (в том числе – в России), он может «разговаривать», «петь», двигать руками и ногами, ходить, танцевать, выражать свое настроение.



*Шагающий робот
Pino*

До появления Pino на рынке были сложные и дорогие высокотехнологичные роботы с мощными моторами для управления «ходьбой». Алгоритм управления строился на основании анализа походки человека и движения его суставов.

При создании робота Pino оказалось возможным использовать намного менее мощные моторы за счет того, что Pino сам научился ходить, используя ГА. Сначала Pino «еле двигал ногами», но потом довольно быстро научился ходить, а затем и танцевать. Обучение Pino во многом напоминает поведение маленького ребенка, который сначала часто падает, осваивая

собственные конечности, но постепенно накапливает опыт, формируя в своем мозгу «программы» движения.

Еще одна интересная научно-техническая проблема, в решении которой может помочь ГА, – обучение полету искусственных насекомых. Механизмы полета насекомых до сих пор не вполне понятны, поскольку реальные аэродинамические характеристики шмеля или майского жука сильно отличаются от тех значений, которые бы позволили им летать в соответствии с методами расчета летательных аппаратов.

Ученым из Технического университета в Гётеборге (Швеция) удалось построить робота, который самостоятельно, методом проб и ошибок, научился летать. Первоначально робот имел только органы, необходимые для полета, но не умел ими пользоваться. Основную часть его конструкции составляли два легких крыла, которые должны были приводиться в движение с помощью многочисленных миниатюрных моторов. Робот был установлен на двух направляющих планках, которые позволяли ему перемещаться только по вертикали.

В начале эксперимента в компьютере робота отсутствовали сведения об управлении полетом. Каждые двадцать секунд он генерировал случайную последовательность команд для моторов. Соответственно, крылья перемещались случайно, и поведение робота было хаотичным. Каждая серия команд оценивалась с точки зрения длительности подъема робота в воздух. С помощью механизмов ГА происходила эволюция поведения робота, которая завершилась, после нескольких часов обучения, устойчивым парением в воздухе.

Расчет орбит спутников для эффективной связи также оказался приложением, в котором ГА показал хорошие результаты. Эта проблема заключается в том, что летающие на самой большой (более 35 тыс. км) высоте спутники способны «видеть» половину земного шара, однако стоимость их вывода на такие орбиты очень высока. Более дешевые орбиты от 130 до 480 км неудобны для спутников вследствие кривизны земной поверхности, которая препятствует связи со станциями в ходе большей части 90-минутного витка вокруг планеты. ГА выбрал наиболее производительные группировки спутников путем изменения таких параметров как удаление одного аппарата от другого и высота полета над поверхностью планеты. Низкоорбитальные спутники незаменимы для пользователей мобильных компьютеров, поскольку дают возможность использовать устройства беспроводной связи.

Генетическое программирование, т. е. автоматическое составление программ с помощью ГА, становится все более реальным по мере увеличения мощности компьютеров.

В Стэнфордском университете генетическое программирование (одно из направлений ГА) нашло применение для оценки дублирования и заимствований на существующем множестве патентов США. Эта система работает на базе сети из 1000 компьютеров. Разработчики системы надеются, что на ее базе можно будет создать «машину изобретений», способную к самостоятельному творчеству.

Деятельность инженера-проектировщика часто связана с проблемой выбора нужного технического решения в условиях огромного количества альтернатив.

Например, при разработке новых реактивных двигателей возникает задача конструирования газовой турбины. Турбина имеет более 100 параметров, на которые наложены несколько десятков условий, так что существует около 10^{380} различных вариантов решения. По сообщениям прессы, в одном довольно типичном случае инженеру понадобилось 8 недель вычислений на рабочей станции, чтобы достичь удовлетворительного варианта конструкции. Дальнейшее применение традиционного метода оптимизации позволило за сутки повысить эффективность конструкции турбины в 2 раза. Однако это оказался локальный максимум оценочной функции, из которого не удавалось выйти. Использование ГА позволило за двое суток обнаружить другой максимум, на 50% выше найденного традиционными средствами оптимизации.

Можно сказать, что ГА прочно вошел в арсенал методов решения сложных задач. Об этом свидетельствует появление библиотек ГА (**Genetic Algorithm and Direct Search Toolbox**) в последних версиях популярного математического пакета MatLab [19]. В сочетании с уже имевшимися в MatLab пакетами **Fuzzy Logic Toolbox** и **Neural Network Toolbox** это дает инструменты для решения многих «интеллектуальных» задач.

Таким образом, изучение ГА является необходимым компонентом в подготовке инженеров-разработчиков систем управления.

1. ОСНОВЫ ГЕНЕТИЧЕСКОГО АЛГОРИТМА

1.1. Глобальная оптимизация

1.1.1. Задачи локальной и глобальной оптимизации

Задачи поиска оптимального решения часто возникают в науке, технике и в любой другой области человеческой деятельности: при моделировании реальных процессов, при синтезе систем управления, идентификации, анализе данных и других областях, где необходимо получить экстремум целевой функции $F(X)$ на множестве допустимых параметров X .

Целевая функция описывает качество решения задачи. Обычно рассматривается задача минимизации $F(X)$, хотя это не принципиально, поскольку задача поиска максимума $F(X)$ равносильна задаче поиска минимума $-F(X)$.

Например, если рассматривается задача оптимизации параметров регулятора, то X – набор параметров регулятора, а целевую функцию можно представить в виде

$$F(X) = \int_{t=0}^T |y^*(t) - y(t)| dt,$$

где $y^*(t)$ – желаемый переходный процесс; $y(t)$ – реальный переходный процесс при параметрах регулятора X .

Сложность поиска оптимального решения зависит от вида целевой функции. Если $F(X)$ имеет один экстремум (униmodalная), то задача оптимизации относительно проста. Для такой *локальной оптимизации* предложено большое количество методов, в частности, методы спуска по градиенту целевой функции (см. [20, 21] и др.).

Если $F(X)$ имеет много экстремумов (мультимodalная), то такая задача *глобальной оптимизации* значительно усложняется.

На рис. 1.1 показаны примеры графиков одномерной униmodalной (1.1, а) и мультимodalной (1.1, б) целевой функции.

Задача глобальной минимизации описывается следующим образом:

$$F(X) \rightarrow \min; \quad D \subset R^n,$$

где D – допустимое множество решений задачи; R^n – n -мерное евклидово пространство.

Можно рассматривать множество оптимальных решений задачи

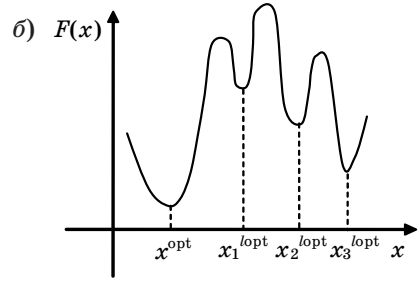
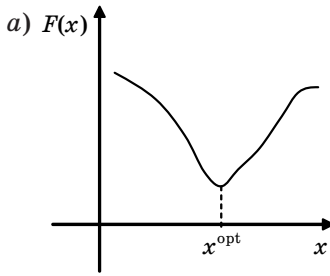


Рис. 1.1. Виды целевой функции

$$X^{\text{opt}} = \underset{x \in D}{\operatorname{argmin}} F(X),$$

и множество локально-оптимальных решений задачи

$$X^{\text{lopt}} = \{X \in D; \exists \varepsilon > 0 : F(X) \leq F(X'), \forall X' \in B_\varepsilon(X) \cap D\},$$

где

$$B_\varepsilon(X) = \{X' \in R^n; |X' - X| < \varepsilon\}.$$

Таким образом, задача глобальной оптимизации заключается в поиске точки $X^{\text{opt}} \neq X^{\text{lopt}}$.

Процедуру (или алгоритм) решения задачи оптимизации можно представить в виде итеративного процесса, который порождает последовательность точек в соответствии с предписанным набором правил, включающим критерий окончания счета. Поиск глобального решения задачи оптимизации происходит путем перебора локальных решений.

Существует множество методов глобальной оптимизации [21–26], однако все они имеют общие черты:

- для поиска глобального оптимума следует анализировать различные области множества решений X ;
- в каждой области происходит поиск оптимума с помощью алгоритма локальной оптимизации (спуск по градиенту);
- необходим критерий остановки алгоритма на основании качества полученного решения, поскольку область поиска может быть бесконечно большой.

Нужно подчеркнуть, что в общем случае нельзя гарантировать точное решение задачи глобальной оптимизации для многоэкстремальной функции за конечное число шагов. Для доказательства того, что найденное решение является глобальным оптимумом, надо выполнить полный перебор всех возможных значений векто-

ра параметров. В большинстве случаев это невозможно, поэтому при глобальной оптимизации речь идет обычно не о поиске оптимального, а о поиске субоптимального решения:

$$X \in X^{\text{opt}}(\varepsilon) = \{X \in D; F(X) \leq F(X^{\text{opt}}) + \varepsilon\},$$

где ε – малая величина (с учетом специфики решаемой задачи).

1.1.2. Классификация методов глобальной оптимизации

Обзор методов глобальной оптимизации приводится, например, в работах [22–26]. Все известные методы глобальной оптимизации можно разделить на две категории: детерминированные и стохастические.

Детерминированные методы получают глобальное решение посредством исчерпывающего поиска на всем допустимом множестве [27]. Поэтому большинство детерминированных методов теряют эффективность и надежность с возрастанием размерности задачи. Более того, чтобы гарантировать успех, такие методы требуют выполнения дополнительных предположений, наложенных на функцию цели.

Стохастические алгоритмы более универсальны. Здесь стохастический подход присутствует не только в разработке и анализе алгоритма, но и используется в решении базовых проблем, например при определении условия останова. Большинство стохастических методов оценивают значение функции цели в случайных точках допустимого множества с последующей обработкой выборки ([28–30] и др.).

Исторически первым методом глобальной оптимизации является метод Монте-Карло, на базе которого был создан метод мультистарта [30]. В этом методе из множества D случайно или детерминированно выбирается некоторое подмножество из N точек. Последовательно из каждой точки запускается алгоритм локального спуска, и из полученного множества локальных решений выбирается наилучшее. В чистом виде метод мультистарта не является эффективным, так как одно и то же локальное решение может быть найдено не один раз. Мультистарт – это обобщенный подход: большинство эффективных методов глобальной оптимизации основано на идее метода мультистарта – запуска стандартных локальных алгоритмов из множества точек, равномерно распределенных на множестве D . Таким образом, метод мультистарта можно назвать прототипом таких методов.

Методы группировок (clustering methods) [31] являются одной из модификаций метода мультистарта. Здесь предпринята попыт-

ка устранить главный недостаток мультистарта путем тщательного отбора точек, из которых запускается локальный поиск. Рассматривается некоторая выборка точек, например равномерно распределенных на D . Затем, пока не выполнится некоторое условие останова, последовательно выполняются следующие три шага:

1) из каждой точки запускается алгоритм локального спуска, в результате будет получен набор локальных решений;

2) используя специальную технику группировки, определяют группы точек;

3) в качестве новой выборки точек рассматривается каждая m -я точка из группы и осуществляется переход к первому шагу. Таким образом, решения находятся посредством локальных алгоритмов спуска из наилучших точек каждой группы.

Основная идея метода имитации отжига (simulated annealing) [32, 33] исходит из физики процесса замерзания жидкостей или рекристаллизации металлов в процессе отжига. Целевая функция здесь является аналогом равновесия термодинамической системы и видоизменяется путем добавления случайных величин (условий температурного режима). Процесс повторяется достаточное число раз для каждой температуры, после чего температура понижается и весь процесс происходит снова до состояния полной заморозки. Избегание попадания в незначительные локальные минимумы (замерзание) зависит от «схемы отжига», выбора начальной температуры, количества итераций для каждой температуры и того, насколько уменьшается температура на каждом шаге процесса «охлаждения».

Рассмотрим алгоритм имитации отжига подробнее.

1.1.3. Алгоритм имитации отжига

Метод имитации отжига отражает поведение расплавленного материала при отвердевании с применением процедуры отжига (управляемого охлаждения) при температуре, последовательно понижаемой до нуля.

В расплавленном металле атомы находятся в сильном беспорядочном движении, а по мере охлаждения все большее их количество переходит в состояние с меньшими энергиями, пока, в конце концов, не будет достигнут глобальный минимум энергии. Распределение энергетических уровней при этом описывается формулой

$$P(E) = \frac{1}{e^{\frac{E}{kT}}}, \quad (1.1)$$

где $P(E)$ – вероятность нахождения системы в состоянии с уровнем энергии E ; k – постоянная Больцмана; T – температура по Кельвину.

При высоких температурах вероятность любого энергетического состояния близка к 1, а при уменьшении T вероятность высоких энергий падает.

Алгоритм глобальной оптимизации с помощью имитации отжига представляет собой аналог физического процесса. Классический алгоритм имитации отжига можно описать следующим образом.

1. Искусственная температура получает максимальное значение $T = T_{\max}$. Модельное время обнуляется: $t = 0$.

2. Выбирается начальная точка (аргумент функции) $x = x_0$.

3. Рассчитывается целевая функция $F(x)$.

4. Аргумент получает приращение $x' = x + \Delta x$.

5. Рассчитывается целевая функция $F(x')$.

6. Рассчитывается изменение целевой функции $\Delta F = F(x') - F(x)$.

7. Если $\Delta F < 0$, то $x = x'$ (значение аргумента сохраняется).

8. Если $\Delta F > 0$, то вероятность сохранения ΔF (и, соответственно, x') вычисляется по формуле, подобной (1.1):

$$P(\Delta F) = \frac{1}{e^{\frac{\Delta F}{kT}}},$$

где k – константа, выбираемая из эвристических соображений; T – искусственная температура.

Затем $P(\Delta F)$ сравнивается со случайным числом $n \in [0, 1]$. Если $P(\Delta F) > n$, то $x = x'$, иначе x не изменяется.

9. Искусственная температура уменьшается, модельное время увеличивается: $t = t + \Delta t$, происходит возврат к п. 4, и так до тех пор, пока T не уменьшится до заданного порогового значения.

Таким образом, пока искусственная температура T большая, алгоритм отжига может делать большие шаги даже в направлениях, увеличивающих значение минимизируемой функции. За счет этой особенности оказывается возможным попасть в окрестность глобального минимума. При этом, конечно, важно правильно определить начальное значение T_{\max} и задать закон ее изменения. Считается, что эта величина должна быть обратно пропорциональна логарифму t :

$$T(t) = \frac{T_{\max}}{\log(1+t)}.$$

Эта формула показывает, что искусственная температура должна меняться медленно.

1.1.4. Генетический алгоритм и другие методы оптимизации

Генетический алгоритм относится к классу стохастических алгоритмов глобальной оптимизации. Как и другие алгоритмы этой группы, он ориентирован на поиск субоптимального решения в условиях невозможности полного перебора вариантов.

Для того чтобы подчеркнуть ограниченность возможностей полного перебора вариантов в сложных проблемах, рассмотрим задачу коммивояжера, которая формулируется так. Имеется n городов, расположенных на различном расстоянии друг от друга и соединенных дорогами. Требуется обойти все города кратчайшим путем, побывав в каждом городе только однажды. На рис. 1.2 показан пример для пяти городов.

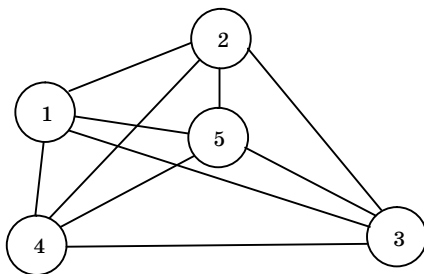


Рис. 1.2. Задача коммивояжера для пяти городов

Когда рассматривается n городов, при построении маршрута существует n вариантов выбора первого города, $n-1$ вариантов выбора второго города и т. д. Таким образом, количество вариантов путей в задаче для n городов оказывается равно $n!$

И если для $n=5$ получается $5!=120$ вариантов, то для $n=20$ получается $20! \approx 2,4 \cdot 10^{18}$ вариантов. Несложные подсчеты показывают, что при скорости расчетов 500 млн вариантов в секунду потребуется более 150 лет для полного перебора вариантов. Таким образом, решение задач перебором вариантов возможно далеко не всегда.

Сделанное замечание позволяет утверждать, что ГА в среднем будет всегда эффективнее, чем перебор вариантов (рис. 1.3).

Отличия ГА от других алгоритмов случайного поиска заключаются в следующем:



Рис. 1.3. Качественное сравнение различных методов оптимизации

1) в ГА используются коды параметров, а не сами параметры. Параметры кодируются в цепочки конечной длины, в процессе итераций эти цепочки тестируются и видоизменяются;

2) поиск происходит не в точках пространства, а на основе популяции точек;

3) ГА использует знания о прошлых полученных решениях. Он обеспечивает концентрацию решений в наиболее эффективных областях пространства решений;

4) ГА использует только оценки решений, а не их производные или другие параметры.

В ГА не накладываются ограничения на свойства целевой функции в виде непрерывности, дифференцируемости и т. д. Она подразумевается как устройство типа «черного ящика», которое на вход получает параметры, а на выход выводит результат.

Следует подчеркнуть, что ГА в окрестности глобального минимума может быть менее эффективным, чем алгоритм спуска по градиенту или другой алгоритм локальной оптимизации (см. рис. 1.3). Это выражается в меньшей скорости сходимости и, возможно, в меньшей точности полученного решения. ГА является робастным алгоритмом, т. е. позволяет находить хорошее решение (субоптимальное), но нахождение точного оптимального решения может оказаться трудной задачей в силу стохастичности принципов работы алгоритма.

Поэтому ГА можно использовать не только в «чистом виде», но и совместно с традиционными алгоритмами локальной оптимизации. ГА применяется на начальном этапе для эффективного суже-

ния пространства поиска, а затем, попав в окрестность глобального экстремума, можно применить один из «классических» методов оптимизации, отличающихся высокой точностью.

Важное достоинство ГА заключается в том, что он естественным образом позволяет распараллеливать вычисления. Это значительно увеличивает скорость расчетов при использовании компьютерной сети или многопроцессорных компьютеров.

1.2. Общие сведения о генетическом алгоритме

1.2.1. Основные представления об эволюции

Современные доминирующие представления об эволюции основаны на синтезе дарвинизма и генетики.

Дарвинизм, как научная парадигма, изменил мировоззрение человечества. Споры относительно дарвинизма продолжаются до сих пор, поэтому будет не лишним напомнить основные положения этой теории.

Первые основания своей теории Ч. Дарвин заложил в 1835 г., находясь на Галапагосских островах. Изучая местных птиц, он сравнил экземпляры с четырех островов и убедился в их отличии друг от друга и от материковых форм. Этот факт позволил высказать гипотезу, что изоляция может вести к появлению новых видов. Чем дольше изоляция, тем больше возникающие различия, что приводит к возникновению новых разновидностей, а затем и видов. Таким образом, Дарвин решил искать эволюцию в медленном накоплении самых обычных изменений организмов.

Механизм естественного отбора был известен биологам и до Дарвина как процесс, обеспечивающий стабильность видов. Сильные и жизнеспособные особи выживают, оставляя потомство, которому передаются их особенности. Слабые особи, напротив, гибнут, не участвуя в размножении.

Дарвин впервые выдвинул идею об эволюционной силе естественного отбора и основал всю свою теорию на этом утверждении. Название основополагающего труда Дарвина говорит о том, что он взял за основу своей теории именно идею естественного отбора. Книга Дарвина «О происхождении видов путем естественного отбора, или сохранение удачных пород в борьбе за жизнь» вышла в свет в Лондоне в ноябре 1859 г. [34].

Естественный отбор – это процесс выживания и воспроизведения организмов, наиболее приспособленных к условиям среды, и гибели в ходе эволюции неприспособленных.

Согласно Дарвину, естественный отбор включает три компонента:

- возникновение множества наследуемых малых случайных (ненаправленных) мутаций;
- выживание наиболее приспособленных мутантов в результате конкуренции и взаимодействия со средой;
- накопление выживающих на протяжении ряда поколений малых мутаций в адаптивные (прогрессивные) признаки.

Таким образом, эволюционная теория утверждает, что жизнь на нашей планете возникла вначале лишь в простейших формах – в виде одноклеточных организмов. Затем эти формы постепенно усложнялись, приспособляясь к окружающей среде и порождая новые виды. Каждый биологический вид с течением времени совершенствует свои качества так, чтобы эффективней справляться с важнейшими задачами выживания, самозащиты, размножения и т. д. Таким путем возникла защитная окраска у многих рыб и насекомых, панцирь – у черепахи, яд – у скорпиона и т. п.

Однако происхождение человека не удалось объяснить идеей естественного отбора. Можно говорить о преобразовании анатомии человека, но вопрос о возникновении мышления оставался открытым. Никто не брался объяснить, как за счет конкуренции диких предков человека образовался интеллект. Для ответа на этот вопрос Дарвин добавил к естественному отбору половой отбор [35].

Половым отбором в дарвинизме именуется преимущественное размножение самцов с теми качествами, которые позволяют победить в борьбе за самку.

Дарвин считал, что для того чтобы избегать неприятелей или успешно нападать на них, ловить диких животных, выделять оружие, необходимо развитие умственных способностей. Эти способности развились у мужчины частью путем полового отбора, т. е. путем борьбы между соперничающими мужчинами, частью – путем естественного отбора, т. е. успеха в общей борьбе за жизнь. Таким образом, речь шла о наследовании потомством накопленного в течение жизни опыта.

Первоначально Дарвин использовал гипотезу пангенеза (всеобщего порождения). Этот принцип (еще античный) предполагает, что каждая часть тела посылает свою частицу в орган размножения, где частицы вместе и образуют семенную жидкость. Но в 1883 г. немецкий зоолог А. Вейсман указал, что для передачи потомству изменение должно попасть в половую клетку, тогда как проявляется оно в соматических (неполовых) клетках. То есть приобретенные в

течение жизни признаки не могут наследоваться. Это стало серьезнейшим возражением против теории Дарвина.

Однако распространение генетики, датой рождения которой считают 1865 г. (законы Менделя), позволило придать новый импульс дарвинизму. Генетика может быть использована для обоснования внутренних механизмов эволюции.

Ген – единица наследственной информации. Гены образуют хромосомы. В каждой клетке организма есть набор хромосом, несущих информацию обо всей особи.

Эволюционируют хромосомы, а не сами живые существа. Признаки особи текущего поколения формируются в результате репродукции, которая происходит при рекомбинации генов родителей из предыдущего поколения. Слабые особи, которые не обладают качествами, способствующими выживанию, с большой вероятностью не проживут долго и не смогут создать многочисленное потомство. Кроме того, им сложнее будет найти хорошую пару для скрещивания, поэтому с большой вероятностью генотип таких особей исчезнет из генофонда популяции.

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

В клетке живого организма хромосомой является молекула ДНК (дезоксирибонуклеиновая кислота), окруженная защитной оболочкой. Каждая молекула ДНК – это цепочка, состоящая из молекул *нуклеотидов* четырех типов, обозначаемых А, Т, С и Г. Таким образом, генетический код индивидуума – это длинная строка символов, где используются 4 буквы. Клетки человека содержат 46 хромосом. Каждая хромосома имеет длину порядка 100 000 генов. Ген – это отрезок цепи ДНК, ответственный за определенное свойство особи, например за цвет глаз, тип волос, цвет кожи и т. д. В каждой клетке любого животного содержится вся генетическая информация об этой особи.

Различают два вида клеток: половые (сперматозоид и яйцеклетка) и соматические. В каждой соматической клетке человека содержится 46 хромосом, они образуют 23 пары, причем в каждой паре одна из хромосом получена от отца, а вторая – от матери. В половых клетках хромосом только 23, и они непарные. При оплодотворении происходит слияние мужской и женской половых клеток и получается клетка зародыша, содержащая 46 хромосом.

Интересно, что около 95% генома человека и шимпанзе совпадают. В 5% отличий укладываются все внешние признаки чело-

века – строение руки, стопы, черепа и т. д., а также внутренние признаки – способность к членораздельной речи, абстрактному и логическому мышлению.

Какие свойства потомок получит от отца, а какие – от матери? Это зависит от того, какие именно половые клетки принимали участие в оплодотворении. Процесс выработки половых клеток (мейоз) в организме подвергается случайностям, благодаря которым потомки во многом отличаются от своих родителей. При мейозе парные хромосомы соматической клетки сближаются вплотную, потом их нити ДНК разрываются в нескольких случайных местах и хромосомы обмениваются своими частями. Этот процесс обеспечивает появление новых вариантов хромосом и называется «кроссинговер». Каждая из вновь появившихся хромосом окажется затем внутри одной из половых клеток, и ее генетическая информация может реализоваться в потомках данной особи.

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

Таким образом, накопление и распространение мутаций может привести к появлению нового вида. Естественный отбор регулирует мутации, закрепляя только полезные новые признаки. Наиболее приспособленные особи дадут большое количество потомков.

Синтез генетики и дарвинизма заключается в том, что эволюция происходит путем естественного отбора мутаций.

Дарвиновский подход к эволюции подвергался и подвергается обоснованной критике. Вот некоторые из возражений классическому дарвинизму.

1. Скорость эволюции по Дарвину была бы неизмеримо ниже, чем наблюдаемая в природе. Для получения полезного признака организму требуется не одна, а несколько последовательных мутаций. Так, ящерице, чтобы превратиться в птицу, требуется: и облегчить кости, и отрастить перья, и изменить строение скелета, и т. д. Допустим, что при каждой мутации гена возможно 10 вариаций, а всего требуется 10 мутаций. Тогда для получения из ящерицы птицы нужно отобрать одну особь из 10 млрд мутантов. Если же

число вариаций гена равно 100, то цифры становятся совсем астрономическими.

2. Чтобы мутация не терялась при скрещиваниях, она должна возникнуть сразу у значительного процента особей. При этом должны отсутствовать мутации, ухудшающие вид. Это еще более усиливает предыдущее замечание и ставит под сомнение естественный отбор как движущий фактор эволюции.

3. Множество признаков организмов никак не связаны с естественным отбором. Так, длинная шея и окраска жирафа хорошо согласуется с теорией, но полосатая шкура зебры едва ли приносит ей ощутимые преимущества в саванне.

4. В теории естественного отбора мутации уничтожаются или не уничтожаются средой, в результате чего происходит накопление малых изменений в направлении, средой задаваемом. Однако органическая эволюция не может быть объяснена только адаптацией к среде, поскольку появляющиеся сложные формы часто не превосходят по адаптированности старые формы. Так, бактерии и лишайники проявляют выживаемость в самых суровых условиях.

5. Многие (если не все) существующие виды проявляют феноменальный «консерватизм», сохраняя неизменными свои признаки в течение миллионов лет. Например, знаменитая кистеперая рыба целакант, которую дарвинисты считали переходным звеном от рыб к земноводным, оказалась не ископаемым, а существующим видом (поймана в 30-е гг. XX века).

6. Естественный отбор не в состоянии объяснить происхождение систем и органов, имеющих комплексное строение. Эти системы и органы образуются в результате совокупной деятельности взаимосвязанных частей, и отсутствие или дефект хотя бы одной из них приводит к нарушению их функций. Таким системам свойственна «неупрощаемая сложность». К примеру, строение человеческого глаза не может быть проще, чем оно есть, так как отсутствие какой-либо части этого органа станет причиной его неполноценного функционирования.

7. До сих пор не было зафиксировано появление новых видов с помощью механизма естественного отбора. Искусственный отбор (в животноводстве) можно рассматривать как многократно ускоренный естественный отбор, но и здесь новые виды получены не были. Возможно, методы генной инженерии позволят получать новые виды животных, однако это не имеет прямого отношения к дарвинизму.

Эти возражения можно было бы продолжить. Однако теория Дарвина, во многом несовершенная, способствовала развитию эволюционных представлений.

В последние десятилетия во всем мире активно развивается универсальный эволюционизм, в рамках которого эволюция всего сущего – от Большого взрыва нашей Метагалактики до био- и ноосферы на Земле – рассматривается в едином ключе. Это помогает выработать определенный взгляд на эволюцию как процесс развития наблюдаемого мира в определенном направлении в результате саморазвития материи. Саморазвитие связано с рядом процессов, таких как [36]:

- 1) возрастание сложности и разнообразия форм;
- 2) интенсификация «метаболизмов» разной природы, включая энергообмен и обмен веществ, химические метаболизмы и «метаболизмы» социальные;
- 3) интенсификация и расширение круговоротов энергии и вещества;
- 4) рост связанности «всего со всем» и др.

Таким образом, теория эволюции по Дарвину является только одной из моделей, не объясняющей все аспекты развития органической материи. Однако на основании этой модели могут быть построены эффективные алгоритмы, с помощью которых можно описать эволюцию виртуальных объектов, более простых, чем живые организмы.

Рассмотрим основные термины, заимствованные из биологии и генетики и используемые при описании эволюционных алгоритмов.

Популяция – многочисленная совокупность особей определенного вида, населяющая определенное географическое пространство. Каждая популяция эволюционирует самостоятельно, подчиняясь генетическим факторам. Популяция является основой эволюционных процессов в органическом мире. Популяции сменяют друг друга.

Поклоение – очередная популяция.

Организм (особь) – отдельный элемент популяции. Каждая особь популяции является носителем уникального набора признаков.

Фен – отдельный признак организма.

Фенотип – совокупность признаков организма.

Хромосома – материальный носитель признаков организма.

Ген – атомарная (неделимая) часть хромосомы, отвечающая за конкретный признак. Иногда ген называют детектором.

Геном – совокупность генов организма.

Алель – значение конкретного гена.

Локус (позиция) – место конкретного гена в хромосоме.

Репродукция – процесс возникновения новых хромосом, включающий скрещивание и мутации.

Скрещивание – создание двух новых хромосом – потомков из двух хромосом-предков.

Мутация – произвольное изменение отдельных генов организма.

Для практического применения ГА в инженерных задачах необходимо использовать еще одно понятие – *функция относительной пригодности* (ОП). В англоязычной литературе принят термин *fitness function*. В живой природе подобная функция существует неявно – она характеризует процесс выживания и распространения наиболее приспособленных организмов.

В задачах, которые ставит перед собой человек, функция ОП характеризует качество решения. Так, например, в животноводстве для получения новых пород животных нужно отбирать особи, дающие больше молока или имеющие более густую шерсть. В инженерных задачах ОП соответствует целевой функции (в оптимизации), функции ошибки (в теории управления), функции цены (в теории игр) и т. п. При проектировании регуляторов автоматических систем ОП должна быть выше у тех вариантов, которые обеспечивают лучшие показатели качества управления.

1.2.3. Классический генетический алгоритм

Генетический алгоритм воплощает представления о дарвиновской эволюции на основе генерации, тестирования и отбора наиболее жизнеспособных особей.

Генетический алгоритм является универсальным алгоритмом, поскольку под особью понимается вариант решения некоторой задачи. Каждое возможное решение представляет собой точку в пространстве поиска.

Хромосома кодирует признаки особи, т. е. хромосома – это кодированное решение задачи (рис. 1.4).



Рис. 1.4. Соотношение между генотипом и фенотипом

Для кодирования решения могут использоваться вещественные числа или двоичный алфавит: $\{0, 1\}$. При использовании двоичного алфавита строка битов длиной m может рассматриваться как хромосома. Отдельные позиции хромосомы (биты) выступают в качестве генов.

Рассмотрим простой пример. Пусть необходимо найти максимум функции одной переменной $F(x)$. Если кодировать x цепочкой из пяти битов, то пространство поиска разбивается на две части A и B гиперплоскостями вида (символ $*$ обозначает произвольное значение)

$$0**** \text{ и } 1****$$

Каждое из пространств A и B разбивается на два подпространства соответственно цепочками

$$00***, 01*** (A1, A2) \text{ и } 10***, 11*** (B1, B2)$$

Геометрически это можно изобразить следующим образом (рис. 1.5).

Шаблоны, разделяющие пространство поиска, называются *схемами*. Схема – это строка длиной m , состоящая из знаков алфавита $\{0; 1; *\}$, где $\{*\}$ – неопределенный символ.

В соответствии с рис. 1.5 схемы с малым количеством закрепленных символов проверяют крупные подпространства в пространстве решений.

Например, если в хромосоме всего три бита, то возможны следующие решения:

$$000 \ 001 \ 010 \ 011 \ 100 \ 101 \ 110 \ 111$$

Количество решений равно $2^3=8$. Эти решения можно представить вершинами куба в трехмерном пространстве (рис. 1.6).

Можно описать отдельные области, в которых происходит поиск решения (т. е. схемы). Для этого используется символ $*$. При использовании символа $*$ получается общее количество решений,

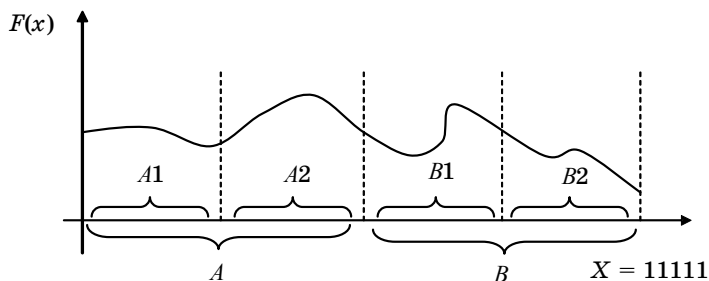


Рис. 1.5. Разбиение пространства поиска

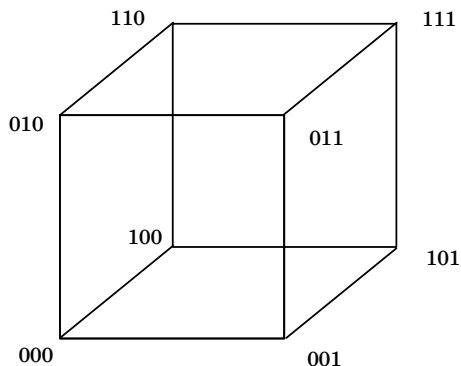


Рис. 1.6. Геометрическое представление вариантов решения задачи

равное $3^3=27$. Так как схема должна содержать хотя бы один символ *, то общее количество схем равно $27-8=19$.

Сами схемы имеют вид:

```

0**  1**  01*  ***
*0*  *1*  10*  **0
**1  *01  00*  11*
*10  0*0  1*1  1*0
*00  *11  0*1

```

В приведенном примере схема *** описывает все пространство решений, т. е. всю поверхность куба. Схемы с одним закрепленным символом описывают грани куба. Схемы с двумя закрепленными символами описывают ребра куба.

Таким образом, если конкретная хромосома – это вариант решения задачи, то схема – это целая область в пространстве решений.

В общем случае если длина цепочки равна m битов, то максимальный размер популяции оказывается 2^m . При использовании популяции максимального размера никакой ГА не нужен – достаточно протестировать все решения и выбрать лучшее из них. Однако проблема заключается в том, что значение 2^m в реальных задачах очень велико, а процедура тестирования хромосомы занимает вполне определенное время, поэтому реальный допустимый размер популяции $N \ll 2^m$.

Работа ГА управляется тремя генетическими операторами: селекцией, скрещиванием, мутацией (рис. 1.7).

В общем случае первоначальная генерация популяции длиной N осуществляется случайным образом в исследуемой области решений. После этого начинается циклическая работа ГА.

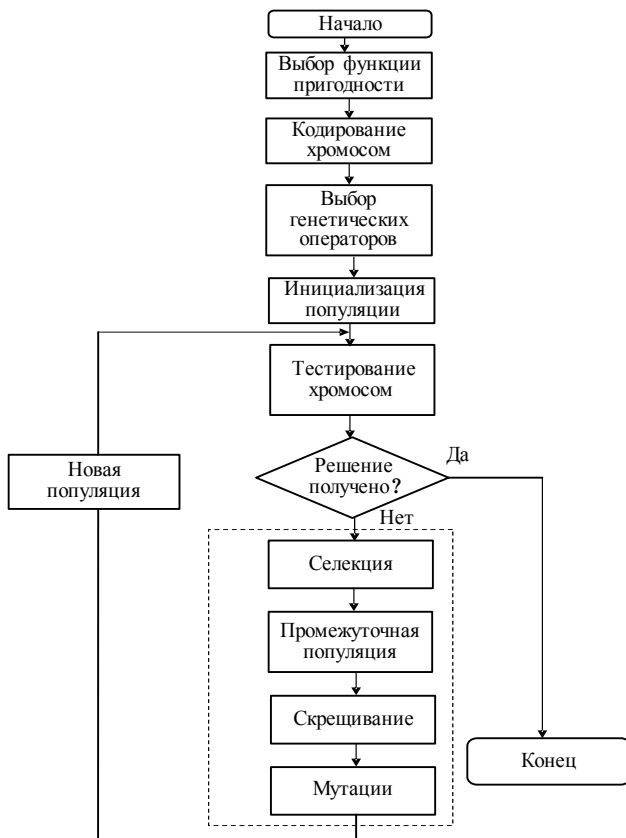


Рис. 1.7. Общая структура ГА

Селекция (другие названия – отбор, репродукция) – это первая генетическая операция, осуществляемая над популяцией. В результате селекции должны быть отобраны хромосомы, которые будут участвовать в процессе генерации новой популяции (популяции потомков). Селекция происходит на основании оценок пригодности хромосом. В итоге возникает промежуточная популяция (**mat- ing pool** или **родительский пул**). **Промежуточная популяция** — это набор особей, которые получили право размножаться. Приспособленные особи могут быть записаны туда несколько раз. «Плохие» особи с большой вероятностью туда вообще не попадут.

В классическом ГА вероятность P_i каждой i -й особи попасть в промежуточную популяцию пропорциональна отношению ее ОП к сумме ОП всех хромосом:

$$P_i = \frac{\text{ОП}_i}{\sum_{i=1}^N \text{ОП}_i}.$$

Такой способ носит название пропорциональный отбор (proportional selection). Его можно реализовать следующим образом: пусть особи располагаются на колесе рулетки так, что размер сектора S_i (в процентах) каждой особи соответствует P_i (рис. 1.8): $S_i = P_i \cdot 100\%$.

Изначально промежуточная популяция пуста. Затем рулетка запускается N раз, так что каждый раз выбирается хромосома, сектор которой оказался под указателем. В итоге выбирается N особей для записи в промежуточную популяцию. Ни одна выбранная особь не удаляется с рулетки, что позволяет ей много раз претендовать на место в промежуточной популяции.

После селекции выполняются операции скрещивания и мутации, которые обобщенно называют рекомбинацией.

Скрещивание (другие названия этой операции – пересечение, кроссовер, кроссинговер) выполняется в целях комбинирования и смешения признаков родительской популяции в популяции потомков. В классическом генетическом алгоритме применяется одноточечный оператор кроссовера: для родительских хромосом случайным образом выбирается точка раздела, и они обмениваются

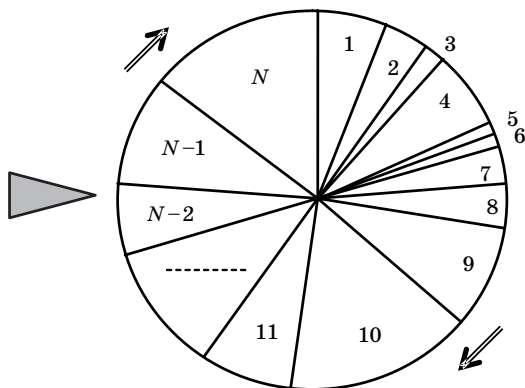


Рис. 1.8. Колесо рулетки

ся отсеченными частями. Полученные две хромосомы являются потомками.

Мутация небольшого количества генов в популяции потомков призвана сообщить потомкам новые признаки, которые могли отсутствовать в родительской популяции.

Схема ГА (см. рис.1.7) предполагает, что получено такое же количество потомков, как количество родителей, и после выполнения мутации новая популяция полностью сформирована, так что может начинаться новый цикл развития. Однако при этом существует опасность потери наилучших хромосом, поскольку все потомки могут оказаться хуже своих родителей. Для избежания этой опасности могут быть использованы следующие приемы:

- количество потомков делается меньшим количества родителей, так что замещаются только худшие родительские хромосомы;
- генерируется большее количество потомков, чем это необходимо, после чего происходит тестирование, и только лучшие хромосомы-потомки попадают в новую популяцию вместе с лучшими родителями;
- выделяется элита популяции (порядка 10%), которая попадает в новую популяцию без изменения, выделяются наихудшие хромосомы (также 10%), которые замещаются хромосомами, сгенерированными случайным образом, а остальные хромосомы подвергаются генетическим операциям.

Помимо операторов скрещивания и мутации в ГА может использоваться оператор инверсии.

1.2.4. Пример работы классического генетического алгоритма

Рассмотрим в качестве примера работы ГА процесс решения диофантова уравнения. Древнегреческий математик Диофант пытался найти ответ на вопрос: дано уравнение с целыми коэффициентами, имеет ли оно целое решение?

Пусть диофантово уравнение имеет вид

$$FD(a, b, c, d) = a + 2b + 3c + 4d = 30,$$

где a, b, c и d – некоторые положительные целые. Решение этого уравнения может быть найдено путем перебора вариантов значений a, b, c и d . Очевидно, что выполняется условие $1 \leq a, b, c, d \leq 30$. Поэтому перебор здесь потребует не более $30^4=810\,000$ вариантов. Конечно, для компьютера здесь не составляет труда найти решение прямым перебором, но при большем количестве переменных использование ГА помогает значительно сократить время поиска решения.

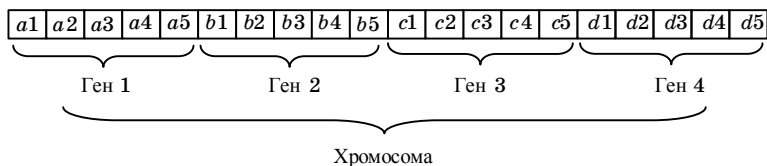


Рис. 1.9. Кодирование решения диофантова уравнения

Хромосома в рассматриваемой задаче состоит из 4-х генов: a , b , c и d . Поскольку ген является целым числом, меньшим 30, для кодирования каждого гена можно использовать 5 битов. Тогда хромосома имеет вид, показанный на рис. 1.9.

Пусть для поиска решения используется маленькая популяция из 5 хромосом. Первоначально сгенерируем 5 случайных допустимых вариантов решений:

$$F(a, b, c, d) = a + 2b + 3c + 4d, \quad a, b, c, d \in \{1, 2, 3 \dots 30\},$$

и свяжем с каждым вариантом значение ошибки решения:

$$\Delta = |F(a, b, c, d) - FD(a, b, c, d)|.$$

В табл. 1.1 показаны варианты решений (в десятичном коде).

Таблица 1.1. Первое поколение хромосом и их пригодность

i^1	Вариант (a, b, c, d)	Ошибка Δ	ОП
1	(1, 28, 15, 3)	$ 114 - 30 = 84$	0,012
2	(14, 9, 2, 4)	$ 54 - 30 = 24$	0,042
3	(13, 5, 7, 3)	$ 56 - 30 = 26$	0,038
4	(23, 8, 16, 19)	$ 163 - 30 = 133$	0,0075
5	(9, 13, 5, 2)	$ 58 - 30 = 28$	0,036

¹ i – № хромосомы.

Принцип работы ГА заключается в том, чтобы выживали хромосомы, имеющие меньшую ошибку решения. Поэтому относительную пригодность хромосомы можно описать формулой $ОП = 1/\Delta$.

Для вычисления вероятности отбора хромосомы в будущую популяцию можно использовать формулу

$$P_i = \frac{ОП_i}{\sum_{i=1}^5 ОП_i}.$$

Результаты применения этой формулы представлены в табл. 1.2.

Таблица 1.2. Вероятность отбора хромосом

i	P_i	$S_i, \%$
1	$0,012/0,135 = 0,09$	9
2	$0,042/0,135 = 0,31$	31
3	$0,038/0,135 = 0,28$	28
4	$0,0075/0,135 = 0,06$	6
5	$0,036/0,135 = 0,26$	26

Для дальнейшего выбора хромосом можно использовать метод колеса рулетки. Каждой хромосоме соответствует сектор колеса S_i (рис.1.10)

После каждого вращения колеса выбирается какой-то сектор, т. е. хромосома. После 10 вращений колеса могут быть отобраны 10 хромосом, из которых случайным образом формируются 5 пар для скрещивания. Пример показан в табл. 1.3.

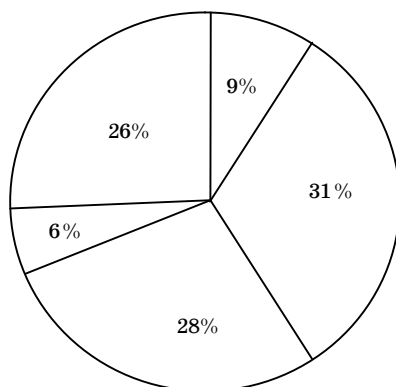


Рис. 1.10. Сектора колеса рулетки

Таблица 1.3. Результаты отбора методом колеса рулетки

i отца	i матери
3	1
5	2
3	5
2	5
5	3

В соответствии с табл. 1.3 самая плохая хромосома 4 ни разу не была отобраана для скрещивания. Хромосома 1 участвовала в скрещивании всего один раз, а хромосомы 2, 3 и 5 отбирались часто, поскольку имеют высокую ОП.

Следующая генетическая операция – скрещивание. В своем классическом варианте операция скрещивания предполагает обмен «хвостов» хромосом после случайно выбранного гена хромосомы (точки скрещивания).

Пример выполнения скрещивания приведен в табл. 1.4.

Таблица 1.4. Скрещивание хромосом

Точки скрещивания	Хромосома-отец	Хромосома-мать	Хромосома-потомок
1	(13 5, 7, 3)	(1 28, 15, 3)	(13, 28, 15, 3)
2	(9, 13 5, 2)	(14, 9 2, 4)	(9, 13, 2, 4)
3	(13, 5, 7 3)	(9, 13, 5 2)	(13, 5, 7, 2)
1	(14 9, 2, 4)	(9 13, 5, 2)	(14, 13, 5, 2)
2	(13, 5 7, 3)	(9, 13 5, 2)	(13, 5, 5, 2)

Пять хромосом-потомков образуют новую популяцию, для которой опять вычисляется ошибка (табл. 1.5).

Таблица 1.5. Новая популяция

i	Вариант (a, b, c, d)	Ошибка Δ
1	(13, 28, 15, 3)	$ 126-30 =96$
2	(9, 13, 2, 4)	$ 57-30 =27$
3	(13, 5, 7, 2)	$ 52-30 =22$
4	(14, 13, 5, 2)	$ 63-30 =33$
5	(13, 5, 5, 2)	$ 46-30 =16$

Средняя ошибка решения в популяции потомков 39, в то время как у первоначальной популяции (табл. 1.1) этот коэффициент равнялся 59.

Последней выполняется генетическая операция мутации. Она может заключаться в замене одного из генов какой-нибудь хромосомы на случайное целое число от 1 до 30. Можно также давать случайное приращение одной из переменных, входящих в решение.

Повторяя генетические операции, можно добиться получения решения, т. е. хромосомы, которой соответствует нулевая ошибка.

Увеличение размера популяции оказывает положительное влияние на работу алгоритма, однако увеличивает вычислительные затраты.

Нужно заметить, что существует множество решений рассмотренной задачи, поэтому можно конкретизировать решение, например: $a + b + c + d \rightarrow \min$.

Вопросы для самопроверки

1. В каких инженерных задачах может использоваться ГА?
2. Какой метод оптимизации является наиболее простым?
3. Чем отличается мультимодальная целевая функция от унимодальной?
4. Какие особенности объединяют различные методы глобальной оптимизации?
5. Чем отличается оптимальное решение от субоптимального?
6. Чем отличаются детерминированные методы глобальной оптимизации от стохастических?
7. В чем заключается идея метода мультистарта в глобальной оптимизации?
8. Какие физические аналогии использует метод имитации отжига?
9. Как изменяется искусственная температура в алгоритме имитации отжига?
10. Какая формула описывает количество вариантов в задаче коммивояжера для n городов?
11. Сформулируйте основные отличия ГА от других методов глобальной оптимизации.
12. Имеет ли смысл использование ГА совместно с другими методами оптимизации?
13. Что такое естественный отбор?
14. В чем заключается гипотеза пангенеза?
15. Могут ли наследоваться приобретенные признаки?
16. Почему генетика дала новый импульс дарвинизму?
17. Что представляет собой хромосома живого организма?
18. Чем отличаются половые клетки от соматических?
19. Насколько велики генетические отличия человека от высших приматов?
20. Почему у одних и тех же предков могут быть непохожие потомки?
21. Что является причиной появления новых биологических видов в соответствии с дарвиновской теорией эволюции?
22. Сформулируйте основные возражения против теории эволюции по Дарвину.
23. Что такое популяция?
24. Что такое фенотип и генотип?

25. Как называются составные части хромосомы?
26. Что такое локус и аллель?
27. Перечислите основные генетические операции.
28. Назовите дополнительные генетические операции.
29. Как называется в ГА целевая функция?
30. Что может выступать в качестве целевой функции при применении ГА в инженерных задачах?
31. Какой алфавит можно использовать при описании хромосом в инженерных задачах?
32. Как называются шаблоны, разделяющие пространство поиска?
33. Сколько схем существует в задаче, где хромосома состоит из 4-х битов?
34. Как генерируется первоначальная популяция в ГА?
35. На основании чего происходит отбор хромосом в промежуточную популяцию?
36. Какие существуют варианты для выполнения операции отбора (селекции) в классическом ГА?
37. Опишите метод колеса рулетки для отбора в промежуточную популяцию.
38. Как выполняется операция скрещивания в классическом ГА?
39. Как выполняется операция мутации в классическом ГА?
40. Как формируется хромосома при решении диофантова уравнения?
41. Как рассчитывается относительная пригодность при поиске решения диофантова уравнения?
42. Как выполняются генетические операции при решении диофантова уравнения?

2. МЕХАНИЗМЫ РАБОТЫ ГЕНЕТИЧЕСКОГО АЛГОРИТМА

2.1. Строительные блоки в генетическом алгоритме

Теория строительных блоков была предложена Д. Голдбергом [3]. Она позволяет понять механизмы видоизменения популяции под действием генетических операторов. Рассмотрим влияние операции селекции на преобразование схем популяции.

Введем обозначения:

$P(k)$ – популяция на k -м шаге (итерации);

N – мощность популяции;

ch_i – i -я хромосома;

$F(ch_i)$ – пригодность i -й хромосомы;

S – схема;

$F(ch_i^S)$ – значение ОП i -й хромосомы, включающей схему S ;

$n(S, k)$ – количество хромосом, соответствующих схеме S на k -м шаге (т. е. $n(S, k) = \text{card}\{P(k) \cap S\}$);

$F(S, k)$ – пригодность схемы k -й итерации или среднее значение пригодности хромосом из популяции $P(k)$, которые соответствуют схеме S .

Очевидно:

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

Рассмотрим суммарную пригодность хромосом из популяции $P(k)$

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

и среднее значение пригодности

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

Вероятность выбора некоторой хромосомы ch_i в родительский пул (промежуточную популяцию) $M(k)$ определяется соотношением

$$P_i = \frac{F(ch_i)}{F(k)}.$$

Следовательно, ожидаемое количество хромосом ch_i в $M(k)$ будет равно

$$N_i = NP_i = N \frac{F(ch_i)}{F(k)} = \frac{F(ch_i)}{F(k)}.$$

Ожидаемое количество хромосом со схемой S в родительском пуле

$$n(S, k+1) = \sum_{i=1}^{n(S, k)} \frac{F(ch_i^S)}{F(k)} = n(S, k) \frac{F(S, k)}{F(k)}.$$

Эта формула показывает, что при селекции происходит распространение схем с приспособленностью выше, чем средняя.

Для того чтобы оценить влияние операции скрещивания на распространение схем, введем обозначения:

$p(S)$ – порядок схемы, т. е. количество закрепленных позиций в схеме;

$d(S)$ – длина (охват) схемы – расстояние между последним и первым закрепленным символами.

Например:

$$p(01***0**1) = 4; d(01***0**1) = 9 - 2 = 7.$$

Если в схеме одна закрепленная позиция, то ее охват равен 0.

Пусть хромосома с длиной L битов соответствует схеме S , тогда вероятность того, что ни один из ее потомков не будет соответствовать этой схеме, равна

$$P_{\bar{S}} = \frac{d(S)}{L-1}.$$

(внизу стоит $L-1$, так как точка скрещивания находится внутри хромосомы).

Если P_i – вероятность выбора хромосомы в родительский пул (см. выше), то вероятность уничтожения схемы

$$P_{\bar{S}} = P_i \frac{d(S)}{L-1}.$$

Соответственно, вероятность выживания схемы

$$P_S = 1 - P_i \frac{d(S)}{L-1}.$$

Эта формула показывает, что схемы с малой длиной имеют большую вероятность выживания.

Рассмотрим влияние мутации на распространение схем в родительском пуле.

Хромосома со схемой S останется в этой схеме только тогда, когда ни один символ не мутирует. Вероятность этого события

$$P_S = (1 - P_m)^{p(S)},$$

где P_m – вероятность мутации бита. Таким образом, в процессе мутаций разрушаются схемы, имеющие высокий порядок.

Окончательно влияние генетических операторов на выживание схем можно оценить формулой (теорема схем)

$$n(S, k+1) \geq n(S, k) \frac{F(S, k)}{F(k)} \left(1 - P_i \frac{d(S)}{L-1}\right) (1 - P_m)^{p(S)},$$

таким образом, правая часть этого уравнения отражает влияние каждого из генетических операторов. Чем больше каждый из компонентов, тем чаще будет копироваться схема S в будущей популяции.

Формула показывает, что возрастает количество схем, имеющих пригодность выше средней, малый порядок и малую длину. Такие схемы и называют *строительными блоками*. Таким образом, ГА стремится достичь субоптимального решения задачи за счет комбинирования строительных блоков.

Приведенная формула описывает только некоторые аспекты поведения ГА. Она позволяет оценить процесс выживания существующих схем, но не описывает появление новых схем, которые возникают при скрещивании и мутации. По мере эволюции популяции хромосомы становятся все более похожими друг на друга так, что разрушенные схемы будут сразу же восстановлены.

2.2. Кодирование параметров в генетическом алгоритме

Обычно в ГА параметры задачи кодируются строкой битов. Однако можно также использовать действительные числа для представления генов хромосом. Особенности выполнения генетических операций для этого случая будут рассмотрены ниже.

Когда требуется кодировать вещественные числа двоичными, необходимо выполнить достаточно очевидные преобразования. Например, пусть рассматривается вещественная переменная

$$x_k \in [x_{\min}, x_{\max}].$$

Точность представления этой переменной с помощью строки битов зависит от длины этой строки. Если рассматривается бинарная строка из n битов, то расстояние между соседними значениями (шаг) можно оценить по формуле

$$\Delta x = \frac{(x_{\max} - x_{\min})}{(2^n - 1)}.$$

Маленький шаг требует использования хромосомы большой длины, что нежелательно из-за роста вычислительных затрат.

Для кодирования x_k бинарной строкой s_k нужно использовать формулу

$$s_k = (2^n - 1) \frac{(x_k - x_{\min})}{(x_{\max} - x_{\min})}.$$

Так, допустим, что $x_k \in [20, 100]$ и $n=8$, тогда для $x_k=56$ имеем

$$s_k = (2^8 - 1) \frac{(56 - 20)}{(100 - 20)} = 255_{10} \cdot 0,45_{10} \approx 1111111_2 \cdot 0,0111_2 = 01110010_2.$$

Если же требуется получить действительное значение для некоторого двоичного кода, то выполняется обратный перевод:

$$x_k = s_k \frac{(x_{\max} - x_{\min})}{(2^n - 1)} + x_{\min}.$$

Например, для кода $s_k = 00111100_2 = 60_{10}$ получается значение переменной

$$x_k = 60 \frac{(56 - 20)}{(256 - 1)} + 20 = 28,4.$$

Использование двоичного алфавита как имеющего самое малое количество символов имеет следующие особенности.

С одной стороны, двоичный алфавит приводит к использованию хромосом максимальной длины. Тем самым получается максимальное количество гиперплоскостей, что обеспечивает лучший охват пространства поиска. В бинарном алфавите для встречаемости каждого символа в каждой позиции требуется меньший размер популяции.

С другой стороны, для того чтобы в текущей популяции могло сосуществовать достаточное количество разнообразных схем, размер популяции должен быть достаточно большим. Примерная оценка размера популяции составляет $10N$ (где N – длина хромосомы в битах) [18].

Таким образом, точность решения задачи требует использования длинных хромосом. Но при этом неизбежно растет размер популяции и время решения задачи.

Обычный двоичный код отличается тем, что в нем два близких числа могут иметь разные значения битов во всех позициях (т. е.

имеют большое расстояние по Хэммингу). Например, числа 7 и 8 различаются на 4 бита: $0111_2 = 7_{10}$; $1000_2 = 8_{10}$.

Мутация старшего бита, например, превращает 7 в 15, а 8 в 0. Во многих задачах могут быть нежелательны такие резкие изменения, сразу уводящие решение в другую область поиска. Поэтому обычный двоичный код может быть заменен на код Грея, в котором любые два соседних числа отличаются значением только одного бита (табл. 2.1).

Таблица 2.1. Сравнение кода Грея и двоичного кода

Десятичное число	Двоичный код	Код Грея
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Таким образом, при кодировании целочисленных признаков нужно разбить число на тетрады (группы по 4 бита), а каждую тетраду записать в коде Грея.

Двоичная хромосома может иметь большую длину, в такой ситуации полезным может оказаться логарифмическое кодирование [12]. При логарифмическом кодировании первый бит кодовой последовательности (α) – это бит знака показательной функции, второй бит (β) – бит знака степени этой функции, а остальные биты (bin) кодируют значение степени:

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

где запись $[bin]_{10}$ означает, что двоичную последовательность надо преобразовать в десятичное число, например:

$$[10110] = (-1)^0 e^{(-1)^1 [110]_{10}} = e^{-6} = 0,0025.$$

Таким образом, при 5 битах логарифмического кода представимые числа принадлежат диапазону

$$x \in [e^{-7}, e^7] = [0,0009, 1097].$$

Это намного больше, чем диапазон $[0, 31]$, который обеспечивается обычным двоичным кодом.

2.3. Варианты оценки пригодности хромосом

Тестирование хромосом является важнейшим этапом работы ГА. В результате тестирования каждая хромосома получает фиксированный на данном шаге признак – относительную пригодность. Чаще всего ОП описывается численно. Расчет ОП зависит от решаемой конкретной задачи.

Рассмотрим, например, проблему идентификации (настройки параметров) имитационной модели (ИМ). На вход объекта и модели подается одинаковый тестовый сигнал $x(t)$. Задача заключается в подборе таких параметров модели P , при которых выход модели $y(t)$ как можно более близок выходу объекта $z(t)$. Популяция хромосом кодирует различные варианты параметров модели (рис. 2.1).

Вместо непрерывных выходов $y(t)$ и $z(t)$ обычно рассматривается набор из n обучающих пар, каждая из которых соответствует выходу объекта и модели в дискретные моменты времени. Тогда ошибка моделирования может быть определена в виде

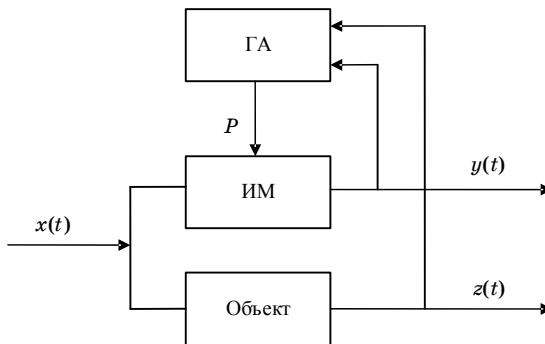


Рис. 2.1. Настройка имитационной модели

$$\varepsilon = \sum_{i=1}^n (y_i - z_i)^2 \text{ или } \varepsilon = \sum_{i=1}^n |y_i - z_i|$$

и ОП рассчитывается по формуле

$$\text{ОП} = \frac{1}{1 + \varepsilon}. \quad (2.1)$$

Таким образом, с каждой хромосомой связывается численная оценка. Если эта оценка используется без изменения в операции селекции, то говорят о пропорциональном присвоении ОП.

Для анализа текущего состояния популяции могут быть введены следующие критерии:

- *селективное давление* – отношение ОП наилучшей хромосомы к средней ОП всей популяции;

- *селективное отклонение* (bias) – абсолютная разность между нормализованной ОП хромосомы и вероятностью ее участия в воспроизводстве;

- *селективная распространенность* (spread) – ранжирование количества возможных потомков одной хромосомы;

- *потеря разнообразия* (loss of diversity) – процент хромосом популяции, которые отбрасываются и не участвуют в дальнейшей селекции.

При **ранговой** селекции популяция сортируется в соответствии со значениями ОП, которая каждой хромосоме присваивается только в зависимости от ее положения (ранга) в отсортированной популяции, а не от абсолютного значения ОП (т. е. целевой функции). Это позволяет контролировать селективное давление и ограничить количество потомков одной хромосомы.

Ранговая селекция позволяет преодолеть проблему масштаба, существующую при использовании пропорциональной селекции. Если ОП хромосом сильно отличаются друг от друга, то это может вызывать такие негативные явления как стагнация при низком селективном давлении или преждевременное сужение зоны поиска (попадание в локальный минимум).

Ранговая селекция предполагает дополнительную сортировку хромосом популяции и пересчет ОП, полученной на основании целевой функции, так что новое значение ОП зависит только от положения (ранга) хромосомы.

Введем обозначения: N – размер популяции, S – селективное давление:

$$S = \frac{\text{ОП}_{\max}}{\text{ОП}_{\text{avg}}}.$$

Пусть все хромосомы отсортированы таким образом, что номер 1 имеет хромосома с наименьшим значением ОП_{\min} , а номер N – хромосома с ОП_{\max} .

Тогда линейное ранжирование всех хромосом может быть выполнено с помощью формулы

$$\text{ОП}'_i = 2 - S + 2(S - 1) \frac{(i - 1)}{(N - 1)}. \quad (2.2)$$

Рассмотрим пример. Пусть отсортированные значения ОП хромосом популяции представлены вектором (рис. 2.2):

$\text{ОП} = [55, 55, 62, 68, 68, 68, 70, 78, 78, 80, 82, 96, 97, 98, 102, 103, 120, 121, 125, 138]$.

Выполнить линейное ранжирование (2.2) можно с помощью следующего m-файла MatLab:

```
K=size(X)
[B,m2]=max(X)
S=m2/mean(X)
for i=1:K(2)
    Y(i)=2-S+2*(S-1)*(i-1)/(K(2)-1)
end
plot(Y, '+' )
```

Результат линейной ранговой селекции хромосом показан на рис. 2.3.

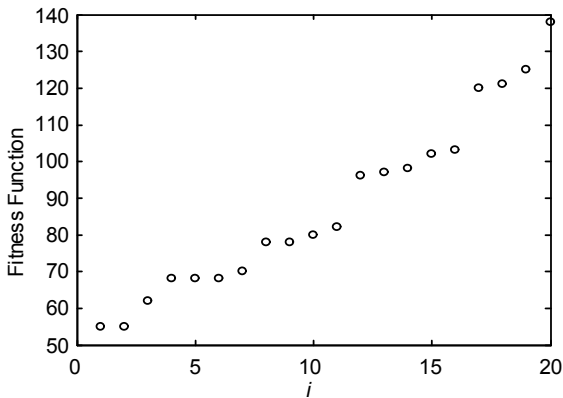


Рис. 2.2. Упорядоченный фитнес хромосом

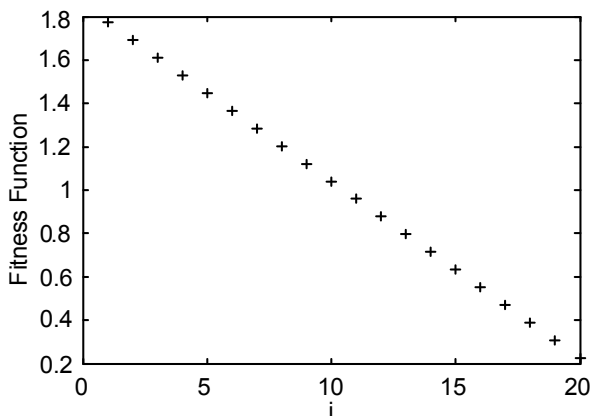


Рис. 2.3. Линейная ранговая селекция

Линейная ранговая селекция позволяет обеспечить условие $S \in [1, 2]$.

Могут быть также предложены формулы для нелинейного ранжирования, с помощью которых может быть повышено селективное давление (см. п. 4.2.5).

Операция оценивания пригодности является основополагающей для эффективной работы ГА.

2.4. Операция селекции

2.4.1. Варианты стохастического отбора

Операция селекции предполагает выбор хромосом, которые составят основу промежуточной популяции. Простейшим методом селекции является рассмотренный метод колеса рулетки (*стохастический отбор с заменой* – stochastic sampling).

Стохастический отбор с заменой можно описать и без использования аналогии с колесом рулетки.

1. Хромосомы отображаются в соприкасающиеся отрезки прямой. Длина каждого отрезка соответствует нормализованной пригодности хромосом или, что то же, вероятности выбора хромосомы, ее можно рассчитать по формуле

$$P_i = \frac{ОП_i}{\sum_{i=1}^N ОП_i}. \quad (2.3)$$

2. Генерируется случайное число $k \in [0, 1]$ и отбирается хромосома, в отрезок которой попало это число.

3. Процесс повторяется до тех пор, пока не будет отобрано количество хромосом, достаточное для создания промежуточной популяции.

Метод колеса рулетки обеспечивает нулевое смещение, т. е. вероятность участия хромосомы в воспроизводстве равна нормализованной ОП хромосомы. Однако при этом не ограничивается селективная распространенность.

Например, пусть рассматривается задача настройки ИМ. Имеется эталонный процесс $F^*(t)$ и процесс $F(t)$, который получается при некоторых параметрах ИМ, кодируемых хромосомой. Тогда для каждой хромосомы можно найти ошибку вида

$$\Delta_i = \sum_{i=1}^N |F^*(t_i) - F(t_i)|,$$

где N — множество рассматриваемых моментов времени.

Рассмотрим метод колеса рулетки. Здесь можно положить

$$F_i = \frac{1}{\Delta_i},$$

и вероятность отбора хромосомы вычисляется в соответствии с (2.3).

Допустим, что для кодирования решения используется строка из 8 битов. На рис. 2.4 показан вариант популяции из 7 хромосом, с каждой хромосомой связана ошибка Δ_i , по которой можно найти F_i и P_i .

Для отбора в родительский пул генерируется 7 случайных чисел в диапазоне от 0 до 1. На рис. 2.5 показано выполнение стохастического отбора (где стрелка соответствует значению случайного числа, а цифра — номер операции выбора).

Таким образом, формируется промежуточная популяция (рис. 2.6).

Здесь лучшая хромосома с номером 4 копируется в промежуточную популяцию несколько раз, в то время как хромосомы с номерами 3 и 7, имеющие низкую пригодность, не попадают в промежуточную популяцию вообще (хотя они имели на это шансы).

Другой способ отбора, который также является пропорциональным, называется *стохастический отбор с остатком* (remainder stochastic sampling). Для каждой особи вычисляется отношение ее приспособленности к средней приспособленности популяции. Целая часть этого отношения указывает, сколько раз нужно записать

	Исходная популяция								Δ_i	$ОП_i$	P_i	$ОП_i / ОП_{ср}$
x_1	1	1	1	0	0	1	0	1	17	0,06	0,14	1
x_2	0	1	0	1	0	1	0	0	128	0,008	0,02	0,13
x_3	0	0	1	0	1	0	1	1	169	0,006	0,015	0,1
x_4	1	1	0	1	0	0	0	0	4	0,25	0,6	4,16
x_5	1	0	1	1	0	0	0	0	36	0,03	0,07	0,5
x_6	1	1	0	0	0	1	0	0	16	0,06	0,14	1
x_7	0	1	0	0	0	1	0	1	143	0,007	0,016	0,11

Рис. 2.4. Пример вычисления вероятностей при стохастическом отборе

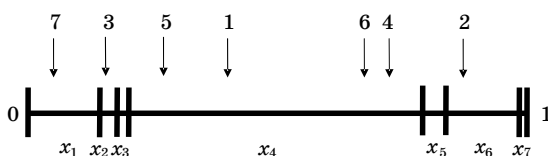


Рис. 2.5. Выполнение стохастического отбора

	Промежуточная популяция							
x_4	1	1	0	1	0	0	0	0
x_4	1	1	0	1	0	0	0	0
x_4	1	1	0	1	0	0	0	0
x_4	1	1	0	1	0	0	0	0
x_1	1	1	1	0	0	1	0	1
x_6	1	1	0	0	0	1	0	0
x_2	0	1	0	1	0	1	0	0

Рис. 2.6. Промежуточная популяция при стохастическом отборе

особь в промежуточную популяцию, а дробная — это ее вероятность попасть туда еще раз. Пусть для некоторой особи i

$$\frac{ОП_i}{ОП_{ср}} = 1,12.$$

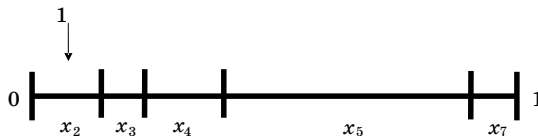


Рис. 2.7. Стохастический выбор с остатком

Тогда она будет выбрана один раз, а затем с вероятностью $0,12$ – еще раз. Реализовать такой способ отбора удобно следующим образом: расположим хромосомы, как было показано на рис. 2.5. Теперь пусть у рулетки не одна стрелка, а N стрелок, причем они отсекают одинаковые сектора. Тогда один запуск рулетки выберет сразу все N особей, которые нужно добавить в промежуточную популяцию.

Если использовать данные рис. 2.4, то оказывается, что хромосома с номером 4 должна детерминированно попасть в родительский пул 4 раза, а хромосомы с номерами 1 и 6 – по разу. Таким образом, остается выбрать всего одну хромосому (рис. 2.7).

В этом методе, очевидно, число потомков одной хромосомы ограничено.

2.4.2. Турнирная селекция

Турнирный отбор может быть описан следующим образом: из популяции, содержащей N хромосом, выбирается случайным образом t хромосом (тур), и лучшая хромосома из тура попадает в родительскую популяцию, т. е. между выбранными хромосомами проводится турнир. Эта операция повторяется N раз.

Размер группы хромосом, отбираемых для турнира, часто равен 2. В этом случае говорят о *двоичном/парном турнире* (binary tournament). Вообще же величина t называется *численностью турнира* (tournament size). Чем больше турнир, тем более жесткий вариант селекции, т. е. тем меньше шансов у особей с низкой пригодностью.

Преимуществом турнирной селекции является то, что она не требует дополнительных вычислений и упорядочивания строк в популяции по возрастанию приспособленности.

2.4.3. Селекция отсечением

В этом способе селекции хромосомы сортируются в соответствии со значением ОП. В родительскую популяцию отбираются только лучшие хромосомы, у которых ОП превышает заданный порог. Остальные хромосомы отбрасываются.

2.4.4. Селекция генов

Описанные выше варианты предполагают, что в селекции участвуют хромосомы популяции целиком. Эти подходы примерно соответствуют формуле «структура хромосом родительского пула соответствует структуре ОП».

Между тем возможна методика выполнения селекции как операции, выполняемой не над хромосомами, а над генами популяции. Впервые вариант этого подхода описан в работе [37]. Таким образом, здесь идет речь о формуле «структура признаков родительского пула соответствует структуре ОП».

Рассмотрим, как можно математически описать общую схему селекции признаков популяции. Будем считать, что один признак кодируется одним битом.

Введем обозначения: N – мощность популяции; m – длина хромосомы в битах; \mathbf{X}_i – i -я хромосома; $F(\mathbf{X}_i)$ – пригодность i -й хромосомы.

Тогда средняя пригодность хромосом популяции

$$F_{\text{cp}} = \frac{\sum_{i=1}^N F(\mathbf{X}_i)}{N}.$$

Пусть j – номер признака и $\mathbf{X}_i(j=1)$ – хромосома, у которой есть j -й признак. Тогда количество хромосом с j -м признаком

$$M_j = \text{card}\{\mathbf{X}_i(j=1)\}$$

и средняя пригодность хромосом с j -м признаком

$$F_{\text{cp}}^j = \frac{\sum_{i=1}^{M_j} F(\mathbf{X}_i(j=1))}{M_j}. \quad (2.4)$$

Тогда предпочтительность отбора j -го признака в популяцию потомков определяется формулой

$$G_j = \frac{F_{\text{cp}}^j}{F_{\text{cp}}}. \quad (2.5)$$

Эта величина, в отличие от вероятности, может превышать единичное значение. Следовательно, новое количество хромосом, которые будут иметь j -й признак, можно оценить по формуле

$$M_j^H = \min \{N, M_j G_j\}. \quad (2.6)$$

Для того чтобы избежать преждевременной сходимости, можно ограничить число хромосом с j -м признаком:

$$M_j^H = \min \{Nk, M_j G_j\},$$

где константа $k \in [0, 1]$.

Возникает вопрос: как распределять признак на основании полученной оценки между хромосомами популяции потомков? Здесь нет смысла говорить о номерах хромосом, важно лишь их количество, поэтому можно положить

$$X_i(j=1), \quad i = \overline{1, M_j^H} \quad \text{и} \quad X_i(j=0), \quad i = \overline{M_j^H + 1, N}.$$

Подобные операции продолжают по всем признакам популяции.

Естественно, при таком способе важно основательно перемешать признаки, так как первые хромосомы будут тяготеть к заполнению единицами, а последние – нулями. Поэтому операция скрещивания должна быть многоточечной (см. ниже).

Рассмотрим далее, как можно выполнить отбор признаков для той же популяции, что была показана на рис. 2.4.

Вариант вычисления параметров для отбора признаков показан на рис. 2.8, при этом использовались формулы (2.4) – (2.6).

При вычисленных значениях промежуточная популяция приобретет вид, показанный на рис. 2.9.

Сравнение промежуточных популяций, полученных при стохастическом отборе и при отборе признаков, показывает, что они имеют примерно одинаковое количество однотипных схем длиной 2. Можно сказать, что при небольшом размере популяции метод отбора признаков не приносит особых преимуществ.

Как показали проведенные вычислительные эксперименты [37, 38], описанная схема отбора признаков в новую популяцию позволяет в среднем значительно ускорить сходимость ГА при попадании в перспективную область решения. Это особенно важно в задачах, где хромосома имеет значительную длину (порядка 100 битов и более). При такой длине хромосом популяция должна содержать тысячи особей, и вопросы увеличения скорости вычислений становятся особенно острыми.

Скорость сходимости является важнейшим параметром ГА, поэтому описанную схему работы можно назвать «быстрым генетическим алгоритмом». Правда, здесь существует опасность преждевре-

Исходная популяция – отбор признаков								$F_{\text{ср}}^i$	G_i	M_i^H
1	1	0	0	1	1	1	0	0,1	1,66	7
2	1	1	0	1	0	1	1	0,08	1,33	7
3	1	0	1	0	1	0	0	0,03	0,53	2
4	0	1	0	1	1	0	0	0,097	1,6	5
5	0	0	1	0	0	0	0	0,006	0,1	0
6	1	1	0	0	0	1	1	0,034	0,56	2
7	0	0	1	0	0	0	0	0,006	0,1	0
8	1	0	1	0	0	0	1	0,024	0,4	1
$X_1 \quad X_2 \quad X_3 \quad X_4 \quad X_5 \quad X_6 \quad X_7$										

Рис. 2.8. Пример вычисления параметров при отборе признаков

Промежуточная популяция – отбор признаков								
X_1	1	1	1	1	0	1	0	1
X_2	1	1	1	1	0	1	0	0
X_3	1	1	0	1	0	0	0	0
X_4	1	1	0	1	0	0	0	0
X_5	1	1	0	1	0	0	0	0
X_6	1	1	0	0	0	0	0	0
X_7	1	1	0	0	0	0	0	0

Рис. 2.9. Промежуточная популяция при отборе признаков

менной сходимости алгоритма. Для борьбы с этим явлением можно использовать механизм ранжирования признаков, подобный известному механизму ранжирования хромосом, описанному выше.

Отбор признаков в новую популяцию является, очевидно, операцией, разрушающей хромосомы. Однако это не противоречит теореме о строительных блоках, в соответствии с которой при селекции происходит распространение коротких схем. Кроме того, отбор признаков можно совмещать со стратегией элитизма, когда малое количество (порядка 10%) лучших хромосом отбирается в популя-

цию потомков без изменений. Еще 10% хромосом могут быть полностью случайными.

Таким образом, популяция потомков оказывается состоящей из трех частей, для которых смысл операций скрещивания и мутации должен отличаться (табл. 2.2).

Таблица 2.2. Популяция потомков и генетические операции

Структурная часть популяции	Размер, %	Скрещивание	Мутация
Элита (лучшие хромосомы родительской популяции)	10	Одноточечное	Одна мутация на сотню признаков
Отобранные признаки	80	Многоточечное	
Случайные хромосомы	10	Нет	Нет

При наличии случайной части в популяции уже нет смысла говорить о том, что с помощью мутации можно охватить все пространство поиска. Мутация просто может несколько улучшить одно из существующих решений.

«Быстрый ГА» больше соответствует биологическим представлениям о механизме эволюции, поскольку эволюционирует во времени популяция организмов, а не отдельные живые организмы.

2.5. Варианты операции скрещивания

2.5.1. Бинарный кроссовер

Генетическая операция *скрещивания* объединяет две исходные (родительские) хромосомы для получения одной или двух новых хромосом-потомков.

На языке теории множеств операцию скрещивания можно описать так. Пусть имеются два множества A и B ($A \subset U$, $B \subset U$). Случайно выбирается $S \subset U$. Порождаются новые объекты (обмениваются частями, попавшими в S):

$$A' = A - A \cap S + B \cap S; B' = B - B \cap S + A \cap S.$$

Рассмотрим кроссовер для варианта, когда хромосомы представляют собой цепочку битов.

Одноточечное скрещивание выполняется следующим образом:

- на первом шаге происходит случайный выбор пары хромосом;
- на втором шаге случайно выбирается n – точка скрещивания.

Если генами являются отдельные биты, то n является номером бита по длине хромосомы, иначе n – номер гена;

- на третьем шаге происходит сама операция, которая заключается в переписывании генов одной хромосомы в другую хромосому и наоборот, начиная с точки скрещивания.

Таким образом, однотоочечное скрещивание рассматривается просто как обмен «хвостами» хромосом, начиная с некоторой случайной позиции – точки скрещивания. Пример показан на рис. 2.10.

Многоточечное скрещивание предполагает выбор нескольких несовпадающих точек скрещивания. Хромосомы обмениваются «хвостами» после каждой точки скрещивания.

Например, пусть случайно выбраны две хромосомы

$$X = (000111010011) \text{ и } Y = (111000001101).$$

Пусть случайно выбраны позиции скрещивания $n1=3$, $n2=7$ и $n3=10$, тогда после скрещивания получаются хромосомы-потомки

$$X' = (000000010001) \text{ и } Y' = (111111001111).$$

Многоточечное скрещивание может быть эффективно при большой длине хромосом.

Операции однотоочечного и многоточечного скрещивания иллюстрирует рис. 2.11.

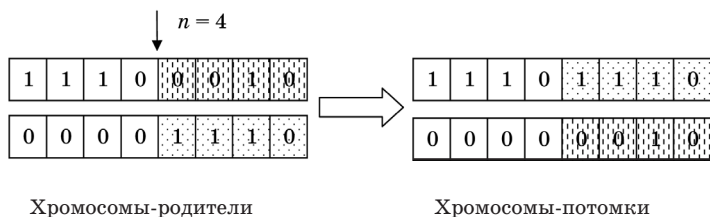


Рис. 2.10. Однотоочечный кроссовер

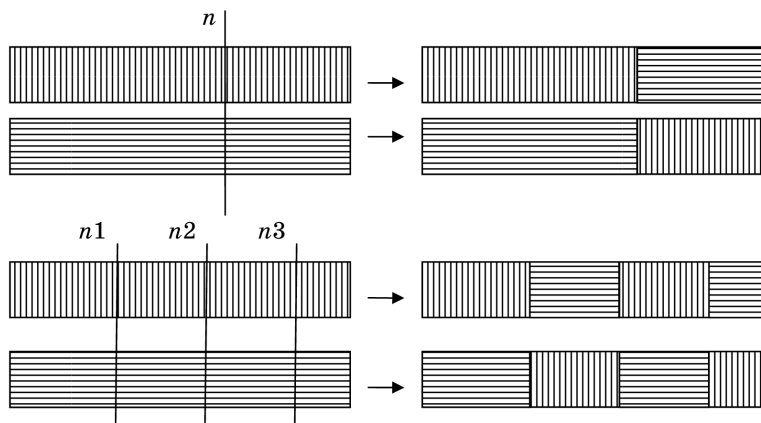


Рис. 2.11. Однотоочечный и многоточечный кроссоверы

2.5.2. Всеобщий кроссовер (uniform crossover)

В этом способе кроссовера каждая позиция по длине хромосомы может быть потенциальной точкой скрещивания. Здесь также случайным образом выбираются две родительские хромосомы, затем для каждого потомка генерируется случайным образом бинарный вектор маски, длина которого совпадает с длиной хромосомы. Нулевое значение бита вектора маски указывает на то, что значение бита потомка должно быть взято из первой родительской хромосомы, единичное значение – из второй родительской хромосомы. Рассмотрим пример.

Пусть выбраны две родительские хромосомы

X: 00011000011; Y: 11010111000

и сгенерирован вектор маски

M: 01000011011

Тогда получается хромосома-потомок

P: 01011011000

Рассмотрим скрещивание при представлении генов действительными числами. Здесь могут использоваться варианты дискретного, промежуточного и линейного скрещивания.

Дискретное скрещивание напоминает многоточечное двоичное скрещивание. Рассмотрим пример.

Пусть имеются две хромосомы, состоящие каждая из 3-х генов:

D = (23 4 68) и S = (112 45 2).

Дважды (для каждого из потомков) случайно генерируется вектор скрещивания, компоненты которого указывают либо на первую, либо на вторую хромосому:

V1 = (1 1 2) и V2 = (2 1 2).

Хромосомы-потомки получают следующие значения:

D' = (23 4 2) и S' = (112 4 2).

В приведенном примере существует, очевидно, восемь вариантов для порождения каждого потомка.

В общем случае (при произвольной длине хромосом) можно сказать, что при дискретном скрещивании потомки могут занимать вершины гиперкуба, заданного параметрами (генами) родителей. Этот тезис для двумерного случая иллюстрирует рис. 2.12, а

Промежуточное скрещивание может использоваться только при кодировании генов хромосом действительными числами. Для нахождения каждого параметра потомков применяется формула

$$P_i = x_i + \alpha |y_i - x_i|,$$



Рис. 2.12. Скрещивание: а – дискретное; б – линейное: \circ – возможные потомки; \square – родители

где x_i и y_i – параметры родительских хромосом; α – масштабирующий коэффициент ($\alpha \in [0, 1]$), выбирается каждый раз заново для каждого параметра.

Рассмотрим пример. Пусть имеются две хромосомы, состоящие каждая из 3-х генов:

$$\mathbf{D} = (23 \ 4 \ 68); \mathbf{S} = (112 \ 45 \ 2),$$

и были случайно выбраны следующие α :

$$\alpha_d = (0,5 \ 0,1 \ 0,8); \alpha_s = (0,4 \ 0,7 \ 0,2).$$

Хромосомы-потомки принимают значения

$$\mathbf{D}' = (67,5 \ 8,1 \ 121); \mathbf{S}' = (58,6 \ 33 \ 81,2).$$

Таким образом, при промежуточном скрещивании хромосомы-потомки могут занимать не только вершины гиперкуба, заданного параметрами хромосом-родителей, но и промежуточные точки внутри этого гиперкуба.

Если выбрать одинаковое значение α для всех параметров хромосом, то промежуточное скрещивание превращается в *линейное* – параметры потомков принимают значения на прямой, соединяющей параметры родительских хромосом (рис. 2.12, б).

Операции скрещивания выполняются над всей популяцией. Каждая хромосома участвует в скрещивании только один раз.

2.6. Операция мутации

Операция мутации заключается в случайном выборе одной хромосомы популяции и случайном изменении одного из ее генов.

В случае, если гены хромосом принимают значения на множестве действительных чисел, операцию мутации для случая двух переменных описывает рис. 2.13.

Максимальная величина шага мутации зависит от размера области определения соответствующего параметра.

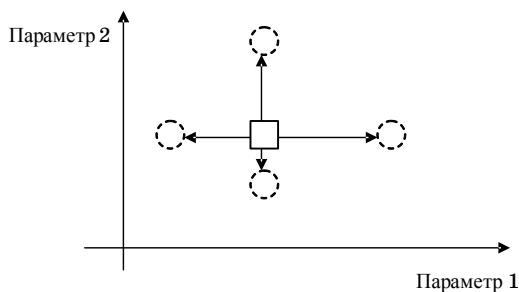


Рис. 2.13. Выполнение мутации: ○ – возможные потомки; □ – родитель

При двоичном кодировании хромосом оператор мутации меняет некоторую позицию хромосомы случайным образом. Вероятность мутации весьма мала, например одна мутация гена на сотню хромосом.

2.7. Миграционная модель генетического алгоритма

Миграционная (или островная) модель ГА предполагает существование в популяции множества субпопуляций.

Здесь также используются биологические аналогии. Представим, что имеется группа островов, на которых живут популяции особей одного вида. Эти популяции развиваются независимо друг от друга, и только изредка оказывается возможным обмен особей между популяциями, например при резком понижении уровня водной поверхности. Островная модель ГА использует описанный принцип для поиска решения.

Суть миграционной модели заключается в том, что в разных субпопуляциях могут развиваться разные доминирующие признаки.

Нетрудно заметить, что при слишком активной миграции островная модель не будет отличаться от обычного ГА.

Миграционная модель ориентирована на использование компьютерной сети для повышения производительности вычислений.

Каждая субпопуляция развивается независимо некоторое число поколений (время изоляции). По истечении времени изоляции некоторое число индивидуумов (хромосом) мигрируют между субпопуляциями. Их количество определяет скорость миграции.

Выбор мигрирующих хромосом может быть случайным или основанным на значении ОП. В последнем случае мигрируют лучшие хромосомы. Направление миграции может выбираться в соответс-

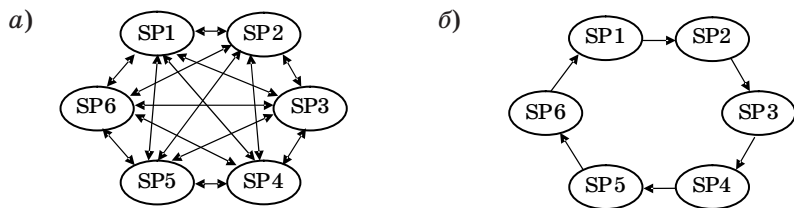


Рис. 2.14. Полносвязная (а) и кольцевая (б) топология миграции

твии с полностью или кольцевой топологией (рис. 2.14, а, б, где SP – субпопуляция).

Мигрирующие хромосомы заменяют худшие хромосомы популяции, в которую попадают.

Могут быть предложены и другие топологии миграции.

Вопросы для самопроверки

1. Что такое порядок схемы в ГА?
2. Что такое длина схемы в ГА?
3. Какие схемы распространяются при селекции?
4. Какие схемы разрушаются при мутации?
5. Что такое строительный блок в ГА?
6. Какие преимущества дает использование двоичного алфавита при описании хромосом?
7. Какие проблемы возникают при двоичном кодировании хромосом?
8. Можно ли использовать действительные числа для представления генов хромосом?
9. Как закодировать действительную переменную строкой битов?
10. Как преобразовать строку битов в действительную переменную?
11. В чем преимущество использования кода Грея по сравнению с обычным двоичным кодом?
12. Как выполняется логарифмическое кодирование двоичной хромосомы?
13. Какой формулой можно описать ОП хромосомы при настройке имитационной модели?
14. Что такое селективное давление?
15. Что такое селективное отклонение?
16. Что такое селективная распространенность?
17. Что такое потеря разнообразия?

18. Чем отличается ранговая селекция от пропорциональной селекции?
19. По какой формуле выполняется линейное ранжирование хромосом?
20. Чем отличается стохастический отбор с заменой от стохастического отбора с остатком?
21. Что такое турнирная селекция?
22. Что такое селекция отсечением?
23. Опишите механизм селекции генов как вариант операции отбора.
24. Как описывается операция скрещивания на языке теории множеств?
25. Как выполняется всеобщий кроссовер?
26. Опишите варианты скрещивания при представлении генов действительными числами.
27. Как выполняется операция мутации при представлении генов действительными числами?
28. Какие достоинства несет использование миграционной модели ГА?
29. Какие топологии миграции применяются в ГА?

3. ПРИЛОЖЕНИЯ ГЕНЕТИЧЕСКОГО АЛГОРИТМА

3.1. Генетический синтез регуляторов

3.1.1. Общие принципы синтеза регуляторов

Существуют две группы методов синтеза регуляторов – аналитические и экспериментальные.

Аналитические методы опираются на «точное» описание объекта управления, заданное в виде дифференциальных уравнений (уравнений состояния) или передаточной функции. Однако эти методы требуют идеализации объекта управления, чтобы отнести его к некоторому общему классу, для которого известны стандартные алгоритмы синтеза управления. Аналитические методы хорошо разработаны для объектов управления, обнаруживающих линейные свойства.

Экспериментальные методы предполагают реализацию некоторой процедуры оптимизации параметров регулятора, во время которой процессы регулирования анализируются либо с помощью модели, либо прямо на объекте управления. Эти методы более универсальны, так как не накладывают дополнительных требований на математическое описание объекта управления, позволяя использовать при моделировании нелинейности, эвристики и т. д.

Генетический алгоритм, как универсальный метод оптимизации, может быть использован и для коррекции параметров регулятора.

Общая схема настройки регулятора с помощью ГА в режиме реального времени (on line) показана на рис. 3.1, она предполагает,

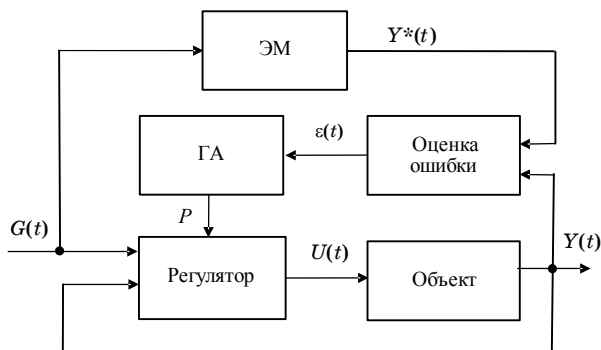


Рис. 3.1. Настройка параметров регулятора в режиме on line

что существует эталонная модель (ЭМ), приближенно описывающая желаемую реакцию объекта $Y^*(t)$ на любое входное воздействие $G(t)$. ЭМ может быть намного проще реального объекта, например, это может быть динамическое звено невысокого порядка.

Альтернативные варианты параметров регулятора P кодируются с помощью хромосом.

Для получения оценки пригодности каждой хромосомы нужно выполнить моделирование переходного процесса с параметрами регулятора P для заданного $G(t)$.

Входным параметром ГА является ошибка функционирования

$$\varepsilon(t) = Y^*(t) - Y(t),$$

где $Y(t)$ – реальный выход объекта при текущих параметрах регулятора.

На основании значения $\varepsilon(t)$ строятся оценки качества управления, т. е. пригодности каждой хромосомы.

Таким образом, закон управления синтезируется с помощью ГА в результате многократных экспериментов с объектом. Однако очевидно, что для большинства реальных объектов управления недопустимо выполнение такого огромного количества экспериментов, которое требуется при работе ГА. К тому же время проведения экспериментов должно быть ограничено. Поэтому в реальности вместо объекта в структуре рис. 3.1 должна присутствовать его ИМ, к которой предъявляются требования адекватности и быстродействия. Конкретизируя рис. 1.4, можно показать схему настройки регулятора следующим образом (рис. 3.2).

При хорошей работе регулятора выходы ИМ и ЭМ должны быть близки, поэтому задача настройки регулятора ставится, как задача минимизации некоторой функции расстояния, например:

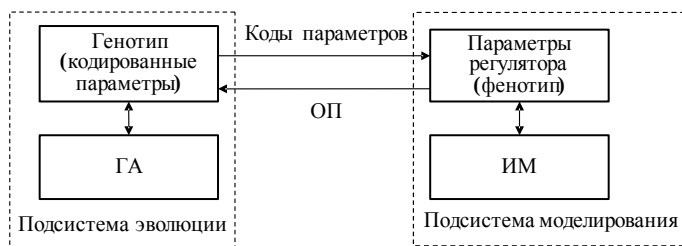


Рис. 3.2. Настройка параметров регулятора в режиме *off line*

$$J = \sum_{i=1}^N (Y_i^* - Y_i)^2 \rightarrow \min; \quad (3.1)$$

$$J = \sum_{i=1}^N |Y_i^* - Y_i| \rightarrow \min, \quad (3.2)$$

где N – количество рассматриваемых моментов времени в течение переходного процесса. Формулы (3.1) и (3.2) поясняет рис. 3.3.

Если объект имеет векторный выход (m выходных переменных), то могут использоваться формулы

$$J = \sum_{j=1}^m \sum_{i=1}^N k_j (Y_i^* - Y_i)^2 \rightarrow \min; \quad (3.3)$$

$$J = \sum_{j=1}^m \sum_{i=1}^N k_j |Y_i^* - Y_i| \rightarrow \min, \quad (3.4)$$

где k_j – масштабный коэффициент, с помощью которого уравнивается влияние разных выходных переменных.

Относительная пригодность хромосомы для всех вариантов может оцениваться по формуле, подобной (2.1):

$$F = \frac{1}{J + 1}. \quad (3.5)$$

Кроме критериев (3.1) – (3.4) могут быть использованы такие традиционные критерии качества переходного процесса как запас устойчивости, время переходного процесса, перерегулирование и т. д.

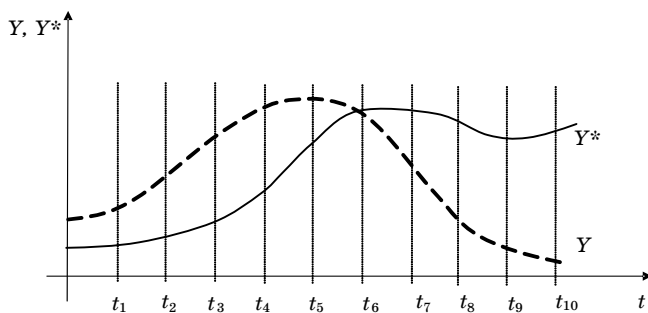


Рис. 3.3. Сравнение эталонного (–) и реального (–) процессов

Таким образом, при использовании ГА для настройки регулятора динамического объекта требуется решить следующие проблемы.

1. Выбор настраиваемых параметров регулятора. Параметры регулятора (фенотип) кодируются с помощью хромосомы, так что длина хромосомы пропорциональна количеству параметров. ГА работает быстрее при малой длине хромосом, поэтому желательно минимизировать количество настраиваемых параметров и диапазон их изменения.

2. Выбор ИМ объекта. Под ИМ понимается любой вычислительный алгоритм, позволяющий получить описание реакции объекта на входной сигнал. Так, в пакете MatLab Simulink модель может быть построена с помощью разнообразных динамических и статических звеньев, в том числе нелинейных.

3. Выбор функции относительной пригодности. Здесь можно использовать приведенные выше расчетные формулы (3.1) – (3.5), однако для сложных объектов они могут давать неудовлетворительный результат, поскольку задача сравнения выходов сложных объектов эквивалентна задаче распознавания образов. Количественное сравнение должно дополняться качественным анализом – числом точек перегиба, расстоянием между ними и т. п. При этом может быть использовано нечеткое описание траектории движения объекта [39, 40].

3.1.2. Синтез ПИД-регуляторов

Пропорционально-интегрально-дифференциальные регуляторы (ПИД-регуляторы) получили самое широкое распространение при управлении производственными и технологическими процессами. Около 90% регуляторов, находящихся в настоящее время в эксплуатации, используют ПИД-алгоритм [41].

Основное уравнение ПИД-регулятора имеет следующий вид:

$$u(t) = k_p \varepsilon(t) + k_i \int_0^t \varepsilon(\tau) d\tau + k_d \frac{d\varepsilon(t)}{dt}, \quad (3.6)$$

где k_p , k_i , k_d – константы, выбираемые в процессе проектирования. С их помощью удастся обеспечить соизмеримость отдельных слагаемых формулы и описать закон регулирования (рис. 3.4).

В гиперпространстве ПИД-регулятор с заданными коэффициентами определяет некоторую гиперплоскость (поверхность отклика), так что каждой тройке входных координат (ошибка управления, ее производная и интеграл) соответствует определенная точка – сиг-

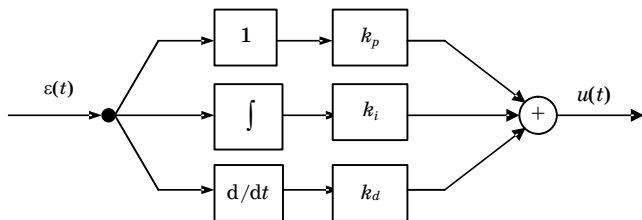


Рис. 3.4. Графическое представление ПИД-регулятора

нал управления. Входные и выходные сигналы для конкретного регулятора всегда ограничены, поэтому реальная поверхность управления является только частью этой гиперплоскости.

Дифференциальная составляющая в формуле (3.6) позволяет повысить быстрдействие регулятора, предсказывая будущее поведение процесса.

Интегральная составляющая в (3.6) призвана ликвидировать статические ошибки управления, поскольку интеграл даже от малой ошибки может быть значительной величиной, вызывающей реакцию регулятора.

На практике часто используются упрощенные версии ПИД-регулятора – ПД- и ПИ-регуляторы, описываемые соответственно формулами

$$u(t) = k_p \varepsilon(t) + k_d \frac{d\varepsilon(t)}{dt}; \quad (3.7)$$

$$u(t) = k_p \varepsilon(t) + k_i \int_0^t \varepsilon(\tau) d\tau. \quad (3.8)$$

В цифровом ПИД-регуляторе вместо производной и интеграла ошибки рассматривают приращение и конечную сумму.

При достаточно малом периоде дискретизации Δt производную и интеграл можно представить в разностном виде с помощью замен:

$$\frac{de(t)}{dt} = \frac{e_n - e_{n-1}}{\Delta t}; \quad \int_0^t e(t) dt \approx \Delta t \sum_{i=1}^N e_{k-i},$$

где n – номер момента времени; N – количество моментов времени; k – текущий момент времени.

Тогда выражение (3.6) можно представить в виде

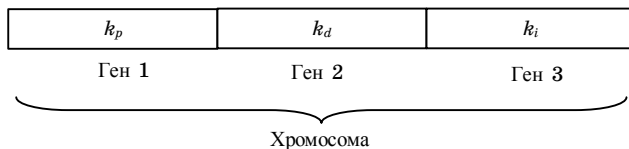


Рис. 3.5. Вид хромосомы при настройке ПИД-регулятора

$$u_k = k_p e_k + k_i \Delta t \sum_{i=1}^N e_{k-i} + k_d \frac{e_k - e_{k-1}}{\Delta t}. \quad (3.9)$$

Рассматривая приращение сигнала управления на интервале Δt , можно преобразовать (3.9) к виду (см., например, [42])

$$u_k = u_{k-1} + d_1 e_k + d_2 e_{k-1} + d_3 e_{k-2}, \quad (3.10)$$

где d_i – коэффициенты, зависящие от Δt .

Таким образом, и для аналогового, и для цифрового ПИД-регулятора хромосома должна кодировать не более трех параметров (рис. 3.5):

Для упрощенных вариантов (ПД- и ПИ-регуляторы) хромосома имеет соответственно всего два гена.

Поскольку настраиваемые коэффициенты ПИД-регулятора могут сильно отличаться по величине друг от друга, может быть полезно перед запуском ГА предварительно получить оценки коэффициентов с помощью известной методики Зиглера–Николса [43].

3.1.3. Синтез нечетких регуляторов

С конца 70-х гг. известны нечеткие логические регуляторы (НЛР), базирующиеся на идеях нечеткой логики Л. Заде [44]. Изначально НЛР были призваны формализовать опыт человека-эксперта, управляющего сложным объектом [45]. Человек не опирается при управлении на точные математические построения, он качественно анализирует протекание процесса, выбирая значение управляющего сигнала на основании сопоставления текущего и желаемого состояния объекта. Аппарат нечеткой логики позволяет описывать качественные понятия и выводить решения на нечетких знаниях. НЛР в настоящее время начинают доминировать во многих областях – от бытовой техники и автомобильной автоматики до космических и оборонных систем. НЛР отличаются робастностью, простотой разработки и реализации, низкой стоимостью.

Подробное описание принципов работы НЛР можно найти, например, в работе [46]. Здесь лишь напомним основные принципы функционирования НЛР, необходимые для понимания принципов его генетической настройки.

Нечетким множеством (НМ) A на универсальном (базовом) множестве X называют совокупность пар элементов вида

$$A = \{\mu_A(x)/x\}, \quad x \in X, \quad \mu_A(x): X \rightarrow [0, 1],$$

где $\mu_A(x)$ – функция принадлежности элемента x НМ A ; знак «/» является разделителем.

Само универсальное множество можно описать так:

$$\mu_A(x) = 1, \quad \forall x \in X.$$

Значение функции принадлежности для конкретного x называют *степенью принадлежности*. Степень принадлежности $\mu_A(x)$ – это субъективная мера того, насколько элемент x соответствует понятию, смысл которого формализуется с помощью НМ A .

На практике желательно иметь аналитическое описание функции принадлежности, чтобы вычислять степень принадлежности для произвольного значения из области определения. Для описания функций принадлежности часто используют гауссову или треугольную формы.

Гауссова функция описывается формулой

$$g_1(x) = \exp\left[-\frac{1}{2b}(x-w)^2\right],$$

где b называется шириной, а w – центром гауссовой функции (рис. 3.6).

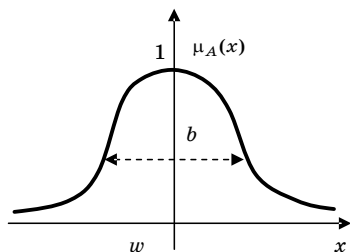


Рис. 3.6. Гауссова функция принадлежности

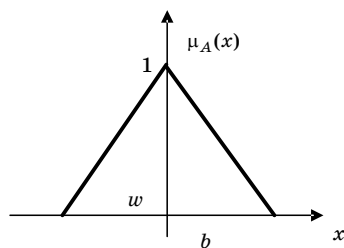


Рис. 3.7. Треугольная функция принадлежности

Треугольная функция принадлежности описывается формулой

$$g_2 = \begin{cases} 1 - \frac{|x-w|}{b}, & \text{если } |x-w| < b; \\ 0, & \text{иначе,} \end{cases}$$

где b и w – половина базовой длины и центр базы треугольной функции соответственно (рис. 3.7).

Таким образом, при этих вариантах описания НМ его функция принадлежности имеет два параметра – центр и ширину.

Возможны и другие функции, например трапецевидная, в которой для описания функции принадлежности нужно использовать четыре параметра.

Нечеткие множества используются при описании лингвистических переменных (ЛП), с помощью которых формулируется качественная информация, представленная в словесной форме [47].

Лингвистической переменной называется переменная, заданная на некоторой базовой шкале и принимающая значения, являющиеся словами естественного языка, которые описываются нечеткими множествами.

Во многих приложениях количество термов ЛП полагается фиксированным и каждому терму заранее присваивается значение. В этом случае ЛП описывается тройкой

$$\{\beta, T(\beta), E\},$$

где β – наименование ЛП; $T(\beta)$ – множество значений (термов) ЛП; E – базовая шкала, на которой определяется ЛП.

Примем: β = «Ошибка управления»; $T(\beta) = \{\text{«малая»}, \text{«средняя»}, \text{«большая»}\}$; $E = [0, 10]$ (рис. 3.8).

С помощью ЛП описываются входы и выходы НЛР. При управлении скалярным объектом, в зависимости от комбинации поступающих входных сигналов, можно рассматривать НЛР П-типа, ПД-типа, ПИ-типа или ПИД-типа. Однако НЛР, в отличие от регу-

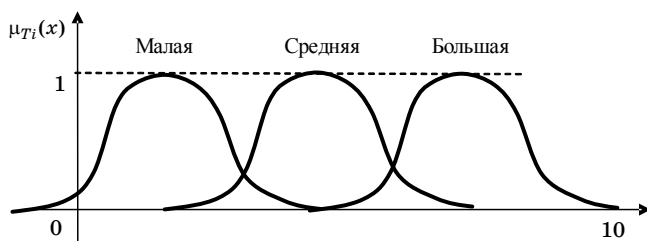


Рис. 3.8. Пример описания ЛП

ляторов, описанных в п. 3.1.2, способен обеспечивать нелинейный закон управления.

Общая структура НЛР представлена на рис. 3.9.

Фаззификация означает операцию вычисления степени принадлежности входного (четкого) значения к различным термам соответствующей ЛП. Дефаззификация предполагает вычисление четкого выходного значения.

Пусть рассматривается НЛР с двумя входными переменными. Тогда правила управления могут быть представлены в одной из следующих двух форм:

If $(X_1=A)$ and $(X_2=B)$ then $Y=C$;

If $(X_1=A)$ and $(X_2=B)$ then $Y=c_0+c_1x_1+c_2x_2$,

где X_1 , X_2 и Y – ЛП; A и B – термы этих ЛП; C – терм выходной переменной; x_1 и x_2 – входные (четкие) значения; c_0 , c_1 и c_2 – константы.

Первое выражение, в котором сигнал управления описывается одним термом, соответствует НЛР типа Мамдани. Второе выражение, в котором сигнал управления описывается линейной комбинацией входных переменных, соответствует НЛР типа Сугэно.

Рассмотрим работу механизма вывода для регулятора Мамдани, в базе правил которого есть всего два правила:

R_1 : If $(X_1=A_1)$ and $(X_2=B_1)$ then $(Y=C_1)$;

R_2 : If $(X_1=A_2)$ and $(X_2=B_2)$ then $(Y=C_2)$.

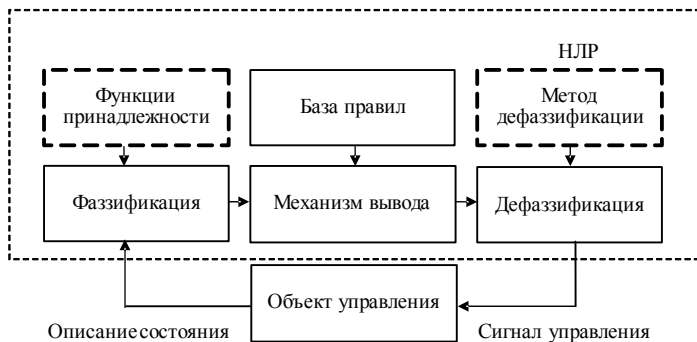


Рис. 3.9. Общая структура НЛР

С помощью аппарата нечетких отношений каждое правило можно описать следующим образом:

$$R_i(x, y, z) = [A_i(x) \wedge B_i(y)] \rightarrow C_i(z).$$

При входных данных $x=x_0$ и $y=y_0$ выход i -го правила получается по формуле

$$C'_i = \underbrace{[A_i(x_0) \wedge B_i(y_0)]}_{\alpha_i} \rightarrow C_i(z),$$

где величина α_i определяет силу запуска правила.

Для всей совокупности правил выходной сигнал описывается формулой

$$C = \bigcup_{i=1}^N C'_i.$$

Логическое умножение и импликация могут быть выполнены с помощью операции взятия минимума. Графическая иллюстрация подобной схемы вывода показана на рис. 3.10.

Для дефаззификации можно использовать, например, дискретный вариант метода центра тяжести, который рассматривает взвешенную сумму сигналов, выдаваемых отдельными правилами:

$$z = \frac{\sum_{i=1}^N \alpha_i z_i}{\sum_{i=1}^N \alpha_i}.$$

Эта же формула может быть использована для регулятора Сугэно.

Существуют разные варианты для описания операций логического умножения и логического следования в правилах НЛР, а так-

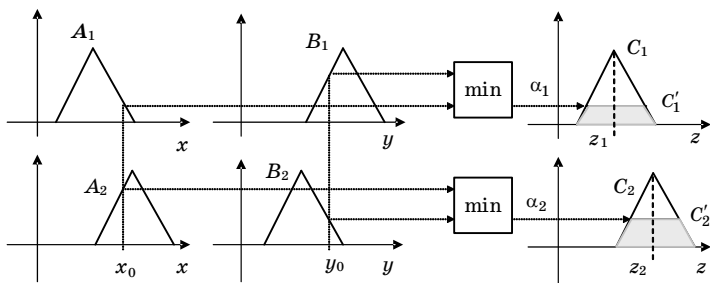


Рис. 3.10. Схема вывода Мамдани для двух правил

же для описания дефаззификации [46]. Однако в гораздо большей степени на результат работы НЛР влияет содержимое базы правил.

Поэтому механизм вывода, а также механизмы фаззификации и дефаззификации можно полагать в НЛР неизменяемыми. Они выбираются на этапе проектирования. Что же касается базы правил, то для ее описания есть следующие основные варианты:

- использование опыта человека-эксперта;
- использование алгоритмов самоорганизации НЛР.

Первый подход не универсален в силу ограниченных психофизических возможностей человека. Второй подход может быть реализован с помощью ГА, для этого надо кодировать параметры НЛР с помощью хромосомы и организовывать эволюционирующую во времени популяцию.

Рассмотрим эти проблемы подробнее. Как показало приведенное выше описание, в НЛР существует две группы параметров:

- параметры функций принадлежности термов ЛП;
- параметры правил, т. е. описание посылок и заключения правила.

Соответственно, при генетическом обучении НЛР хромосома может кодировать либо параметры термов, либо правила, либо и то и другое одновременно (рис. 3.11).

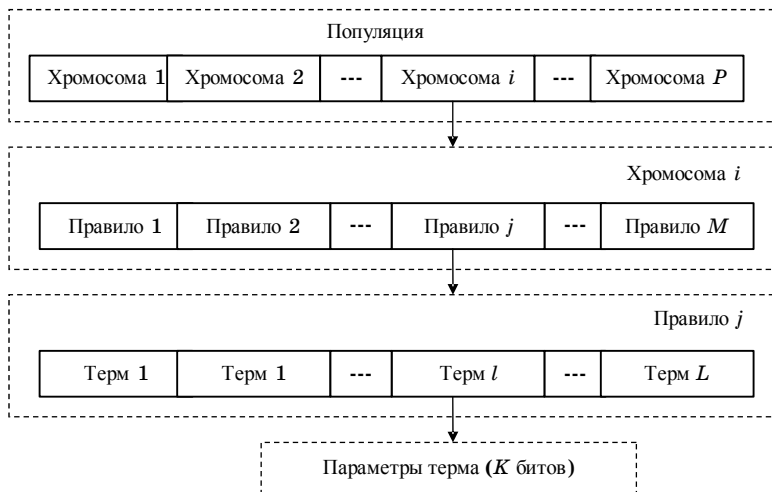


Рис. 3.11. Кодирование параметров НЛР

В генетическом алгоритме традиционно считается, что хромосома имеет постоянную длину. Оценить длину хромосомы при кодировании правил можно следующим образом.

Пусть база правил содержит M правил, каждое правило имеет $L-1$ посылок и одно заключение. Если для кодирования каждого термина взять K битов, то длина хромосомы $N = MLK$. Рассмотрим пример.

Пусть рассматривается 7 правил, каждое правило имеет 2 посылки и одно заключение, каждый терм кодируется тремя битами, тогда $N = 7(2+1)3 = 63$ бита. Получившаяся хромосома показана на рис. 3.12.

В рассмотренном примере терм кодируется тремя битами, что позволяет рассматривать только какой-то один параметр термина. Таким параметром может быть положение термина на базовой шкале (рис. 3.13).

Если рассматривают два параметра термина (центр и ширину), то хромосома будет вдвое длиннее.

Таким образом, даже в рассмотренном простом примере получилась хромосома значительного размера. Это позволяет заметить,



Рис. 3.12. Пример кодирования хромосомы

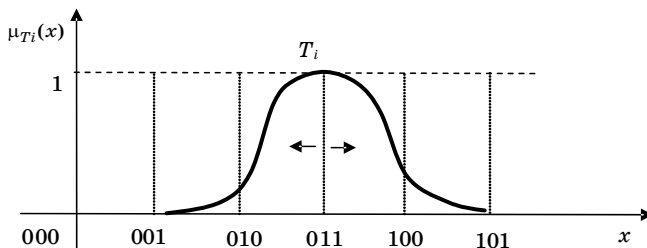


Рис. 3.13. Оптимизация расположения термина на базовой шкале

что при генетическом синтезе НЛР существует проблема размерности.

Одновременное кодирование и термов, и правил вызывает большие сложности, поэтому обычно генетическая настройка НЛР включает несколько этапов ([48 – 50] и др.).

1. Термы ЛП, описывающих посылки и заключения правил, равномерно располагаются по соответствующим базовым шкалам (количество термов – от 3-х до 7). Выбирается фиксированное количество правил.

2. Номера термов, входящих в каждое правило, кодируются двоичными числами, так что каждое правило получает представление в виде двоичного слова. Хромосома представляет собой конкатенацию всех таких слов.

3. Формируется популяция хромосом, начинает работать ГА (см. рис. 3.1). Определяется набор правил, при котором удовлетворяется заданное качество регулирования.

4. Полученные правила фиксируются. Затем делается попытка улучшить поведение НЛР за счет более точного указания параметров термов. Параметры термов (например, ширина и центр) кодируются. Хромосома представляет собой конкатенацию всех таких кодов.

5. Формируется популяция хромосом, начинает работать ГА. Определяется вид термов ЛП, при котором улучшается качество регулирования.

Во многих случаях эффективным оказывается представление НЛР с помощью искусственной нейронной сети с последующим генетическим обучением.

3.1.4. Нейроконтроллеры

Нейроконтроллер – это регулятор, реализованный на базе искусственной нейронной сети (ИНС) ([50 – 52] и др.).

Искусственная нейронная сеть представляет собой набор нейронов (от единиц до миллионов), соединенных каналами обмена информацией.

Искусственный нейрон (ИН) – это простейший аналоговый преобразующий элемент (рис. 3.14).

На вход ИН поступает некоторое множество сигналов. Каждый вход взвешивается – умножается на определенный коэффициент (синаптическую силу). Сумма всех этих произведений определяет уровень активации нейрона. Активационная функция должна быть монотонной и по модулю меньше единицы. Используют мно-

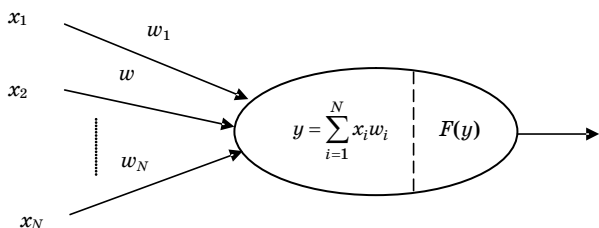


Рис. 3.14. Искусственный нейрон: w_i – множество всех весов; x_i – множество входов; F – активационная функция

жество вариантов активационных функций [53], например гиперболический тангенс:

$$F(y) = \text{th}(ky), \quad k > 0.$$

Существует множество различных топологий ИНС, т. е. способов объединения ИН в сеть. Все эти варианты можно разделить на два класса:

1) статические ИНС, в которых нет обратных связей. Эти сети подобны комбинационным электронным схемам в том смысле, что выходной сигнал зависит только от текущего входа;

2) динамические ИНС с обратными связями. Для таких сетей характерен переходный процесс при изменении входных сигналов.

При конструировании нейрорегуляторов обычно используются статические ИНС – сети прямого распространения, состоящие из нескольких слоев нейронов.

Сети прямого распространения отличаются тем, что ИН каждого слоя не связаны между собой. Нейроны входного слоя (который иногда не изображают и не учитывают при подсчете слоев ИНС) распределяют сигналы между ИН первого скрытого слоя. Выходной сигнал с каждого ИН поступает на входы всех нейронов следующего слоя.

Многослойная ИНС состоит из чередующихся множеств нейронов и весов. При этом каждый слой ИНС может иметь произвольное количество нейронов. На рис. 3.15 показан пример трехслойной ИНС прямого распространения.

При соблюдении ряда условий (к числу которых относится нелинейность активационной функции) многослойная ИНС прямого распространения является универсальным аппроксиматором, т. е. может описать любую функциональную зависимость между входом и выходом [54]. Однако использование ИНС связано с преодо-

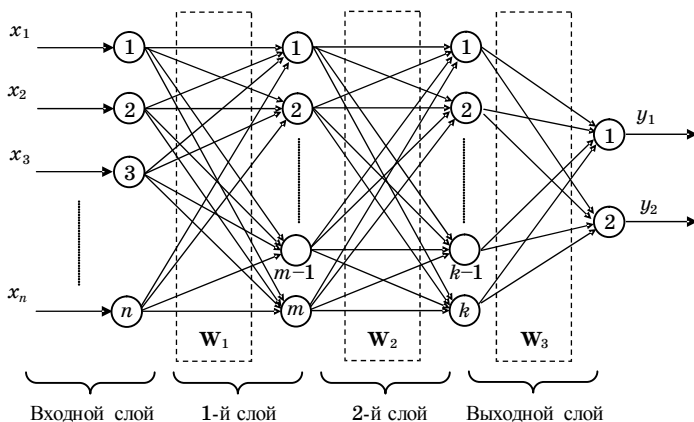


Рис. 3.15. Трехслойная ИНС прямого распространения

лением неопределенности относительно числа ее слоев, числа ИН в каждом слое, а также значений весов межнейронных связей. Эти проблемы решаются с помощью обучения ИНС.

Обычно под обучением ИНС понимается процесс изменения весов межнейронных связей с целью минимизации ошибки. Топология ИНС считается заданной. Нужно отметить, что не существует точных способов выбора количества слоев ИНС и количества ИН в каждом ее слое. Выбор топологии ИНС производят с некоторым «запасом», опираясь на опыт решения практических задач.

Заметим, что ИНС прямого распространения – это не динамическая сеть. В то же время всякий регулятор можно рассматривать как обратную модель объекта, т. е. он должен быть динамическим звеном. Простой путь внесения динамики в поведение статической ИНС заключается в подаче на ее вход задержанных значений ошибки управления. Количество линий задержки зависит от порядка объекта управления (рис. 3.16).

Из существующих методов обучения ИНС наиболее широко используются алгоритм обратного распространения ошибки (АОРО) [55] и ГА.

Алгоритм обратного распространения ошибки показал свои достоинства при решении ряда прикладных задач. Его описание можно найти, например, в работе [46]. АОРО – это алгоритм обучения «с учителем», предполагающий, что заданы обучающие пары, описывающие желаемую реакцию ИНС на входной сигнал. Обучение происходит при минимизации ошибки по всем обучающим парам

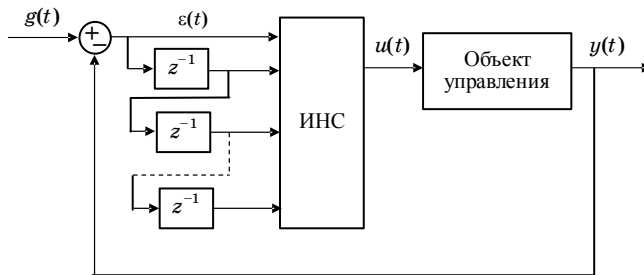


Рис. 3.16. Пример включения ИНС в контур системы управления

путем изменения значения весов ИНС, влияющих на величину ошибки.

Примеры применения АОРО при синтезе систем управления рассмотрены в работе [56].

Однако применение АОРО для проектирования нейроконтроллеров ограничено следующими особенностями:

- **АОРО является локальным алгоритмом, он представляет собой разновидность алгоритма минимизации путем спуска по градиенту целевой функции.** Поэтому АОРО не может работать со сложными функциями ошибки;

- **для применения АОРО надо задать желаемый выход регулятора, но это возможно только в ситуации, когда уже имеется работающий регулятор или человек, управляющий процессом.** Обычно же можно описать лишь желаемый выход объекта.

Вторая из указанных проблем может быть решена за счет использования специальных схем обучения. Например, в схеме обобщенного обучения (или прямого инверсного обучения) ИНС можно рассматривать как инверсную (обратную) модель динамики системы (рис. 3.17).

Таким образом, в этой схеме происходит минимизация ошибки:

$$e_u(t) = u_m(t) - u(t).$$

Основная трудность использования этой схемы заключается в том, что здесь надо находиться в окрестности глобального минимума ошибки, а это требует достаточно точного знания структуры системы. Сигнал $u(t)$ выступает в качестве тестового, он должен быть разнообразным для проявления всех динамических свойств системы. После обучения ИНС используется для непосредственного управления объектом.

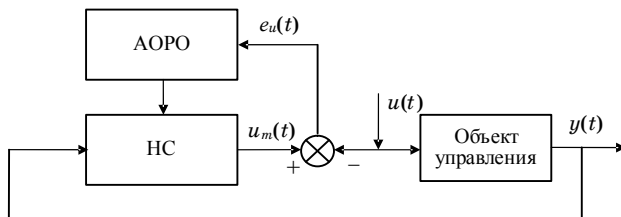


Рис. 3.17. Прямое инверсное обучение НС

Таким образом, если вторую из указанных проблем можно обойти, то первую (локальность АОРО) преодолеть нельзя, что делает актуальным использование ГА для синтеза нейроконтроллеров.

При генетическом синтезе нейроконтроллера хромосома содержит значения весов межнейронных связей, которые могут кодироваться двоичными или действительными числами.

Пример ИНС и хромосомы, кодирующей ее параметры, показан на рис. 3.18.

Использование ИНС для реализации регулятора позволяет (в принципе) описать сложный нелинейный закон управления. Гене-

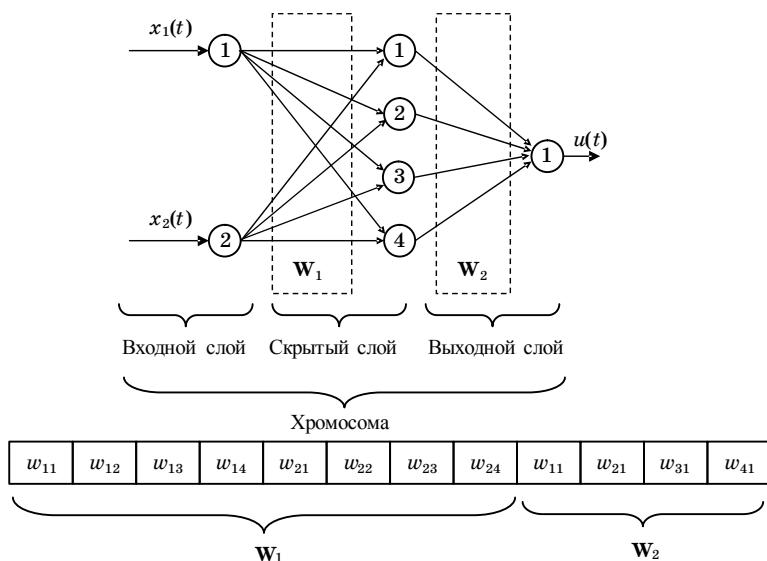


Рис. 3.18. Кодирование параметров ИНС

тическое обучение нейроконтроллера выполнить проще, чем генетическое обучение НЛР [18].

Однако ИНС является «черным ящиком» в том смысле, что ее поведение трудно анализировать. Этим ИНС невыгодно отличается от НЛР, набор правил которого понятен человеку, а непротиворечивость и целостность базы правил поддается анализу [46].

Совместить достоинства ИНС и НЛР можно в рамках нейронечеткой (neurofuzzy) технологии, предполагающей описание НЛР с помощью ИНС.

Для того чтобы описать нечеткие правила, требуется двухслойная ИНС, в которой первый слой отвечает за вычисление степени запуска правил, а второй слой выполняет операцию дефаззификации.

Пусть нечеткая система имеет n правил, а каждое правило содержит m посылок. Тогда первый слой будет иметь $n \times m$ весов:

$$w_{ij}, \quad i = \overline{1, m}, \quad j = \overline{1, n}.$$

Каждый вес должен соответствовать центру терма, описывающего одну из посылок правила.

Активационные функции нейронов первого слоя должны выдавать сигнал, величина которого обратно пропорциональна максимальному отклонению входного сигнала от посылки правила, поэтому в качестве активационной функции здесь можно использовать вариант, представленный на рис. 3.19 (входные сигналы и веса нормализованы).

Таким образом, $F(y_i)$ является степенью запуска i -го правила.

Пример описания простой нечеткой системы, содержащей три правила, каждое из которых имеет две посылки, показан на рис. 3.20, где u_i — это центры термов, описывающих заключения

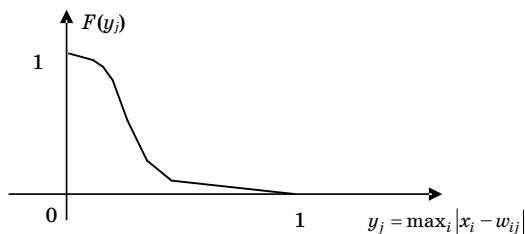


Рис. 3.19. Активационная функция нейрона первого слоя

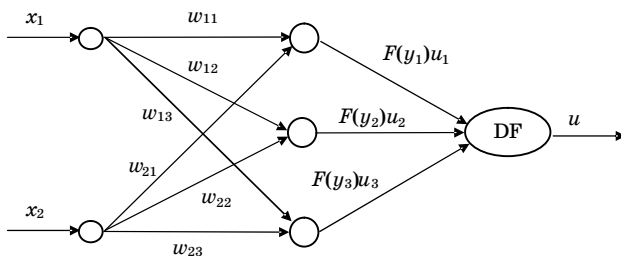


Рис. 3.20. Пример нейронечеткой системы: DF – дефаззификация

правил. Для операции дефаззификации можно использовать дискретный метод центра тяжести.

Генетический алгоритм может быть эффективен не только в задаче синтеза нейроконтроллера, но и при обучении ИНС для решения других задач, таких как идентификация или распознавание образов.

Задача идентификации решается, по существу, в рамках структуры рис. 3.17. Однако там идет речь об обратной модели динамики объекта. Идентификация в классической постановке подразумевает построение модели, аппроксимирующей объект таким образом, что выполняется условие

$$\|Y_M - Y_O\| = \|M(u) - O(u)\| \leq \varepsilon, \quad u \in U,$$

где $\|\cdot\|$ – выбранная норма (функция расстояния); Y_M и Y_O – выходы модели и объекта; M – оператор модели; O – оператор объекта, неявно заданный с помощью множества экспериментальных данных $\{u, Y_O\}$; ε – допустимая величина ошибки; U – допустимое множество управлений.

Идентификационная модель может быть реализована на базе ИНС прямого распространения. Сама процедура идентификации заключается в такой настройке весов ИНС, которая обеспечивает близость выходных сигналов объекта и модели при одинаковых входах (рис. 3.21).

Для придания ИНС прямого распространения динамических свойств здесь можно использовать такой же прием, как и при синтезе нейроконтроллера – на вход ИНС подаются задержанные значения выходного сигнала. Здесь различают два варианта:

1) в цепи обратной связи используется выходной сигнал объекта. К моделям такого типа относятся одношаговые предикторы,

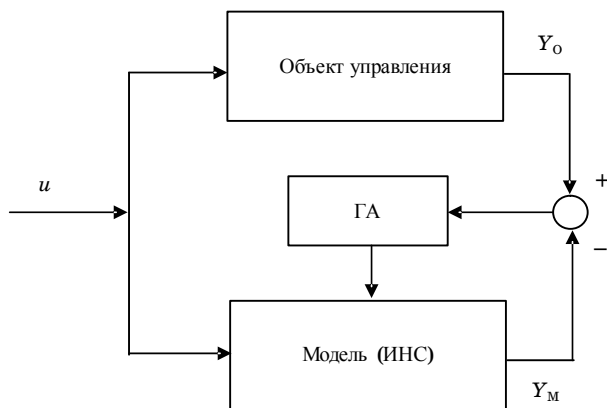


Рис. 3.21. Генетическая настройка идентификационной модели

которые могут предсказать выход объекта на один шаг вперед на основании его предыстории;

2) в цепи обратной связи используется выход модели. Прогноз строится на несколько шагов вперед. Такая модель называется нейрореэмулятором.

Сложность структуры идентификационной модели должна соответствовать сложности объекта, хотя здесь не идет речь о внутреннем подобии – модель должна лишь правильно обобщать функциональные зависимости выхода от входа, заданные набором обучающих пар.

3.2. Мультиагентные системы

3.2.1. Понятие интеллектуального агента

В последние годы в дисциплине искусственного интеллекта появилось новое направление, изучающее функционирование так называемых «систем без организации» (СБО). Такие системы называют также «хаотическими системами» либо «системами с непредвиденным поведением».

Наряду с генетическим алгоритмом, СБО являются еще одной биологической аналогией. В СБО общее поведение системы формируется из сочетания поведений отдельных ее элементов, но не может возникнуть в этих элементах. В качестве примера можно привести роение пчел, поведение колонии муравьев, а также охотничью тактику волчьей стаи. Сложность становится следствием

коллективного поведения множества взаимодействующих простых сущностей – агентов [57].

Например, моделирование некоторых аспектов поведения колонии муравьев выглядит достаточно просто. Здесь каждый муравей живет по следующим трем простым правилам.

1. Перемещайся случайным образом.

2. Если найдена пища, отнеси ее в муравейник и оставь след феромона (фермента), который со временем испарится, а затем перейди к правилу 1.

3. Если след феромона найден, двигайся по нему к пище, а затем перейди к правилу 2.

Если какой-то муравей нашел много пищи, то он приносит ее часть в муравейник, быстро возвращается, снова несет пищу и т. д. После каждого прохода муравья запах феромона на его пути усиливается. По правилу 3 другие муравьи это чувствуют и присоединяются к первому муравью. Так образуются муравьиные тропы.

Компания British Telecom использует модель муравьев и феромонов в своей сети направления звонков, где успешные звонки оставляют своего рода эквивалент феромону, влияющий на последующие звонки.

Еще один распространенный пример того, как простые правила порождают возникающую структуру, — это стая птиц. Каждое движение стаи целесообразно и кажется более плавным, чем движения любой птицы. Однако стая не имеет никакого управления высокого уровня или даже птицы-вожака. Каждая птица следует простому набору правил, которые она использует для взаимодействия с птицами, находящимися вблизи нее.

Поведение птиц можно описать тремя правилами.

1. Если ты оказалась далеко от других птиц, следуй к ближайшей из них.

2. Если ты можешь врезаться в другую птицу, поверни в сторону.

3. Если первых два правила не выполняются, лети в том же направлении, что и ближайшая к тебе птица.

Используя эти три простых правила, ни одна из птиц не представляет себе поведения всей стаи. Таким образом, управление в стае птиц децентрализовано. Так же ведут себя пчелы или машины в автомобильной пробке.

В СБО основное внимание уделяется поведенческим, а не параметрическим аспектам объектов, т. е. описанию для получения системного решения подлежат только различные (хотя и относительно немногочисленные) типы сущностей, но не поведение всей

системы и отдельных сущностей. Это представляет собой подход «снизу вверх», начинающийся с объектов и позволяющий системе свободно развиваться.

Нейронные сети и нечеткие системы могут рассматриваться как подмножества сущностей, называемых агентами. В нечетких системах агентом можно считать одно продукционное правило, участвующее в выработке общего сигнала управления. В нейронных сетях простейший агент – это один нейрон. В генетических (эволюционных) системах агент – это хромосома.

Двумя базовыми понятиями агентно-ориентированного подхода служат агент и среда. В первом приближении агент и среда могут быть описаны так:

- агент – автономная система, способная принимать решения, исходя из своих внутренних представлений о среде;
- агент существует в среде;
- агент взаимодействует с другими агентами и со средой;
- агент может изменять среду.

3.2.2. Архитектура и обучение интеллектуального агента

Агент рассматривается также как некоторый объект, который можно отделить от остального мира. Он существует во времени и пространстве, может взаимодействовать с другими агентами и средами, выполняя некоторые действия (рис. 3.22).

Таким образом, агент – это программная или программно-аппаратная система, которая решает комплекс задач в изменяющейся среде.

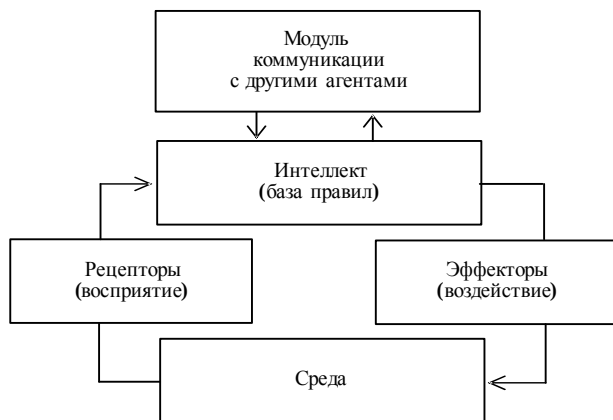


Рис. 3.22. Упрощенное описание интеллектуального агента

Мультиагентная система – это группа агентов, работающих асинхронно. Каждый агент в такой системе может иметь свое специальное назначение в структуре группы.

К агентам могут предъявляться следующие требования [57]:

- интеллектуальность – агент должен быть способен к обдумыванию своих действий, чтобы действовать целеустремленно;
- реактивность – агент должен правильно и вовремя реагировать на неожиданные события и изменения в окружающем мире;
- эффективность – агент должен решать свои задачи эффективно;
- коллаборативность (интерактивность) – агент должен уметь взаимодействовать с другими агентами;
- адаптивность (приспособляемость) – агент должен уметь адаптироваться в изменчивой среде.

Понятие агента используется в разных областях, например на производстве агентом может называться робот, а в области телекоммуникаций – программа и т. п.

Классификация агентов.

1. Реактивные агенты. Эти агенты действуют в режиме реального времени, обычно основываясь на очень небольшом количестве информации и простых правилах «ситуация-действие». Отсутствует символическое представление мира, вместо этого реактивные агенты принимают решения, основываясь непосредственно на информации от своих датчиков.

2. Мыслящие агенты сохраняют внутреннее представление о мире как ментальное состояние, которое может меняться в процессе размышлений агента.

3. Смешанные типы архитектур.

Рассмотрим описание реактивного агента на основе системы классификаторов, предложенное в работе [58].

Система классификаторов состоит из четырех главных компонент (рис. 3.23):

- входной интерфейс (детекторы), передающий состояние среды;
- выходной интерфейс (эффекторы), обеспечивающий взаимодействие или изменение среды;
- база правил – популяция классификаторов, которую можно назвать долговременной памятью;
- база данных – оперативная память, содержащая список поступивших сообщений.

Входной интерфейс обеспечивает систему классификаторов информацией о среде. Например, если система классификаторов ис-



Рис. 3.23. Архитектура интеллектуального агента на основе классификаторов

пользуется для контроля робота, входной интерфейс может являться множеством датчиков. Они могут передавать информацию об окружающих робота объектах. Информация от датчиков кодируется и поступает в оперативную память.

Выходной интерфейс – устройство, получающее команды и производящее на их основе манипуляции со средой. Например, у робота выходной интерфейс может являться набором устройств (эффекторов), с помощью которых можно выполнить некоторые действия (повернуться влево или вправо, двигаться вперед и т. д.).

Список сообщений в любой момент времени может иметь ноль или больше сообщений. Каждое сообщение представляет собой фиксированную строку символов длиной L . Обычно предполагается двоичный алфавит сообщения.

Список классификаторов состоит из множества классификаторов, которые являются правилами, содержащими набор посылок X_i и заключение Y : $X_1, X_2, \dots, X_n \rightarrow Y$.

Если посылка в правиле одна, то классификатор – одноусловный, иначе – многоусловный.

Алфавит, используемый классификатором, такой же, как алфавит сообщений, но к нему добавляется символ #, означающий произвольное значение, например: $A = \{0, 1, \#\}$.

Условная часть классификатора напоминает, таким образом, схему в ГА, т. е. описывает некоторую область в «пространстве состояний» агента.

Иногда используется специфичность (specificity) классификаторов. Специфичность – количество незакрепленных (т. е. не равных #) символов в условной части классификатора, разделенной на ее длину. Таким образом, классификатор тем более специфичный, чем меньше у него символов «#». Значение специфичности изменяется от 0 до 1 и вычисляется по формуле

$$\text{spec} = \frac{L - W}{L},$$

где L – длина условной части классификатора; W – количество символов «#» в условии.

В случае конкуренции двух правил преимущество имеет более специфичное правило.

Когда условная часть классификатора удовлетворяет входному сообщению, происходит активация классификатора и классификатор помещает одно или больше сообщений в список сообщений.

Для многоусловных классификаторов совпадение происходит только в том случае, когда каждое условие классификатора совпадает с каким-нибудь сообщением из списка сообщений. Примеры показаны в табл. 3.1.

Таблица 3.1. Примеры соответствия сообщений и условий классификатора

Условие	Сообщение	Совпадение
10#0	1000	Да
1# #1	1001	Да
1# # #	0110	Нет
10#0, 0#1#	0011 1010 1111	Да
0# #1, 111#	0011 1010 1111	Да
0#10, #101	0011 1010 1111	Нет
10#0, 11#1, 1#0#	0000 1000 1010 1101	Да

Работа системы классификаторов заключается в циклическом выполнении действий:

- 1) добавить сообщения, полученные с помощью входного интерфейса, в список входных сообщений;
- 2) сравнить все сообщения в списке сообщений с условными частями всех классификаторов и активизировать классификаторы, у которых произошло совпадение;
- 3) создать новые сообщения, выполнив действия активизированных классификаторов;
- 4) подать новые сообщения на выходной интерфейс;
- 5) заменить содержимое списка входных сообщений новыми сообщениями;
- 6) перейти к шагу 1.

Рассмотрим пример работы системы классификаторов.

Пусть $L=4$, лист сообщений может содержать два сообщения, и имеется следующая система из 4-х классификаторов:

классификатор 1: 11##:0001;
классификатор 2: 1#00:1010;
классификатор 3: #000:1110;
классификатор 4: ##01:0110,

где символ «:» отделяет условную часть классификатора и его действие.

Допустим, что через детекторы поступило сообщение 1100, тогда выполняются следующие действия:

- 1) сообщение 1100 записывается в лист сообщений;
- 2) активизируются классификаторы 1 и 2, условия которых совпали с сообщением. Классификатор 1 посылает сообщение 0001. Классификатор 2 посылает сообщение 1010;
- 3) сообщение 1010 не соответствует условным частям классификаторов. Сообщение 0001 активизирует классификатор 4, который посылает сообщение 0110. Это сообщение не соответствует условным частям классификаторов, и работа заканчивается.

При постоянном наборе классификаторов поведение агента никак не меняется, в то время как в реальных условиях требуется совершенствовать поведение агента, оптимизируя набор классификаторов и подстраиваясь под окружающую среду. Для этого каждый классификатор имеет силу (strength), показывающую его текущую полезность в системе. Сила изменяется в процессе накопления опыта.

Рассмотрим обучение классификаторов.

Алгоритм пожарников (bucket-brigade) был предложен в работе [58]. Он основан на упрощенной модели экономики. Алгоритм со-

стоит из двух основных компонентов – аукциона (auction) и расчетной палаты (clearing house).

Когда условные части классификаторов совпадают с входным сообщением, они посылают свои сообщения не напрямую, а после аукциона. На аукционе классификаторы делают ставку (bid) пропорционально своей силе. Классификатор, сделавший большую ставку, имеет предпочтение (bid competition) для выигрыша перед другими классификаторами.

После того как выбран классификатор для активации, он должен расплатиться через расчетную палату ставками, участвующими в аукционе. Расплачивается он с теми классификаторами, которые привели его на предыдущем шаге к возможности участия в аукционе. На каждом шаге все классификаторы платят налоги.

Пусть в рассмотренном выше примере все классификаторы стартуют с силой запуска 100 единиц. Положим, что их ставка равна $1/5$ их силы. Тогда на втором шаге активизированные классификаторы 1 и 2 платят по 20 единиц среде. Их сила уменьшается на 20, а сила среды увеличивается на 40. На третьем шаге активизируется классификатор 4, который платит 20 единиц вызвавшему его классификатору 1. Процесс заканчивается.

В результате подобных манипуляций появляются цепочки активизировавшихся классификаторов. Последний активизирующийся классификатор в цепочке отправляет свое сообщение во внешнюю среду и в случае удачного действия получает награду (reward) от среды. Многократное награждение данной цепочки приведет к тому, что силы попавших в нее классификаторов будут постоянно увеличиваться, что в свою очередь увеличит вероятность выбора этих классификаторов в аукционе.

Алгоритм пожарной команды изменяет только силы классификаторов, поэтому качество системы сильно зависит от начальной популяции. Для устранения этого недостатка используется ГА.

Операция отбора выполняется на основе силы классификаторов, остальные генетические операторы выполняются обычным образом.

Генетический алгоритм должен применяться редко, например один раз в тысячу итераций или при стабилизации сил классификаторов в популяции. Это дает алгоритму пожарной команды время подстроить силы классификаторов, так как их правильные значения очень важны при выполнении отбора.

Подобная система обычно используется при реализации адаптивного поведения (роботы-автоматы в динамической среде).

3.3. Генетическое программирование

3.3.1. Автоматическое составление программ

Генетическое программирование (ГП) представляет собой одно из направлений развития ГА [59, 60]. ГП предполагает описание механизмов автоматического составления программ на основе искусственной эволюции.

В настоящее время программирование является одной из наиболее интеллектуальных сфер деятельности человека, полностью автоматизировать процесс написания программ вряд ли когда-либо удастся. Сложности здесь связаны с плохой формализацией процесса программирования как рода деятельности. В процессе написания программы происходит обычно постоянное переосмысление алгоритма решения задачи, структуры программы, вычислительных методов и т. п.

Но, с другой стороны, программист часто прибегает к методу проб и ошибок, корректируя алгоритм по итогам пробных запусков программы. Этот процесс может быть автоматизирован на базе эволюционных алгоритмов.

Поэтому уже сейчас можно выделить области, в которых автоматическое составление программ может быть перспективно. Например, описание поведения интеллектуального агента (см. подразд. 3.2) может представлять собой относительно короткую программу в виде системы иерархически организованных правил. Эта система может постоянно видоизменяться на основе анализа взаимодействия агента и среды.

Отличие ГП от классического ГА заключается в том, что длина программы может быть переменной, соответственно и хромосома будет иметь переменную длину. Кроме того, специфика выбранного языка программирования может накладывать отпечаток на характер генетических операций отбора, скрещивания и мутации.

При изложении принципов ГП обычно рассматривается язык искусственного интеллекта ЛИСП [61], хотя объектом эволюции может быть и программа, написанная на любом другом языке программирования. Но в этом случае требуется рассматривать циклы, ветвления, подпрограммы, что требует использования не деревьев, а направленных графов общего вида.

Программа рассматривается как некое дерево, для которого описывается набор терминальных символов (завершающих ветви), а также набор функциональных символов для описания операций в нетерминальных узлах. Терминальными элементами являются

константы, переменные и функции без аргументов. Функции определяются по отношению к аргументам – константам и переменным. Кроме этого, для работы ГП необходимо описать функцию приспособленности, с помощью которой оценивается качество программы, а также критерий остановки эволюции программ.

3.3.2. Механизмы языка ЛИСП

ЛИСП (LISP) называют функциональным языком или языком обработки списков (механизм обработки списков есть также и в Прологе, но там он не доминирует). ЛИСП разработан еще в 1960 г. Считается, что он особенно популярен в США, в отличие от стран Европы и Японии, где предпочитают Пролог.

Функциональная программа состоит из совокупности определенных функций. Функции могут вызывать другие функции или самих себя. Циклов в программе нет, и повторные вычисления происходят через рекурсию. Рассмотрим базовые принципы языка ЛИСП [61].

Математической основой языка ЛИСП является λ -исчисление, использование которого позволяет конструировать новые функции.

Функция одного аргумента x образуется написанием λ перед этим аргументом, после чего ставится точка и пишется значение функции. Выражение, которое следует за точкой, называется телом λ -абстракции, например:

$$f = \lambda x. 2 \times x + 3, f(5) = 13;$$

$$g = \lambda x. \text{if } x > 5 \text{ then } f(x) \text{ else } -2x, g(3) = -6.$$

При описании функций часто используется так называемая префиксная форма, когда вначале записывается оператор, а потом – операнды:

$$f = \lambda x. +(\times 2 x)(3).$$

Все операции с данными выполняются при помощи функций, и все программы языка ЛИСП носят название функций. Всякую функцию можно описать списком в силу того, что любое символьное выражение является списком.

Напомним, что списком называется любая ограниченная последовательность пунктов ($x_1, x_2, x_3, \dots, x_N$), где $N \geq 0$ (при $N = 0$ список называется пустым). Любая позиция списка может являться подсписком, например: $(A, (B, C), D, E)$ – здесь (B, C) – подсписок. Первая позиция списка называется его головой, остальная часть – хвостом.

В виде списков представляются и программы, и данные. Это позволяет одновременно модифицировать и то, и другое.

Любому списку можно сопоставить древовидную структуру – из каждой вершины отходят две ветви; с одной связана голова списка, а с другой – хвост. Например, пусть необходимо найти значение выражения $x/2 + 8 \times 5$. Переписываем это выражение в префиксной форме $(+ (/ x 2) (\times 8 5))$.

Этой формуле соответствует граф (рис. 3.24), где вершины обозначены кружками

Таким образом, здесь терминальные элементы $T = \{x, 2, 8, 5\}$, функциональные $F = \{+, \times, /\}$.

Выполнение ЛИСП-программы – это процесс упрощения исходного выражения. Процесс упрощения называют *редуцированием*. Упрощаемая часть выражения называется *редексом*. Операция упрощения называется редукцией. Процесс продолжается, пока выражение содержит редексы. Выражение без редекса называется нормальной формой.

Рассмотрим процесс редукции графа для приведенного примера (рис. 3.25).

Полученное число 13 является нормальной формой.

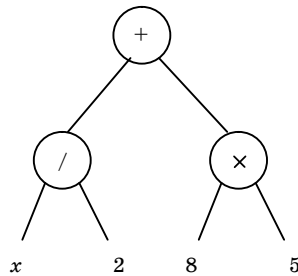


Рис. 3.24. Представление формулы с помощью графа

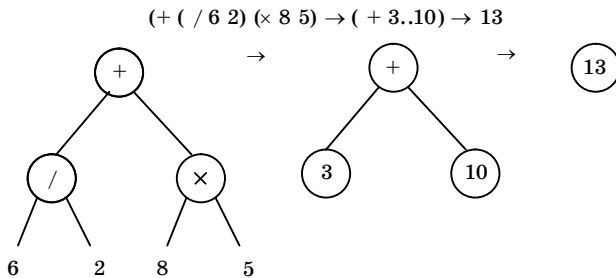


Рис. 3.25. Процесс редуцирования графа

Существуют две теоремы (теоремы Черча – Россера) относительно редукции.

1. Никакое выражение нельзя преобразовать в две нормальные формы, если они разные.

2. Если существует нормальная форма, то существует и способ редукции, из которого эта нормальная форма получается.

При редукции вводятся понятия свободной и связанной переменных.

Переменная, которая является параметром функции (стоит после λ), называется связанной переменной, а переменная, входящая в функцию, но не являющаяся параметром, называется свободной переменной, например:

$$f = \lambda x. + (\times 2 x)(y).$$

В этом выражении x – связанная переменная, а y – свободная.

Существует несколько достаточно очевидных способов редукции.

1. α -преобразование или изменение формального параметра λ -абстракции. Это правило гласит, что можно заменять имя связанной переменной в функции, поскольку если две функции эквивалентны, но имеют разные связанные переменные, их значения при одинаковых аргументах равны. Новое имя переменной не должно совпадать с именем свободной переменной исходной функции.

2. β -преобразование или процесс подстановки в функцию конкретного значения.

3. η -преобразование, смысл которого состоит в упрощении записи λ -абстракции, например:

$$f = \lambda x. +(2) (x) \leftrightarrow (+ 2).$$

Особенно важным является тот факт, что выражение, содержащее несколько редексов, допускает параллельное редукционирование. Иначе говоря, древовидная структура позволяет выполнять вычисления одновременно в разных ветвях. В этом заключается важное преимущество функционального языка программирования по отношению к алгоритмическому, поскольку здесь автоматически происходит выявление компонентов программы, которые допускают параллельную обработку.

3.3.3. ЛИСП и генетическое программирование

Автоматическая генерация программ на языке ЛИСП с помощью ГА была рассмотрена в работе [59]. Популяция формируется из множества альтернативных вариантов программы, каждый из

которых оценивается с точки зрения эффективности решения поставленной задачи.

Для того чтобы автоматически генерировать ЛИСП-программы с помощью ГА, нужно ввести понятие строительного блока – т. е. минимальной неизменяемой части информации. В качестве строительного блока ЛИСП-программы может рассматриваться ветвь дерева.

Особенности использования ГА заключаются в следующем.

1. Первоначальная популяция программ генерируется случайным образом. Количество уровней дерева может быть ограничено (рис. 3.26, где X и Y – переменные логического типа).

2. Каждая программа получает оценку относительной пригодности, отражающую специфику решаемой задачи. Например, пусть программа должна описывать функцию «исключающее ИЛИ» – XOR . Тогда пригодность j -го варианта программы можно описать следующим образом:

$$P_j = \frac{1}{1 + \sum_{i=1}^4 |F_j(x_i, y_i) - XOR(x_i, y_i)|},$$

где $F_j(x_i, y_i)$ – значение функции в соответствии с j -м вариантом программы для i -го набора входных данных.

3. При операции отбора (копирования) рассматриваются программы целиком, т. е. эффективные программы размножаются, а неэффективные – исчезают. Отбор выполняется традиционным способом, например методом колеса рулетки.

4. При операции скрещивания случайно выбираются две программы, затем случайно выбираются два узла в этих программах, и происходит обмен поддеревьями, растущими из этих узлов. Рассмотрим пример выполнения скрещивания.

Пусть заданы два выражения на языке ЛИСП, которым соответствуют две древовидные структуры (рис. 3.27):

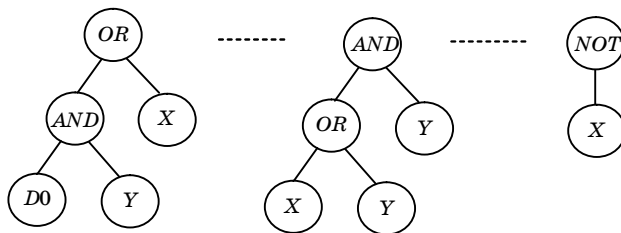


Рис. 3.26. Пример первоначальной генерации программ

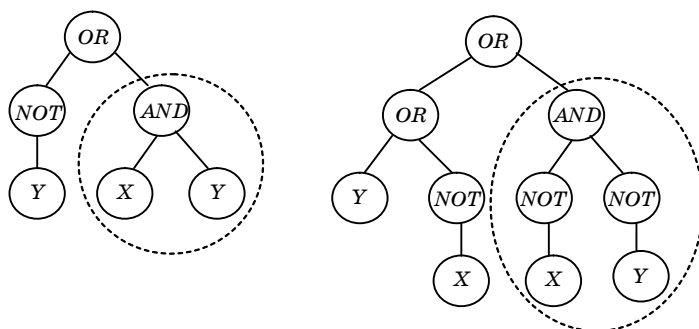


Рис. 3.27. Программы до скрещивания

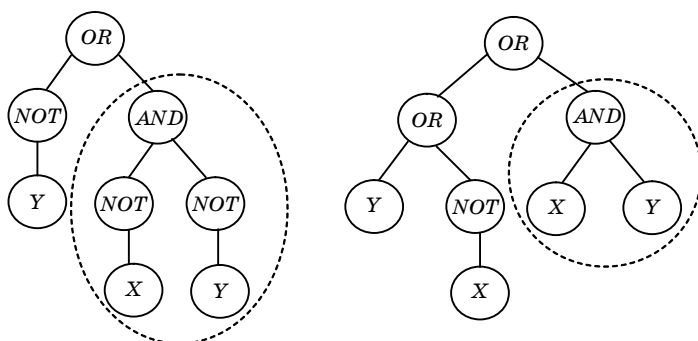


Рис. 3.28. Программы после скрещивания

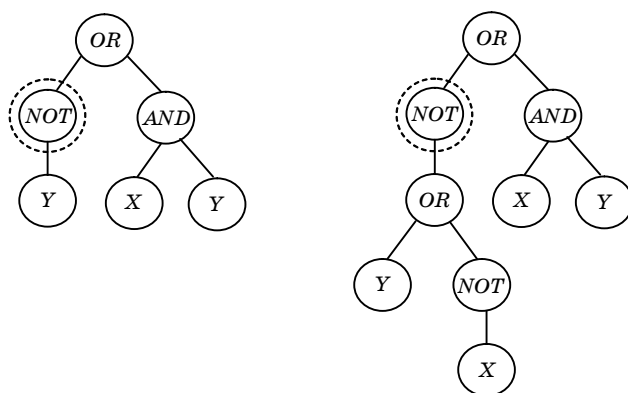


Рис. 3.29. Выполнение мутации в генетическом программировании

1) $OR (NOT(Y)) (AND (X) (Y))$;

2) $OR (OR(Y) (NOT(X))) (AND (NOT(X)) (NOT(Y)))$.

Программы после скрещивания показаны на рис. 3.28, где пунктиром выделены поддеревья.

5. При операции мутации поддерево заменяется случайно порожденным символьным выражением, в котором участвуют допустимые операции над допустимыми данными (рис. 3.29, где узел мутации выделен пунктиром).

3.4. Генетическое тестирование программного обеспечения

3.4.1. Проблемы тестирования программ

Программирование является одним из наиболее массовых и наиболее сложных видов интеллектуальной деятельности человека. Одним из необходимых способов достижения высокого качества программного обеспечения (ПО) является тестирование. По некоторым оценкам, тестирование занимает около 50% общего времени разработки и более 50% стоимости конечного продукта. Основная цель тестирования заключается в оценке надежности ПО. Таким образом, сложность и значимость проблемы тестирования сопоставима с собственно программированием. Автоматизация процесса тестирования является насущной проблемой в технологии разработки ПО.

Между тем хотя надежность является важнейшим требованием, предъявляемым к качеству ПО, само это понятие является для ПО неоднозначным.

При исследовании надежности ПО многие методы заимствуют идеи теории надежности технических устройств, и надежность ПО часто понимают в узком смысле как вероятность того, что ни одна программная ошибка не проявится в течение времени t . Наиболее известными моделями надежности ПО, опирающимися на теорию надежности аппаратуры, являются модели Джелинского – Моранды, Мусы, Шумана и др. [62 – 64]. Однако с точки зрения надежности между техническими устройствами и ПО имеется существенное различие:

- технические устройства подвержены эксплуатационному износу, поэтому их надежность со временем уменьшается;
- ПО не подвержено старению, и его надежность может оставаться постоянной или со временем увеличиваться за счет устранения выявляемых ошибок.

Поэтому вместо временного параметра функции надежности более естественным представляется использование понятия незави-

симых прогонов (запусков) программы. В этих условиях понятие надежности ПО можно интерпретировать как вероятность безотказного выполнения одного прогона программы. Но, с другой стороны, миллион прогонов программы с одними и теми же исходными данными несет не больше информации о надежности ПО, чем один прогон. Кроме того, необходима классификация собственно ошибок ПО.

Нужно выделять критические ошибки, возникновение которых сразу делает ПО ненадежным, а также некритические ошибки, лишь ухудшающие характеристику надежности. Не во всех случаях ошибка допускает количественное измерение, более общим является понимание ошибки как события, имеющего и качественные, и количественные характеристики.

Таким образом, задача разработки автоматических тестирующих программ отличается высокой сложностью, и при ее решении целесообразно использовать методы искусственного интеллекта.

Тестирование ПО является процессом поиска ошибок, поэтому, как будет показано ниже, здесь уместным (и даже неизбежным) является использование ГА – мощного механизма глобальной оптимизации.

3.4.2. Принципы тестирования программ

Всякая программа помимо статической структуры (т. е. кода, описывающего исходный алгоритм) характеризуется динамической структурой, возникающей при определенных входных данных. Соответственно, все виды тестов ПО можно разделить на статические и динамические. При статическом тестировании ПО не исполняется, а происходит анализ кода, структур данных. Динамическое тестирование, напротив, требует выполнения прогонов тестируемого ПО. Для оценки надежности требуется динамическое тестирование.

Один из способов динамического тестирования называется стратегией черного ящика (тестирование с управлением по данным или тестирование с управлением по входу-выходу). При использовании этой стратегии тестовые данные используются только в соответствии со спецификацией программы, без учета какой-либо информации о ее внутренней структуре [65]. При таком подходе обнаружение всех ошибок в программе возможно только при исчерпывающем тестировании, когда на вход программы подаются все возможные наборы входных данных. Для любой достаточно сложной программы количество возможных входных наборов данных очень велико. Поэтому исчерпывающее тестирование программы

невозможно. Однако с помощью эвристик входное пространство программы может быть сужено до обозримых пределов.

Вторая стратегия тестирования называется стратегией белого ящика или тестирования, управляемого логикой программы. Здесь принимается во внимание внутренняя структура программы. Вместо исчерпывающего тестирования входного пространства программы рассматривается исчерпывающее тестирование маршрутов. Подразумевается, что программа полностью проверена, если с помощью тестов удастся проверить выполнение всех ветвей графа, описывающего программу. Однако мощность множества неповторяющихся маршрутов в программе может быть равна мощности множества наборов входных данных программы. Поэтому в работе [62] делается вывод о предпочтительности исчерпывающего входного тестирования по отношению к исчерпывающему тестированию маршрутов, поскольку перебор маршрутов очевидно сложнее перебора вариантов входных данных.

На практике нет необходимости применять ни стратегию черного ящика, ни стратегию белого ящика в чистом виде, поскольку входные данные все равно влияют на выбор и успешность прохождения того или иного маршрута. Рассмотрим пример.

Пусть имеется фрагмент программы:

```
if X≤Y
then
  if X=Y then
    write('ответ 1')
  else
    write('ответ 2')
else write('ответ 3')
```

Используя графовую модель, в которой каждый узел описывает состояние программы, этот фрагмент можно изобразить в следующем виде (рис. 3.30).

Трем возможным исходам работы программы соответствует попадание входных данных в одну из трех областей входного пространства (пример для $X, Y \in [0, 10]$ показан на рис. 3.31)

Таким образом, выбор точек во входном пространстве программы позволяет проанализировать ее динамическую структуру. Для этого программа может снабжаться отладочными операторами, с помощью которых можно построить в оперативной памяти динамический «портрет» программы и узнать, какие ветви программы (маршруты) выполняются для определенного набора входных данных.

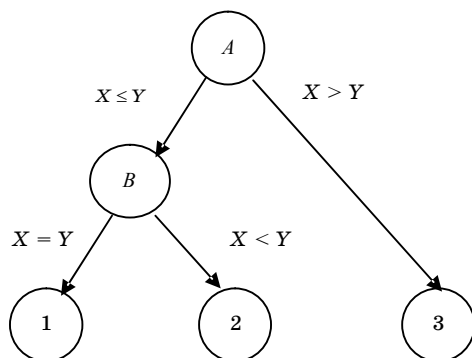


Рис. 3.30. Граф, соответствующий фрагменту программы

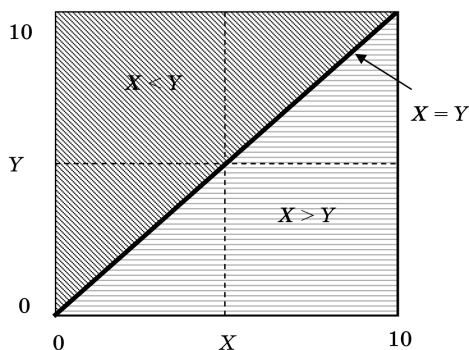


Рис. 3.31. Разбиение входного пространства задачи

Однако, как показывает рис. 3.31, области входного пространства могут сильно отличаться друг от друга по площади. Соответственно, сильно отличается друг от друга и вероятность прохождения того или иного пути в динамической структуре программы. Можно считать, что программа тем надежнее, чем меньше вероятность выбора маршрута, приводящего к ошибочному результату [66].

Таким образом, поведению всякой достаточно сложной программы свойственен дуализм:

– с одной стороны, программа, написанная на процедурном языке программирования, реализует конкретный алгоритм. Поэтому

ей можно поставить в соответствие конечный автомат, поведение которого строго детерминировано. Такой подход получил развитие в работах [67, 68];

– с другой стороны, число состояний этого автомата бывает настолько велико, что его анализ невозможен, и поведение программы приобретает некоторую стохастичность.

3.4.3. Количественная оценка надежности программы

Допустим, что все возможные результаты запуска ПО можно разбить на два класса – правильные и ошибочные. Если считать, что любой результат всегда можно отнести к одному из этих классов, то в качестве меры надежности программы можно принять величину

$$p = 1 - q = 1 - \frac{k}{n}, \quad (3.11)$$

где p – вероятность успешной работы; q – вероятность ошибочного прогона; k – количество прогонов, завершившихся ошибкой; n – общее количество прогонов программы.

Понятие ошибочного завершения программы не всегда может быть точно определено, поэтому более корректной будет запись (3.11) в виде

$$P(n) = 1 - Q(n) = 1 - \frac{\sum_{i=1}^n \theta_i}{n}; \quad (3.12)$$

$$\theta_i = \begin{cases} 0, & |F'(X_i) - F(X_i)| \leq \Delta_i \\ 1, & |F'(X_i) - F(X_i)| > \Delta_i \end{cases}, \quad (3.13)$$

где X_i – возможная комбинация входных данных; $F'(X_i)$ – результат выполнения программы; $F(X_i)$ – эталонное выходное значение; Δ_i – допустимый диапазон ошибки.

Очевидно, что $P \in [0, 1]$, и может показаться, что при достаточно больших n эта величина будет стремиться к вероятности безошибочной работы программы. Однако это не совсем так.

При использовании (3.11) и (3.12) следует учитывать, что любое количество запусков ПО при одних и тех же данных будет давать одинаковые результаты.

Поэтому при использовании (3.12) для экспериментальной оценки надежности ПО должны выполняться следующие требования:

1) входные комбинации X_i должны быть разнообразными – случайными, равномерно распределенными в пространстве входов или распределенными в соответствии с вероятностью их появления на входе программы;

2) при вынесении решения об оценке надежности программы следует учитывать соотношение между количеством входных комбинаций, использованных при тестировании программы, и общим объемом пространства входов программы.

С учетом этих соображений вероятность безотказной работы ПО может быть оценена в виде

$$P = 1 - \sum_{i=1}^n (p_X(X_i)\theta_i), \quad (3.14)$$

где $p_X(X_i)$ – вероятность подачи на вход ПО комбинации входных данных X_i , такая, что выполняется условие

$$\sum_{i=1}^n p_X(X_i) = 1.$$

При равновероятных входных данных выполняются условия

$$p_X(X_i) = \frac{1}{n} \text{ и } \sum_{i=1}^n \theta_i = k.$$

Таким образом, точная оценка уровня надежности ПО предполагает проверку работоспособности ПО для всех возможных сочетаний исходных данных. Но на практике число таких сочетаний оказывается чрезмерно велико и подать все комбинации на вход ПО невозможно. Кроме того, обычно входы ПО распределяются неравномерно, так что чаще всего используется ограниченное множество вариантов.

3.4.4. Генетическое тестирование программ

При тестировании на надежность необходимо выяснить, какие именно комбинации слов входного алфавита вызывают ошибки и насколько вероятно возникновение этих комбинаций на входе.

Для проведения подобного исследования перспективным представляется использование ГА – эффективного механизма поиска в больших пространствах решений [69].

Генетический алгоритм предполагает обработку большого количества вариантов решения задачи (хромосом), образующих популяцию, поведение которой подчиняется генетическим операторам, анализирующим качество различных решений.

В разд. 1 было введено понятие схемы – шаблона, разделяющего пространство поиска. Если хромосома целиком – это вариант решения задачи, то схема – это целая область в пространстве решений. Схемы с малым количеством закрепленных символов имеют большую вероятность появления на входе ПО, а схемы с большим количеством закрепленных символов – малую вероятность.

Допустим, что все хромосомы получили бинарные оценки вида (3.13). Тогда для оценки надежности ПО может быть предложена следующая формула:

$$P = 1 - \sum_{S=1}^N p_S q_S = 1 - \sum_{S=1}^N 2^{-S} q_S, \quad (3.15)$$

где p_S – вероятность схемы с S закрепленными битами; q_S – вероятность ошибки для данной схемы; N – рассматриваемое число схем.

Для расчета величины q_S нужно рассматривать отношение

$$q_S = \frac{nx}{n},$$

где nx – число «ошибочных» хромосом популяции, у которых имеется данная схема; n – общее число «ошибочных» хромосом популяции.

Формула (3.15) оценивает популяцию хромосом в целом, а для нормальной работы ГА требуется иметь индивидуальные оценки по каждой хромосоме.

Для работы ГА бинарные оценки ОП вида (3.13) непригодны. В качестве варианта описания можно предложить следующий прием. Значение $F(X_i)$ можно рассматривать как центр нечеткого множества «Правильная работа программы», описываемого с помощью гауссовой или треугольной функции принадлежности. Тогда для любого $F'(X_i)$ можно получить оценку ОП как соответствующее значение степени принадлежности $\mu_n(F'(X_i))$ (рис. 3.32).

С учетом такого описания формула (3.15) несколько изменяется:

$$P = 1 - \sum_{S=1}^N 2^{-S} q_S \mu_o(F'(X_i)). \quad (3.16)$$

Таким образом, использование ГА для поиска в пространстве входных параметров позволяет выявить области пространства входных данных (схемы), в которых происходит концентрация ошибок. В зависимости от размера этих областей и характеристики происходящих ошибок может быть вынесено суждение о текущей

надежности программы и выполнены действия по повышению ее надежности.

Принципы генетического тестирования ПО поясняет рис. 3.33.

Обобщая приведенный краткий обзор принципов использования ГА для задачи тестирования программ, можно отметить следующее.

Поскольку входные сигналы программного модуля кодируются, непрерывная область определения сводится к дискретной. Это дает возможность, используя на начальном этапе большой шаг дискретности, просмотреть всю область определения входных параметров. Если в какой-то подобласти обнаруживаются ошибки (или подозрения на ошибки), то она исследуется более детально. Если ошибки не обнаруживаются, то просматривается вся область с уменьшенным шагом.

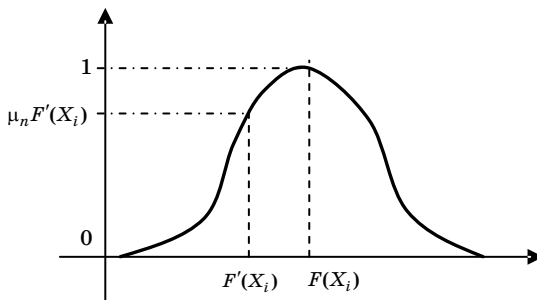


Рис. 3.32. Нечеткое описание ошибки программы

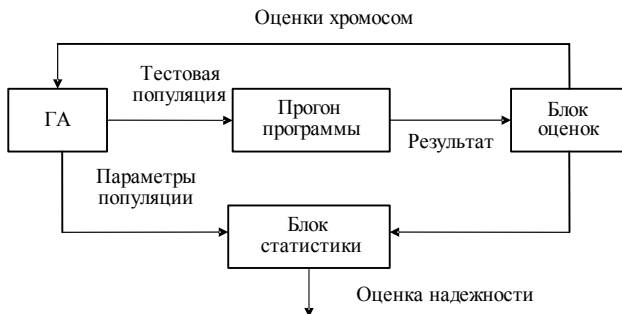


Рис. 3.33. Генетическое тестирование ПО

В каждый момент времени надежность ПО может быть оценена по формуле (3.16), которая учитывает объем входного пространства ПО, вероятность ошибочной комбинации и величину ошибки.

Для работы ГА требуется вычислять функцию ОП, сравнивающую реальный выходной сигнал ПО и его ожидаемое значение. С точки зрения пользователя, неправильная работа программы может характеризоваться целой шкалой оценок – от мелких до критических. Проблема оценки пригодности в общем случае должна решаться с помощью имитационной модели, порождающей эталонный сигнал (а такой моделью уже является техническое задание на программную продукцию).

Анализируя структуру популяции, можно найти критические маршруты в динамической структуре ПО, исправление которых приведет к росту надежности.

Таким образом, предложенная методика представляется весьма перспективной, поскольку позволяет автоматизировать процесс тестирования и сопровождения ПО.

Вопросы для самопроверки

1. Почему экспериментальные методы синтеза регуляторов более универсальны, чем аналитические?
2. Чем отличается эталонная модель от имитационной модели?
3. Что кодирует хромосома при генетическом синтезе регулятора?
4. Каким образом описать соответствие реального выходного сигнала объекта управления желаемому?
5. Как описать относительную пригодность при генетической настройке регулятора?
6. Что представляет собой хромосома при генетической настройке ПИД-регулятора?
7. Что такое нечеткое множество?
8. Какие существуют варианты для аналитического описания функции принадлежности?
9. Что такое лингвистическая переменная?
10. Опишите структуру НЛР.
11. Чем отличается НЛР Мамдани от НЛР Сугэно?
12. Опишите процесс выработки сигнала управления в НЛР.
13. Какими способами можно проектировать базу правил в НЛР?
14. Какие параметры кодируются при генетическом синтезе НЛР?

15. Как оценить длину хромосомы при кодировании параметров НЛР?
16. Что такое нейроконтроллер?
17. Что представляет собой искусственный нейрон?
18. Что такое ИНС?
19. Чем отличается статическая ИНС от динамической?
20. Что представляет собой ИНС прямого распространения?
21. Какие параметры меняются при обучении ИНС?
22. Каким образом можно придать динамические свойства статической ИНС?
23. Как должен отражаться порядок объекта управления в структуре управляющей ИНС?
24. В чем заключаются преимущества ГА по отношению к алгоритму обратного распространения ошибки?
25. Что такое прямое инверсное обучение ИНС?
26. Какие сравнительные достоинства имеют нейроконтроллеры по отношению к НЛР и наоборот?
27. Что такое нейронечеткая система?
28. Опишите структуру ИНС для представления нечетких правил.
29. Как решается задача идентификации при помощи ИНС и ГА?
30. Чем отличается одношаговый предиктор от нейроэмулятора?
31. Что такое система без организации?
32. Как понятие агента можно применить, рассматривая поведение ИНС, нечеткой системы, ГА?
33. Опишите структуру интеллектуального агента.
34. Что такое мультиагентная система?
35. Как классифицируются интеллектуальные агенты?
36. Что такое система классификаторов, как она используется при описании интеллектуального агента?
37. Как происходит активизация классификатора? Что такое специфичность классификатора? Как описывается полезность классификатора?
38. Какую роль играет ГА при описании интеллектуального агента с помощью классификаторов?
39. Что такое ГП?
40. В каких сферах перспективно использование ГП?
41. Чем отличается ГП от классического ГА?

42. Что представляет собой функциональная программа на языке ЛИСП?
43. Как записывается функция в префиксной форме?
44. Что такое список?
45. Что такое редуцирование и редекс?
46. Какие способы применяют при редуцировании?
47. Какой тип графа используется для представления списка?
48. Сформулируйте теоремы Черча – Россера.
49. Что является строительным блоком в ЛИСП-программе в процессе ГП?
50. Как выполняются генетические операции в ГП?
51. Чем отличается понятие надежности ПО от надежности технических устройств?
52. Что такое статическая и динамическая структура программы?
53. Опишите стратегию черного ящика при тестировании ПО.
54. Опишите стратегию белого ящика при тестировании ПО.
55. Возможно ли описание программы как конечного автомата?
56. Как описать вероятность успешной работы программы, используя информацию о результатах ее запусков?
57. Опишите принципы генетического тестирования ПО.

4. ГЕНЕТИЧЕСКИЙ АЛГОРИТМ В MATLAB

4.1. Общие сведения

Genetic Algorithm and Direct Search Toolbox (GADS) предназначен для расширения функциональных возможностей пакета MatLab при решении задач оптимизации [70]. В GADS содержатся новые алгоритмы по сравнению с классическими алгоритмами оптимизации, описанными в Optimization Toolbox. GADS содержит программы для решения задач оптимизации на основе использования двух методов:

- генетического алгоритма (Genetic Algorithm);
- метода прямого поиска (Direct Search).

Генетический алгоритм и метод прямого поиска можно использовать для задач, которые трудно решить обычными методами оптимизации, например, когда искомая целевая функция является разрывной, нелинейной, не имеет производных.

Исходя из поставленной в учебном пособии задачи ниже рассматриваются только принципы использования ГА.

Реализованный в GADS вариант ГА работает согласно следующей общей схеме.

1. Создается произвольное исходное семейство хромосом (индивидуумов), образующее семейство (популяцию).

2. Воспроизводится новая популяция, при этом каждая i -я популяция создается из $(i-1)$ -й популяции с помощью последовательности шагов:

- для каждой хромосомы текущей популяции вычисляется значение пригодности (Fitness function);
- значения пригодности всех хромосом подвергаются ранжированию;
- часть индивидуумов из текущей популяции, которые имеют наилучшие значения пригодности (элита), передаются без изменения в следующую популяцию;
- отбираются хромосомы-родители для создания части новой популяции;
- из хромосом-родителей производятся хромосомы-потомки. Одна часть лучших хромосом проходит в будущую популяцию без изменений (elite children), другая часть получается после применения операции мутации (mutation children), третья часть – после применения кроссовера (crossover children). При кроссовере двух хромосом получается один потомок.

3. Новая популяция заменяет старую популяцию.

4. Проверяются критерии останковки алгоритма; если они не выполнены, то происходит возврат к п. 2.

В GADS функцией пригодности называется минимизируемая функция, но любую задачу можно рассматривать как задачу максимизации, если взять функцию с обратным знаком.

Под индивидуумом в ГА понимается точка, в которой возможен расчет функции пригодности. Таким образом, вектор допустимых значений, чья размерность равна числу переменных данной задачи, и есть индивидуум (хромосома). Ему соответствует определенная пригодность (score).

Хромосома (геном) состоит из генов, таким образом, гены – это компоненты хромосомы.

Популяция есть массив хромосом. Например, если размер популяции равен 100 и число генов (параметров функции пригодности) равно 3, то данное семейство представляет собой матрицу размером 100×3. Значения хромосом (геномов) могут повторяться более одного раза.

Основные функции ГА в пакете GADS:

- ga – запуск ГА из командной строки;
- gaoptimget – получить параметры ГА;
- gaoptimset – задать параметры ГА;
- gatool – графическое окно ГА.

Все функции GADS являются М-файлами MatLab. Имеется возможность просмотра этих функции путем выполнения обычного оператора:

```
>> type <function_name>
```

Пользователь может видоизменять стандартные функции GADS или вводить собственные функции.

Графическое окно gatool предоставляет пользователю удобные возможности для работы с ГА, позволяя вводить параметры задачи и анализировать ход работы ГА.

4.2. Использование графического окна GATool

4.2.1. Общий вид графического окна

После ввода команды

```
>> gatool
```

на экране появится основное графическое окно GATool (рис. 4.1).

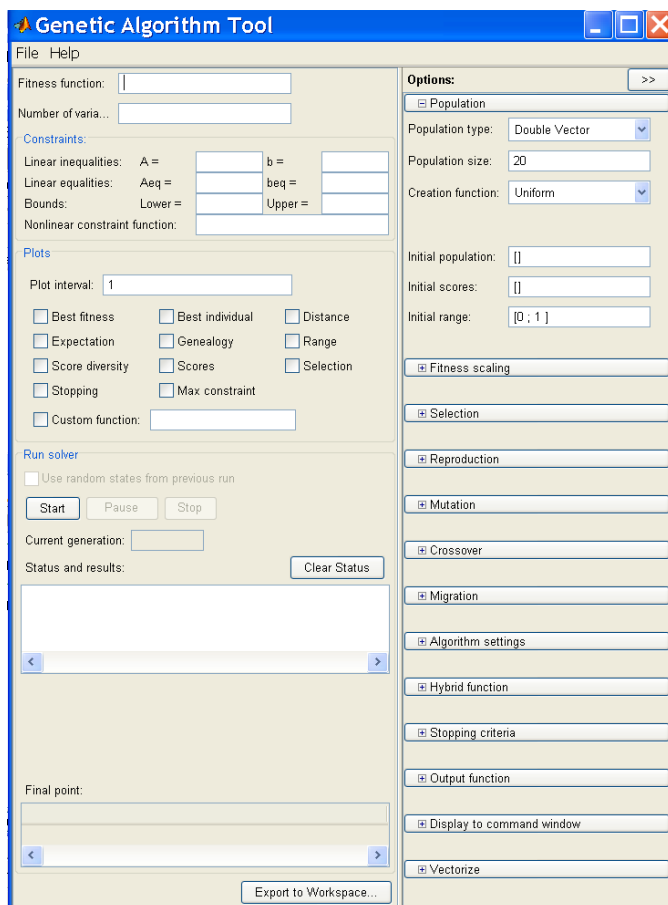


Рис. 4.1. Основное окно GATool

В левой части основного окна gatool имеется два основных поля ввода: Fitness function и Number of variables, а также три панели: Constraints, Plots и Run solver, управляющие, соответственно, заданием ограничений в задаче оптимизации, выводом графической информации и запуском ГА.

В правой части основного окна gatool располагается панель Options, с помощью которой можно изменить выбранные параметры ГА. К опциям ГА относятся опции:

- популяции (population);
- масштабирования пригодности (fitness scaling);

- селекции (selection);
- репродукции (reproduction);
- мутации (mutation);
- скрещивания (crossover);
- миграций (migration);
- гибридных функций (hybrid function);
- критерия останова (stopping criteria);
- вывода функций (output function);
- отображения в командном окне (display to command window);
- векторизации (vectorize).

4.2.2. Описание задачи и ограничений

Для использования возможностей GADS необходимо иметь описание целевой функции в виде М-файла. Функция должна иметь в качестве параметра вектор-строку, чья размерность равна числу независимых переменных задачи. Целевая функция должна возвращать скалярное значение.

Например, предположим, что необходимо найти минимум следующей функции:

$$s = 20 + x^2 + 10\cos(2\pi x).$$

Эта функция имеет множество локальных минимумов, ее график показан на рис. 4.2

Рассматриваемая функция имеет один независимый параметр и возвращает одно значение. М-файл для описания функции имеет вид

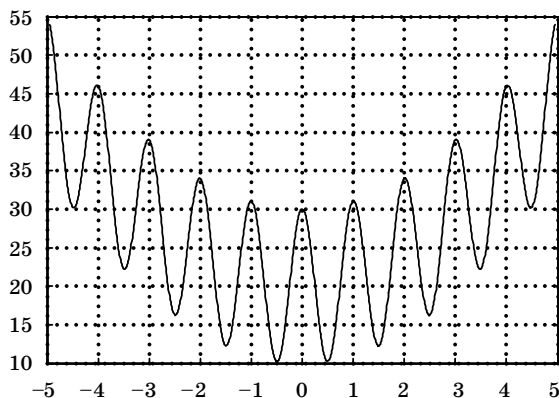


Рис. 4.2. Мультимодальная функция

```
function s = test(x)
    s = 20+x^2+10*(cos(2*pi*x))
end
```

М-файл должен находиться в одном из рабочих каталогов MatLab.

В поле Fitness function необходимо ввести ссылку на минимизируемую функцию: @test.

В поле Number of variables вводится количество независимых переменных минимизируемой функции (рис. 4.3).

Во второй версии GADS появилась возможность описания ограничений (constraints) при поиске минимума функции пригодности (рис. 4.4).

Линейные ограничения в виде равенств задаются с помощью матриц Aeq и beq, линейные ограничения в виде неравенств задаются с помощью матриц A и b, границы переменных задают величины Lower и Upper (рис. 4.4). Окно Nonlinear constraint function служит для указания функции, описывающей нелинейные ограничения.

Таким образом, задача минимизации с ограничениями рассматривается в виде:

$$\begin{aligned} & \min_x (f(x)); \\ & C_i(x) \leq 0, \quad i = \overline{1, m}; \\ & C_i(x) = 0, \quad i = \overline{m+1, N}; \\ & Ax \leq b; A_{eq}x = b_{eq}; Lower \leq x \leq Upper, \end{aligned}$$

где $C(x)$ представляет собой систему ограничений в виде нелинейных равенств и неравенств; m – число нелинейных ограничений в

The image shows a user interface for setting optimization parameters. It has two input fields: 'Fitness function:' with the text '@test' entered, and 'Number of variables:' with the text '1' entered.

Рис. 4.3. Ввод имени функции и числа переменных

The image shows the 'Constraints' section of the GADS interface. It contains several input fields: 'Linear inequalities: A =', 'Linear equalities: Aeq =', 'Bounds: Lower =', and 'Nonlinear constraint function:'. To the right of these fields are corresponding fields for 'b =', 'beq =', and 'Upper ='. The 'Nonlinear constraint function' field is currently empty.

Рис.4.4. Ввод ограничений в задаче оптимизации

виде неравенств; N – общее число нелинейных ограничений. Рассмотрим пример.

Пусть ограничения для функции двух переменных заданы в виде

$$\begin{cases} x_1 + x_2 \leq 2 \\ 2x_2 - x_1 \leq 2 \\ 2x_1 + x_2 \leq 3, \\ x_1 \geq 0 \\ x_2 \geq 0 \end{cases}$$

тогда матрицы имеют вид

» $A = [1 \ 1; -1 \ 2; 2 \ 1];$

» $B = [2; 2; 3];$

» $Lb = [0; 0].$

Пусть заданы нелинейные ограничения в виде неравенств

$$\begin{cases} 1.5 + x_1 x_2 + x_1 - x_2 \leq 0 \\ -x_1 x_2 + 10 \leq 0 \end{cases},$$

эти ограничения можно описать с помощью файл-функции

```
function [c, ceq] = simple_constraint(x)
c = [1.5 + x(1)*x(2) + x(1) - x(2); -x(1)*x(2) + 10];
ceq = [];
```

4.2.3. Визуализация результатов работы алгоритма

Панель Plots дает возможность визуализации параметров работы ГА. Это позволяет анализировать и повышать эффективность работы алгоритма (рис. 4.5).

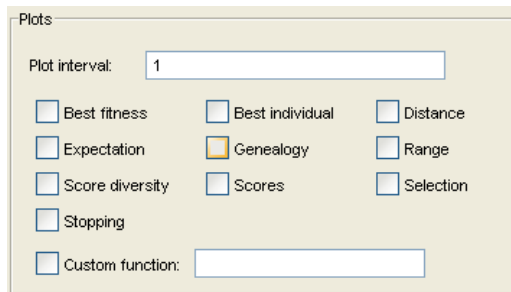


Рис. 4.5. Опции панели Plots

Plot interval – число поколений между обновлениями графиков (по умолчанию – 1).

Best fitness (@gaplotbestf) – графики наилучшего и среднего значения функции пригодности после каждой итерации (рис. 4.6).

Expectation (@gaplotexpectation) (ожидание) – масштабированная пригодность для каждого поколения (ожидаемое число потомков хромосомы в зависимости от значения пригодности). Пример показан на рис. 4.7.

Score diversity (@gaplotscorediversity) – гистограммы значений пригодности для групп хромосом (см. рис. 4.8, где, например, одинаковую пригодность, равную 10, имеют 11 хромосом).

Stopping (@plotstopping) – отображает уровень выполнения критериев останова ГА (рис. 4.9, варианты критериев остановки описаны ниже).

Для примера на рис. 4.6 решение оказалось найдено уже в 23-м поколении. Как правило, значения наилучшей пригодности улучшаются наиболее заметно на ранних поколениях, когда выбранные объекты наиболее далеки от оптимального значения. Для последних поколений, по мере приближения данного семейства к опти-

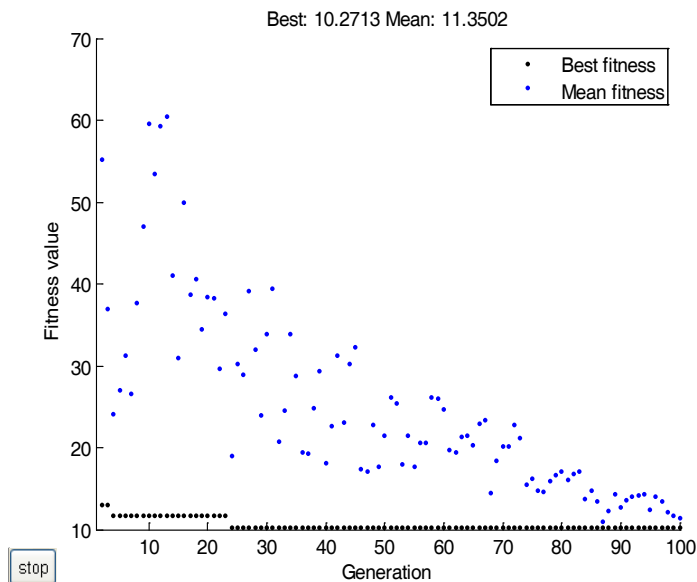


Рис. 4.6. Графики наилучшей и средней пригодности

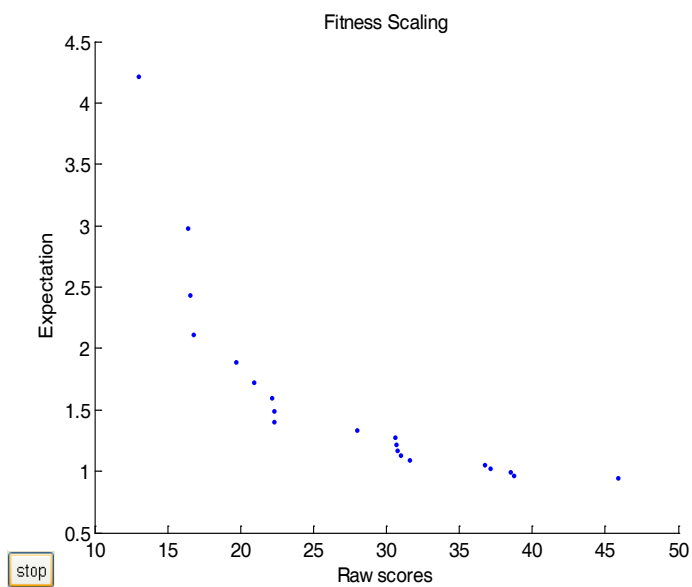


Рис. 4.7. Масштабированная пригодность

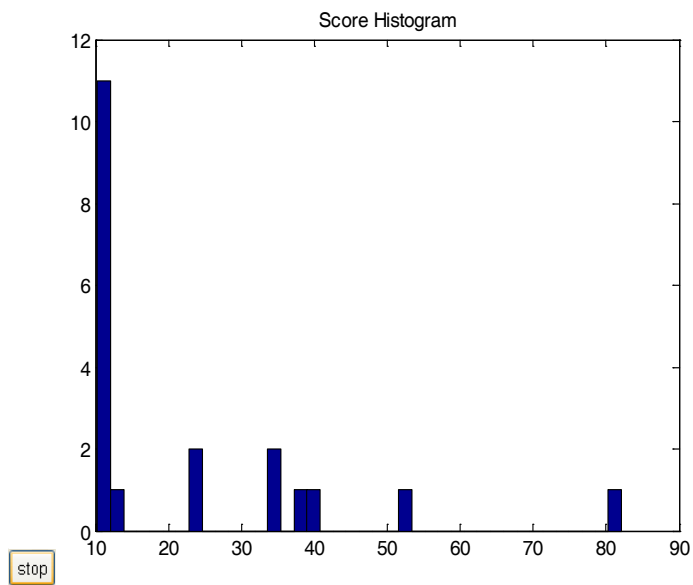


Рис. 4.8. График значений пригодности хромосом

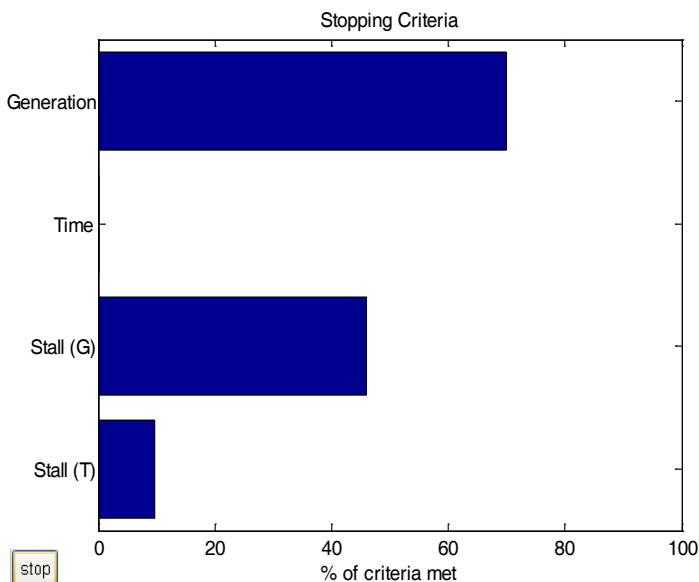


Рис. 4.9. График выполнения критериев остановки

мальной точке, улучшение функции пригодности происходит более медленно.

Best individual (@gaplotbestindiv) – номер и значение лучшей хромосомы в поколении (рис. 4.10). Поскольку в ГА ищется минимум фитнес-функции, лучшим является наименьшее значение.

Genealogy (@gaplotgenealogy) – показывает происхождение каждой хромосомы (рис. 4.11: на экране элитный отбор – черный, скрещивание – синий, мутация – красный).

Scores (@gaplotscores) – значение пригодности для каждой хромосомы популяции (рис. 4.12).

Distance (@gaplotdistance) – среднее расстояние между хромосомами популяции (рис. 4.13). Эта величина описывает разнообразие генетической информации в популяции. При большой дистанции охватывается большее пространство поиска.

Range (@gaplotrange) – наилучшее, наихудшее и среднее значение пригодности хромосом популяции (рис. 4.14).

Selection (@gaplotselection) – гистограмма выбора родительских хромосом (рис. 4.15).

Custom function – функция, вводимая пользователем.

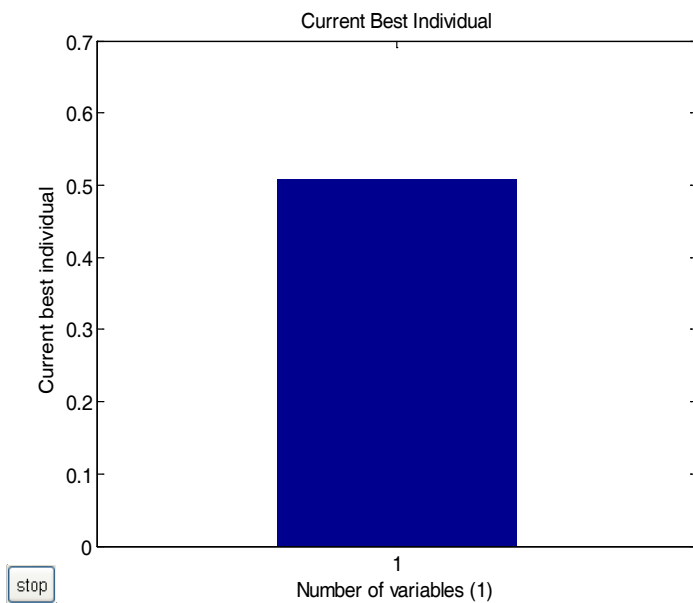


Рис. 4.10. Номер и значение лучшей хромосомы

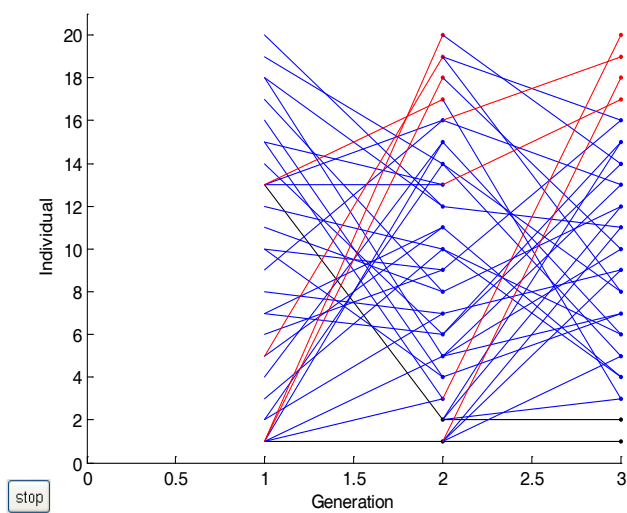


Рис. 4.11. Генеалогия хромосом

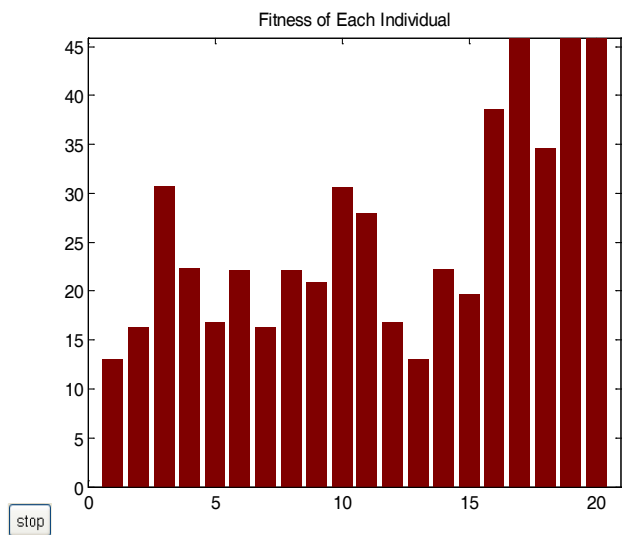


Рис. 4.12. Пригодность хромосом популяции

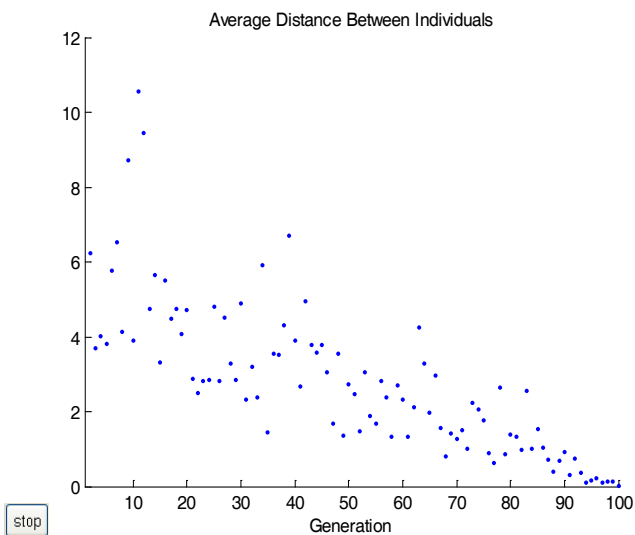


Рис. 4.13. Расстояние между хромосомами популяции

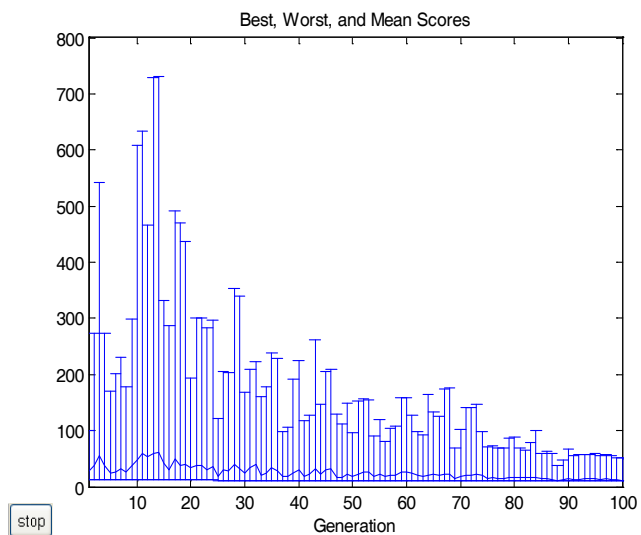


Рис. 4.14. Оценка пригодности хромосом

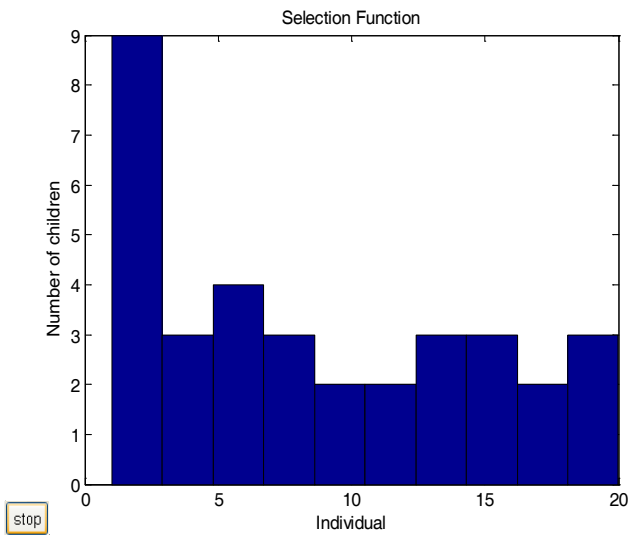


Рис. 4.15. Число потомков хромосом

4.2.4. Параметры запуска генетического алгоритма

Запуск ГА происходит по нажатию кнопки Start на панели Run solver.

По мере того, как происходит выполнение алгоритма, в поле Current generation происходит отображение числа текущих генераций. Можно время от времени приостанавливать алгоритм, нажимая на кнопку Pause. После того, как операция выполнится, имя кнопки заменится на Resume. Для продолжения работы алгоритма с точки останова следует нажать кнопку Resume.

По окончании алгоритма на панели Status and results появится несколько сообщений (рис. 4.16).

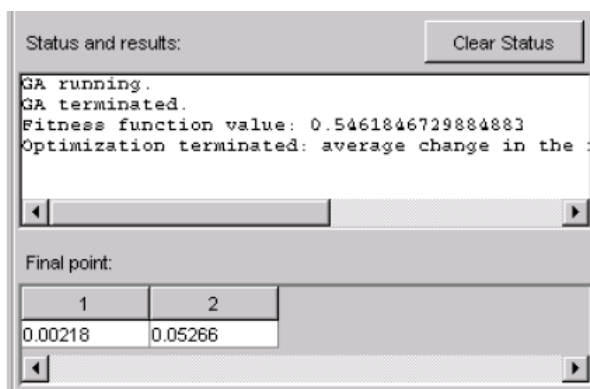


Рис. 4.16. Сообщения по окончании работы ГА

Fitness function value – найденное оптимальное (минимальное) значение функции пригодности.

Final point – аргументы, которым соответствует найденное оптимальное значение.

В окне Status and results появляется также сообщение о причине остановки алгоритма. Например: сообщение «average change in the fitness value less than options.TolFun.» означает, что функция пригодности перестала улучшаться.

4.2.5. Панель Options

Опции популяции показаны на рис. 4.17.

Population type – этот параметр определяет тип хромосом. Здесь возможны следующие варианты:

- Double vector (по умолчанию) – вектор, составленный из действительных чисел формата double (64 бита, формат с плавающей запятой);

Population type:	Double Vector
Population size:	20
Creation function:	Uniform
Initial population:	[]
Initial scores:	[]
Initial range:	[0 ; 1]

Рис. 4.17. *Опции популяции*

- Bit string – строка двоичных символов;
- Custom – тип данных, определяется пользователем. В этом случае требуется разработать собственные функции инициализации, мутации и скрещивания.

Population size – размер популяции (по умолчанию равен 20 хромосомам). При большом размере популяции охватывается большее пространство поиска, однако растут вычислительные затраты для каждой новой генерации. Если в поле Population size задать вектор, то ГА будет работать с субпопуляциями, количество которых соответствует длине вектора, а их размер – компонентам этого вектора.

Creation function – функция инициализации популяции:

- Uniform – создает случайную начальную популяцию с равномерным распределением (используется по умолчанию);
- Custom – использование пользовательской функции инициализации, при выборе этого варианта появляется поле Function name, в котором требуется указать имя используемой функции (@YourFcnNameHere).

Initial population – в этом поле можно задать начальную популяцию для ГА (Creation function использоваться в этом случае не будет). Вводимый в это поле массив должен содержать Population size строк и Number of variables столбцов.

Initial scores – начальное распределение функции пригодности для хромосом.

Initial range – задание диапазона, в котором будут генерироваться случайные значения при создании начальной популяции. Диапазон имеет вид матрицы, в которой две строки и Number of variables столбцов. Каждый столбец имеет вид $[X_{\min}; X_{\max}]$.

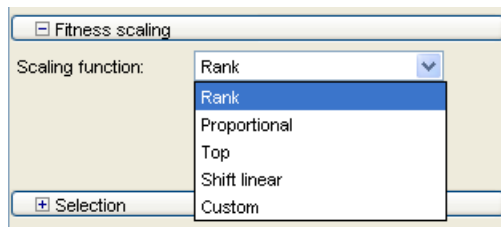


Рис. 4.18. Выбор функции пригодности

Панель Fitness scaling представлена на рис. 4.18.

Опции масштабирования пригодности позволяют преобразовать «сырую» фитнес-функцию к виду, удобному для выполнения селекции. Рассмотрим пример.

Пусть «сырая» фитнес-функция представлена на рис. 2.2. Могут быть выбраны следующие варианты шкалирования функции пригодности:

Rank – функция, используемая по умолчанию. Хромосомы сортируются в соответствии со значениями своей пригодности. Ранг хромосомы зависит от ее места в отсортированной по возрастанию пригодности популяции.

Пример функции ранжирования:

```
nParents=32
[unused,i] = sort(scores);
expectation = zeros(size(scores));
expectation(i) = 1./ ((1:length(scores)).^ 0.5);
expectation = nParents * expectation./ sum(expectation);
```

где nParents – число родительских хромосом, необходимых для создания новой популяции.

Результат использования этой функции по отношению к популяции, представленной на рис. 2.2, показан на рис. 4.19.

Сравнивая с рис. 2.3, можно заметить, что это нелинейное ранжирование.

Proportional – масштабирование пропорционально «сырому» значению пригодности.

Top – здесь параметр Quantity определяет процент лучших хромосом, которые с равной вероятностью будут использованы для формирования промежуточной популяции.

Shift linear – смещенное линейное шкалирование.

Custom – функция шкалирования определяется пользователем.

Опции селекции представлены на рис. 4.20.

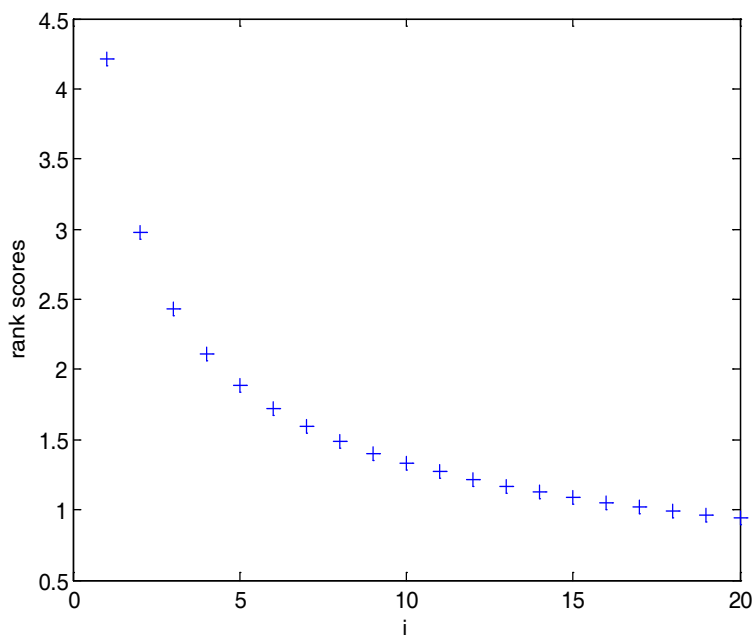


Рис. 4.19. Ранжированная функция пригодности

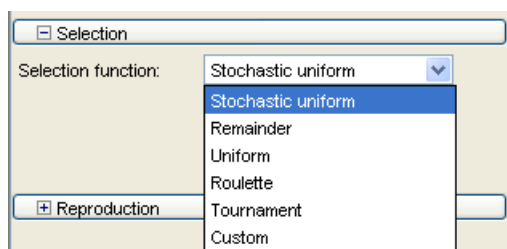


Рис. 4.20. Выбор функции селекции

Функция селекции служит для выбора родительских хромосом. Здесь возможны следующие варианты:

- Stochastic uniform – стохастическая равномерная селекция. Хромосомы отображаются в отрезки прямой, длина которых пропорциональна фитнес-функции. В процессе селекции происходит движение по этой прямой шагами одинаковой длины. Длина шага рассчитывается, исходя из требуемого количества хромосом. Точ-

ка, из которой делается первый шаг, выбирается случайно (в пределах длины шага). После каждого шага отбирается та хромосома, в пределы которой попала точка;

- Remainder – этот метод описан в подразд. 2.4 (стохастический отбор с остатком);

- Uniform – здесь родительские хромосомы отбираются в соответствии со значением пригодности и требуемым числом потомков. Этот метод полезен при отладке и тестировании алгоритмов, но малоэффективен в реальных задачах;

- Roulette – метод колеса рулетки;

- Tournament – турнирная селекция;

- Custom – использование функции селекции, определяемой пользователем. В поле Function name нужно указать ссылку на функцию, определяемую пользователем.

Опции репродукции показаны на рис. 4.21.

Elite count – определяет число лучших хромосом, которые гарантированно попадают в следующую популяцию. По умолчанию значение Elite count=2.

Crossover fraction – определяет часть хромосом, которые попадают в будущую популяцию в результате операции скрещивания. По умолчанию значение этого параметра равно 0,8, что при размере популяции 20 дает 16 хромосом.

Опции мутации (mutation) показаны на рис. 4.22.

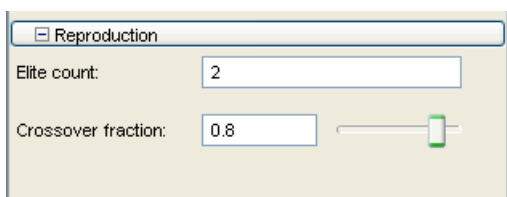


Рис. 4.21. Выбор опций репродукции

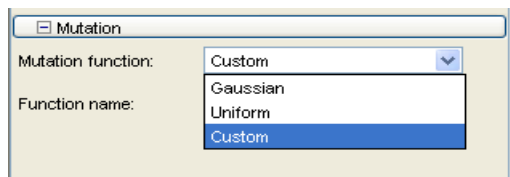


Рис. 4.22. Выбор опций мутации

Gaussian – гауссова мутация. Добавляет случайное число, заданное гауссовым распределением с нулевым средним для каждого компонента входного вектора. Вид распределения определяется параметрами Scale и Shrink.

Параметр Scale (масштаб) определяет величину изменений в первом поколении. Если диапазоны изменения n переменных заданы вектором V размером $2 \times n$, то диапазон изменения i -й переменной равен

$$\text{var}_{1,i} = \text{Scale} * (v(2, i) - v(1, i)).$$

Параметр Shrink (сжатие) определяет скорость уменьшения шага мутации с ростом количества поколений:

$$\text{var}_{k,i} = \text{var}_{k-1,i} \left(1 - \text{shrink} \cdot \frac{k}{\text{generation}} \right).$$

Uniform – стандартная мутация. При этом способе выполняются два шага. На первом шаге отбираются компоненты хромосом, подлежащие мутации (их доля указывается параметром Rate). На втором шаге отобранные компоненты получают случайные значения в пределах своей области определения.

Custom – использование функции мутации, определяемой пользователем.

Опции скрещивания (crossover) показаны на рис. 4.23.

Scattered – распределенный кроссовер (см. uniform crossover в п. 2.5.2). Пусть имеются две родительские хромосомы

$$X = [a \ b \ c \ d \ e \ f \ g \ h];$$

$$Y = [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8],$$

генерируется случайный двоичный вектор

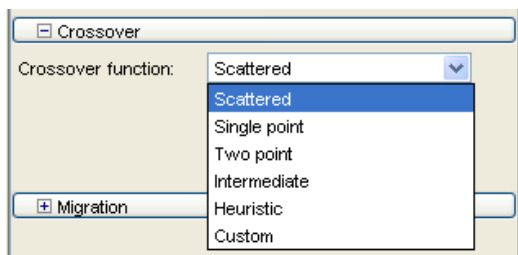


Рис. 4.23. Опции кроссовера

$$[1\ 1\ 0\ 0\ 1\ 0\ 0\ 0],$$

указывающий, откуда брать компоненты в дочернюю хромосому:

$$P = [a\ b\ 3\ 4\ e\ 6\ 7\ 8].$$

Single point – однотоочечный кроссовер. Пусть имеются две родительские хромосомы

$$X = [a\ b\ c\ d\ e\ f\ g\ h];$$

$$Y = [1\ 2\ 3\ 4\ 5\ 6\ 7\ 8].$$

Затем генерируется случайное число – точка скрещивания. Например, если точка скрещивания $n = 3$, то получаем хромосому-потомка

$$P = [a\ b\ c\ 4\ 5\ 6\ 7\ 8].$$

Two point – двухточечный кроссовер. Здесь генерируются два случайных числа – точки скрещивания. Например, если точки скрещивания $n = 3$ и $m = 6$, то для приведенных выше родительских хромосом получаем хромосому-потомка

$$P = [a\ b\ c\ 4\ 5\ 6\ g\ h].$$

Следующие два варианта (Intermediate и Neuristic) используются при представлении генов действительными числами.

Intermediate – промежуточный кроссовер, который выполняется по формуле

$$P = X + rand \cdot Ratio \cdot (Y - X),$$

где $rand$ – случайное число, $rand \in [0, 1]$; $Ratio$ – коэффициент, используемый при промежуточном скрещивании.

Neuristic – в этом варианте кроссовера используется эвристическая формула

$$P = Y + R * (X - Y).$$

Ген-потомок находится на линии, соединяющей гены родителей. По умолчанию $R = 1,2$, поэтому потомок располагается близко к родителю с лучшей фитнес-функцией (X) и на удалении от родителя с худшей фитнес-функцией (Y).

Опции миграции (migration) описывают перемещение хромосом между субпопуляциями, которые оказываются заданными в случае, если значение Population size представляет собой вектор, с длиной больше 1. Во время миграции копии лучших хромосом одной популяции заменяют собой худшие хромосомы другой популяции.

Параметры миграции определяются тремя полями (рис. 4.24).

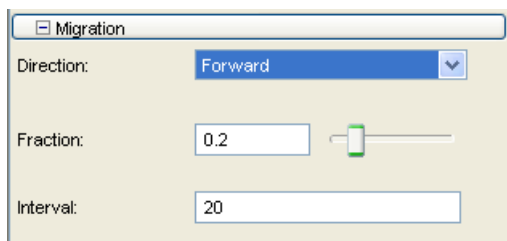


Рис. 4.24. *Параметры миграции*

Direction – направление миграции. Здесь возможны два варианта:

- Forward – миграция по направлению к последней субпопуляции, т. е. от n -й к $(n + 1)$ -й субпопуляции;
- Both – миграция в обоих направлениях.

Fraction – определяет, сколько хромосом перемещается между субпопуляциями. Эта величина относится к меньшей из двух субпопуляций, между которыми происходит перемещение. Например, если хромосомы перемещаются из популяции размером 50 в популяцию размером 100 и величина Fraction = 0,1, то перемещается $50 \times 0,1 = 5$ хромосом.

Interval – определяет, через сколько поколений происходит миграция.

Гибридные функции представляют собой другие алгоритмы оптимизации, которые запускаются после завершения работы ГА. На рис. 4.25 представлены варианты гибридных функций.

Варианты гибридных функций:

- none – не использовать гибридные функции;
- fminsearch – использование функции fminsearch для выполнения минимизации без ограничений;
- patternsearch – использование поиска по образцу для выполнения минимизации без ограничений;
- fminunc – использование функции fminunc для выполнения минимизации без ограничений;

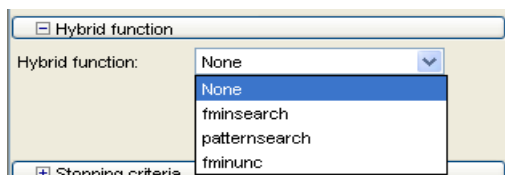


Рис. 4.25. *Опции гибридных функций*

– `fmincon` — использование функции `fmincon` для выполнения минимизации с ограничениями.

Варианты критериев остановки алгоритма показаны на рис. 4.26.

Для определения условий останова в опции `Stopping criteria` в `gatool` используются следующие пять вариантов условий.

`Generations` – алгоритм останавливается тогда, когда число поколений достигает заданного значения `Generations`.

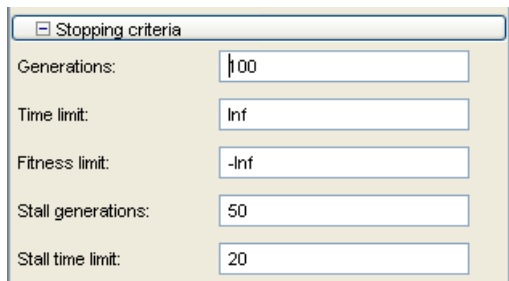
`Time limit` – алгоритм останавливается по истечении заданного времени в секундах `Time limit`.

`Fitness limit` – алгоритм останавливается тогда, когда значение функции пригодности для наилучшей точки текущего семейства будет меньше или равно `Fitness limit`.

`Stall generations` – алгоритм останавливается в случае, если нет улучшения для целевой функции в последовательности следующих друг за другом поколений длиной `Stall generations`.

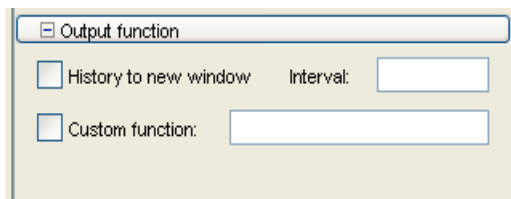
`Stall time limit` – алгоритм останавливается в случае, если нет улучшения для целевой функции в течение интервала времени в секундах, равного `Stall time limit`.

Опции вывода функций (`output function`) показаны на рис. 4.27.



Stopping criteria	
Generations:	100
Time limit:	Inf
Fitness limit:	-Inf
Stall generations:	50
Stall time limit:	20

Рис. 4.26. Критерии остановки алгоритма



Output function	
<input type="checkbox"/> History to new window	Interval: <input type="text"/>
<input type="checkbox"/> Custom function:	<input type="text"/>

Рис. 4.27. Окно опций вывода функций

При выборе History to new window графики отображаются в новом окне через каждые Interval итераций.

При выборе Custom можно определить пользовательскую функцию вывода.

Опций отображения в командном окне (display to command window) показаны на рис. 4.28.

Off – нет вывода на дисплей.

Iterative – информация выводится на каждой итерации.

Diagnose – информация выводится на каждой итерации и снабжается информацией об измененных параметрах алгоритма.

Final – отображается причина завершения работы.

Опция векторизации (vectorize) (рис. 4.29) выбирается в том случае, если фитнес-функция векторизована.

Генетический алгоритм обычно работает быстрее при векторизации фитнес-функции. Это происходит за счет использования операций матричной обработки.

Например, пусть требуется найти минимум функции

$$f(x_1, x_2) = x_1^2 - 2x_1x_2 + 6x_1 + x_2^2 - 6x_2.$$

Эту функцию можно описать с помощью М-файла:

```
function z = my_fun(x)
```

```
z = x(1)^2 - 2*x(1)*x(2) + 6*x(1) + x(2)^2 - 6*x(2);
```

Вычисления ускоряются, когда вторая строка файла принимает вид

```
z = x(:,1).^2 - 2*x(:,1).*x(:,2) + 6*x(:,1) + x(:,2).^2 - 6*x(:,2);
```

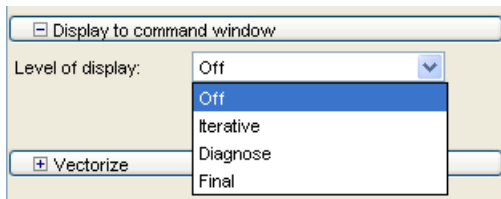


Рис. 4.28. Опции отображения в командном окне

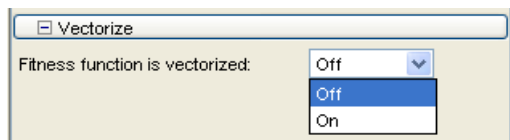


Рис. 4.29. Опции векторизации

4.3. Запуск из командной строки и М-файла

Интерфейс `gatoool` значительно упрощает использование ГА, однако для полноценной работы со всеми возможностями `MatLab` необходимо также уметь использовать командную строку для вызова ГА и уметь включать ГА в структуру М-файлов.

Существуют различные варианты запуска ГА из командной строки в зависимости от заданных линейных и (или) нелинейных ограничений:

```
>>x = ga(@fitnessfcn,nvars)
>>x = ga(@fitnessfcn,nvars,A,b)
>>x = ga(@fitnessfcn,nvars,A,b,Aeq,beq)
>>x = ga(@fitnessfcn,nvars,A,b,Aeq,beq,LB,UB)
>>x = ga(@fitnessfcn,nvars,A,b,Aeq,beq,LB,UB, @nonlcon)
>>x = ga(@fitnessfcn,nvars,A,b,Aeq,beq,LB,UB, @nonlcon,options)
```

где `fitnessfcn` – минимизируемая функция (знак `@` – указатель); `nvars` – количество аргументов этой функции; `A,b,Aeq,beq,LB,UB` и `nonlcon` – ограничения, рассмотренные в подразд. 4.2; `options` – структура, содержащая параметры ГА; если она не указана, то используются стандартные параметры.

Если какое-то из ограничений не задано, то на его месте указывается `[]`, например:

```
>>x = ga(@fitnessfcn,nvars,A,b,Aeq,beq,[],[], [],options).
```

Параметр `options` позволяет изменять многочисленные параметры ГА, используемые по умолчанию. Для этого следует использовать функцию `gaoptimset`.

При запуске ГА с опциями, установленными по умолчанию, можно указать только два входных и выходных параметра:

```
>> [x fval] = ga(@fitnessfun, nvars)
```

где `x` – возвращаемое алгоритмом окончательное решение; `fval` – значение функции пригодности в конечной точке.

Можно также использовать дополнительные выходные параметры. В этом случае ГА вызывается выражением вида

```
>> [x fval reason output population scores] = ga(@fitnessfcn, nvars)
```

Здесь имеются дополнительные выходные аргументы:

- `output` – структура, содержащая информацию о выполнении алгоритма в каждом поколении;
- `population` – популяция последнего поколения (окончательная);
- `scores` – окончательное значение пригодности.

Узнать текущее значение параметров можно следующей командой:

```
>> options = gaoptimset
```

которая возвращает следующий набор полей:

```
options =  
    PopulationType: 'doubleVector'  
    PopInitRange: [2x1 double]  
    PopulationSize: 20  
    EliteCount: 2  
    CrossoverFraction: 0.8000  
    MigrationDirection: 'forward'  
    MigrationInterval: 20  
    MigrationFraction: 0.2000  
    Generations: 100  
    TimeLimit: Inf  
    FitnessLimit: -Inf  
    StallGenLimit: 50  
    StallTimeLimit: 20  
    TolFun: 1.0000e-006  
    TolCon: 1.0000e-006  
    InitialPopulation: []  
    InitialScores: []  
    InitialPenalty: 10  
    PenaltyFactor: 100  
    PlotInterval: 1  
    CreationFcn: @gacreationuniform  
    FitnessScalingFcn: @fitscalingrank  
    SelectionFcn: @selectionstochunif  
    CrossoverFcn: @crossoverscattered  
    MutationFcn: @mutationgaussian  
    HybridFcn: []  
    Display: 'final'  
    PlotFcns: []  
    OutputFcns: []  
    Vectorized: 'off'
```

Можно просматривать и менять значения отдельных полей. Например, узнать текущий размер популяции можно командой

```
>> options.PopulationSize  
ans =  
    20
```

т. е. популяция содержит 20 хромосом.

Задать новый размер популяции можно командой

```
>> options = gaoptimset('PopulationSize', 100)
```

Изменить процент мутирующих хромосом можно командой

```
>> options = gaoptimset('CrossoverFraction', 0.9)
```

Таким же образом можно выбирать вариант генетической операции. Например, выбрать операцию селекции можно командой

```
>> options = gaoptimset('SelectionFcn',@selectiontournament)
```

Для выбора измененных опций ГА при запуске из командной строки следует использовать команду вида

```
» [x fval] = ga(@fitnessfun, nvars, [],[],[],[],[],[],[],options).
```

При многократном запуске ГА с изменяемыми параметрами необходимо использовать М-файлы. Например, если мы хотим исследовать влияние количества скрещиваний хромосом на процесс решения задачи. Пусть доля хромосом популяции, которые подвергаются кроссоверу, меняется от 0 до 1 с шагом 0.05. Для этого требуется циклически изменять значение `options.CrossoverFraction`. Эту операцию выполняет следующий фрагмент программы:

```
options = gaoptimset('Generations',300);  
rand('state', 71); % установка генераторов  
randn('state', 59); % псевдослучайных чисел  
record=[];  
for n=0:.05:1  
    options = gaoptimset(options,'CrossoverFraction', n);  
    [x fval]=ga(@rastriginsfcn, 10,[],[],[],[],[],[],[],options);  
    record = [record; fval]; % добавление нового значения результат  
end
```

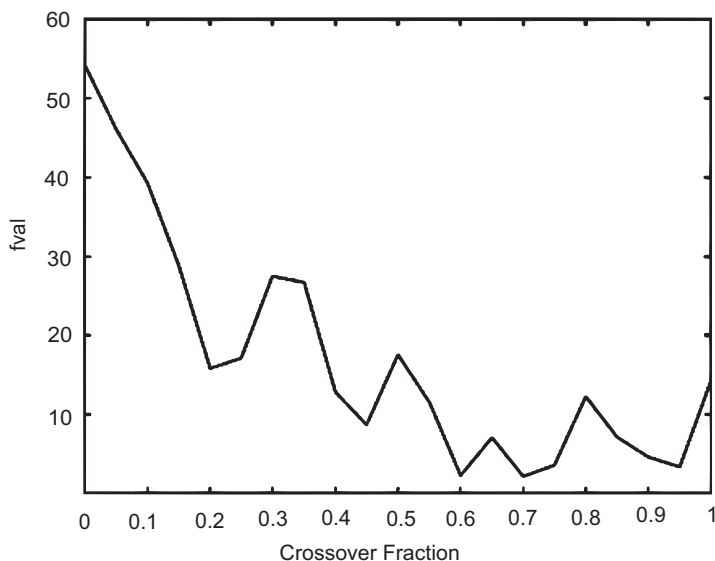


Рис. 4.30. Исследование качества решения задачи от процента скрещиваний

Для просмотра результирующего графика можно использовать команды:

```
plot(0:.05:1, record);  
xlabel('Crossover Fraction');  
ylabel('fval')
```

График на рис. 4.30 показывает, что лучшее значение процента скрещиваний лежит в диапазоне от 0.6 до 0.95, поскольку в этом диапазоне функция принимает наименьшее значение.

В процессе работы с GATool пользователь может запомнить решаемую проблему в рабочую память. После этого можно запустить задачу на решение из командной строки (или М-файла) с помощью обращения

```
>> x = ga(problem)
```

где problem – название запомненной задачи.

Вопросы для самопроверки

1. Как называется пакет расширения (toolbox) MatLab, содержащий генетический алгоритм?
2. Что понимается в MatLab под функцией относительной пригодности?
3. Как трактуется хромосома при использовании MatLab?
4. Как описывается популяция средствами MatLab?
5. Перечислите команды, используемые при работе с ГА в MatLab.
6. Какие панели имеются в графическом окне GATool?
7. Перечислите опции, которыми располагает пользователь для описания работы ГА в MatLab.
8. Как описать целевую функцию для ГА в MatLab?
9. Возможен ли переход от задачи максимизации к задаче минимизации?
10. Какие виды ограничений можно использовать при описании целевой функции в MatLab?
11. Перечислите основные графики, которые можно просматривать при анализе работы ГА в MatLab.
12. Какие опции популяции реализованы в MatLab?
13. Какие имеются возможности для описания функции пригодности в MatLab?
14. Какие варианты генетических операций можно использовать при работе в MatLab?
15. Что можно описать с помощью опций миграции?
16. Зачем используются гибридные функции?

17. Какие существуют критерии остановки ГА в MatLab?
18. Какие преимущества несет векторизация фитнес-функции?
19. Какие параметры можно указывать при вызове ГА из командной строки MatLab?
20. В чем заключается необходимость использования М-файлов при работе с ГА в MatLab?

5. РЕШЕНИЕ ЗАДАЧ В MATLAB

5.1. Минимизация функций

Рассмотрим различные варианты постановки задач минимизации функций.

Вариант 1 – минимизация без ограничений.

Пусть имеется функция одной независимой переменной

```
function s = test(x)
    s = 20+x^2+10*(cos(2*pi*x))
end
```

Эта функция должна быть записана в один из рабочих каталогов MatLab под именем test.m.

Для нахождения минимума тестовой функции нужно ввести команду

```
>> [x fval reason] = ga(@test, 1)
```

По окончании работы алгоритма будут выданы следующие сообщения:

```
x =
    0.4988
      fval =
    10.2491
reason =
Optimization terminated: maximum number of generations exceeded.
```

Найденный минимум соответствует графику, показанному на рис. 4.2. Алгоритм остановил работу после достижения заданного числа поколений.

Вариант 2 – минимизация с линейными ограничениями.

Рассмотрим поиск минимума функции [71]

$$F(x) = 8x_1 + 9x_2 + 13x_3 \rightarrow \min$$

при ограничениях

$$\begin{cases} x_1 + 6x_2 + 20x_3 \geq 90 \\ 3x_1 + 3x_2 + x_3 \geq 70 \\ x_1 + x_2 + x_3 \geq 25 \\ x_1 \geq 0 \\ x_2 \geq 0 \\ x_3 \geq 0 \end{cases}.$$

Целевая функция описывается с помощью файла

```
function s = test10(x)
    s = 8*x(1)+9*x(2)+13*x(3)
end
```

Заданные ограничения являются линейными, поэтому их требуется привести к виду

$$Ax \leq b; A_{eq}x \leq b_{eq}.$$

Выполняя очевидные преобразования, имеем:

```
>> A = A = [-1 -6 -20; -3 -3 -1; -1 -1 -1];
>> B = [-90; -70; -25];
>> Lb = [0; 0; 0];
```

Запуск ГА происходит с помощью команды

```
>> [x fval reason] = ga(@test10,3,A,B,[],[],Lb,[])
```

Генетический алгоритм легко находит решение задачи:

```
x =
    13.8176 10.4899 0.6945
fval =
    213.9784
reason =
Optimization terminated:
average change in the fitness value less than options.
```

Полученный ответ близок решению, полученному в работе [71] с помощью симплекс-метода.

Вариант 3 – минимизация с нелинейными ограничениями.

Пусть минимизируемая функция и ограничения заданы в виде

$$\begin{aligned} f(x) &= 100(x_1^2 - x_2)^2 + (1 - x_1)^2; \\ x_1 x_2 + x_1 - x_2 + 1,5 &\leq 0 \\ 10 - x_1 x_2 &\leq 0 \\ 0 \leq x_1 &\leq 1 \\ 0 \leq x_2 &\leq 13 \end{aligned}$$

Целевая функция описывается с помощью файла

```
function s = test20(x)
    s = 100*(x(1)^2-x(2))^2+(1-x(1))^2
end
```

Нелинейные ограничения в виде неравенств можно описать с помощью файл-функции

```
function [c, ceq] = nlc(x)
c = [x(1)*x(2) + x(1) - x(2)+1.5; 10 -x(1)*x(2)];
ceq = [];
```

Границы изменения переменных:

```
>> Lb = [0; 0];  
>> Ub = [1; 13];
```

Запуск ГА происходит с помощью команды

```
>> [x fval reason] = ga(@test20,2,[],[],[],[],Lb,Ub,@nlc)
```

Получено решение задачи:

```
x =  
    0.8122 12.3122  
fval =  
    1.3578e+004  
reason =  
Optimization terminated: current tolerance on f(x) 1e-007 is less than  
options.TolFun and constraint violation is less than options.TolCon.
```

5.2. Решение диофантова уравнения

В качестве примера работы ГА был рассмотрен процесс решения диофантова уравнения (разд. 1). Пусть задача формулируется в виде

$$\begin{cases} A + B + C + D \rightarrow \min; \\ A + 2B + 3C + 4D = 30 \\ A > 0 \\ B > 0 \\ C > 0 \\ D > 0 \end{cases}.$$

Опишем, как эта задача может быть решена с помощью GATool.

Решением диофантова уравнения являются целые числа, поэтому популяция должна быть выбрана двоичного типа (рис. 5.1).

Заметим, что размер популяции в 300 хромосом охватывает всего около 0,03% пространства поиска.

Можно считать, что каждая переменная находится в диапазоне [0, 27], поэтому для кодирования одной переменной требуется 5 би-

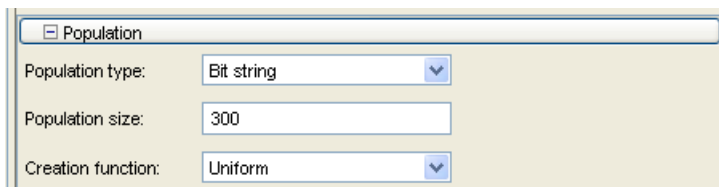


Рис. 5.1. Описание популяции для решения диофантова уравнения

тов двоичного кода, а вся хромосома должна состоять из 20 двоичных переменных (рис. 5.2).

Для перехода от двоичного к десятичному представлению требуется выполнить элементарное преобразование, пример которого показан на рис. 5.3.

Необходимость такого преобразования затрудняет использование окон для ввода ограничений в GATool, поэтому равенство и неравенства можно включить в описание минимизируемой функции:

```
function z=diofant(X)
A=X(5)*16+X(4)*8+X(3)*4+X(2)*2+X(1)*1;
B=X(10)*16+X(9)*8+X(8)*4+X(7)*2+X(6)*1;
C=X(15)*16+X(14)*8+X(13)*4+X(12)*2+X(11)*1;
D=X(20)*16+X(19)*8+X(18)*4+X(17)*2+X(16)*1;
Y=A+2*B+3*C+4*D;
if (A==0)|(B==0)|(C==0)|(D==0)|(Y~=30) z=100;
elseif (Y==30) z=A+B+C+D; end

end
```

В приведенном описании присвоение $z=100$ означает, что целевая функция получает большое значение, если не выполняется хотя бы одно из условий типа равенства или неравенства.

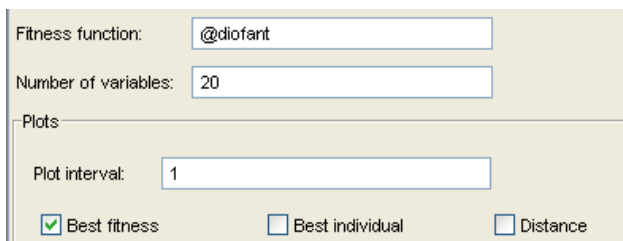


Рис. 5.2. Описание хромосом для решения диофантова уравнения

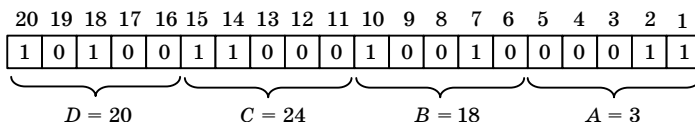


Рис. 5.3. Преобразование двоичной хромосомы в десятичные числа

Из анализа графика средней и лучшей пригодности хромосом (рис. 5.4) следует, что сначала все хромосомы имеют низкое значение пригодности. Решение задачи (со значением пригодности, равным 12), обнаруживается в 26-м поколении, после чего средняя пригодность хромосом популяции быстро изменяется, приближаясь к пригодности найденного решения.

Двоичная хромосома-решение, найденная в процессе генетического поиска, показана на рис. 5.5. Это решение единственное (глобальный экстремум), поскольку сумма $A+B+C+D$ минимальна.

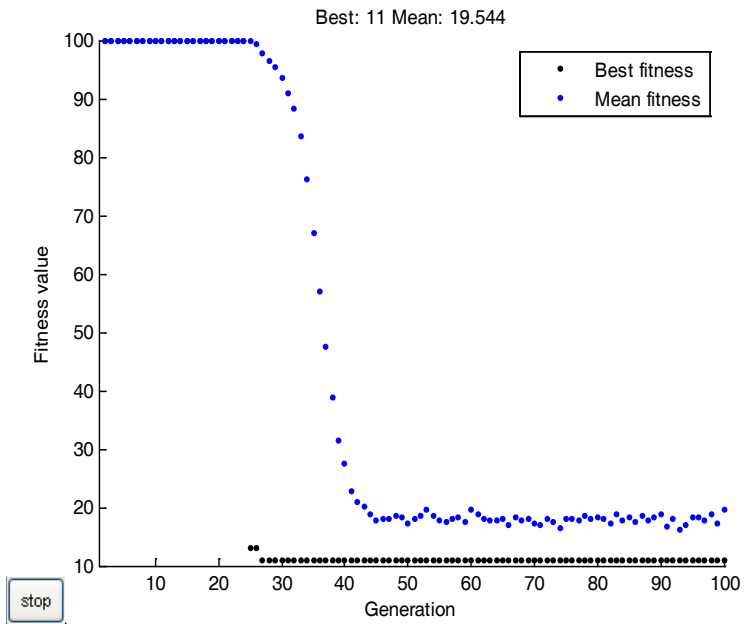


Рис. 5.4. Изменение пригодности хромосом при решении диофантова уравнения

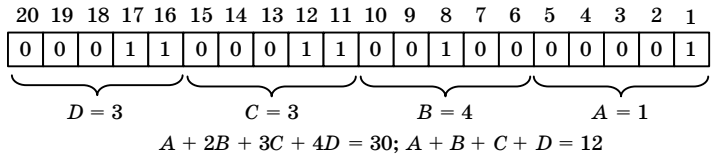


Рис. 5.5. Решение диофантова уравнения

5.3. Настройка ПИД-регулятора

Рассмотрим процесс настройки ПИД-регулятора для объекта, заданного передаточной функцией

$$W_1(s) = \frac{2s+1}{0,5s^2 - s + 1}. \quad (5.1)$$

Реакция этого объекта на единичное ступенчатое воздействие показана на рис. 5.6. Очевидно, объект является неустойчивым.

Допустим, что желаемый процесс в системе описывается с помощью передаточной функции

$$W_2(s) = \frac{1}{0,5s+1}. \quad (5.2)$$

Переходный процесс для этой передаточной функции показан на рис. 5.7. Он является аperiодическим, а время переходного процесса – около 2 с.

Схема моделирования в Simulink MatLab показана на рис. 5.8. Блоки `simout` и `simout1` служат для накопления двумерных массивов «время-выходная координата».

Для получения массивов выходных координат нужно использовать в Simulink MatLab метод интегрирования с постоянным шагом. Так, при выборе шага 0,01 с и времени моделирования 3 с будет получено 300 точек, которые будут помещены в массивы `simout` и `simout1`.

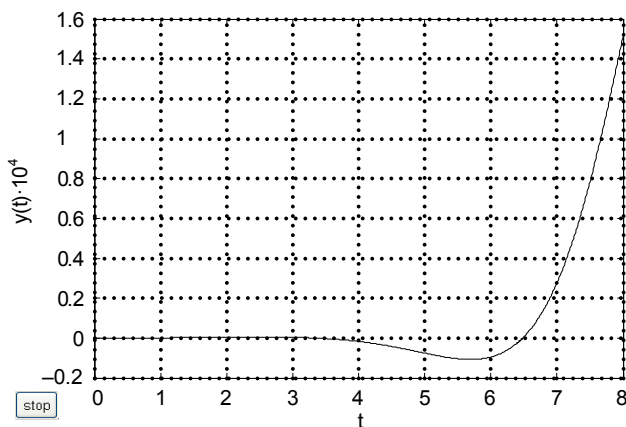


Рис. 5.6. Реакция разомкнутой системы

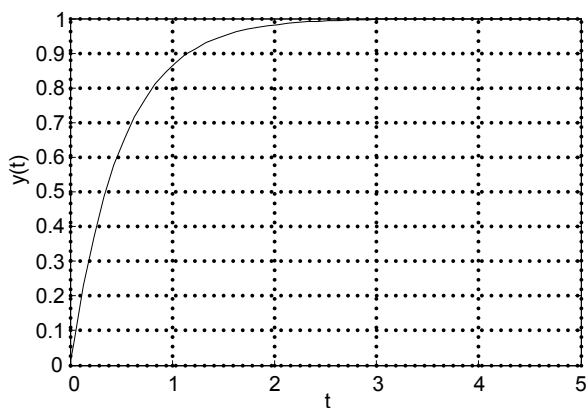


Рис. 5.7. Описание эталонного процесса

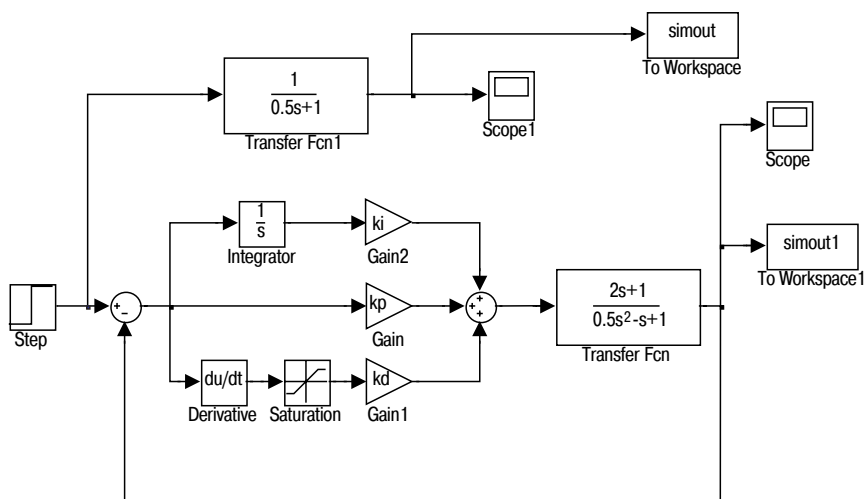


Рис. 5.8. Схема моделирования в Simulink MatLab

Поскольку массивы `simout` и `simout1` имеют одинаковую размерность, функцию ошибки можно описать в виде

$$E = \sum |simout - simout1|. \quad (5.3)$$

Тогда целевую функцию можно описать с помощью файла

```
function z=pid_reg(X);  
global kp,  
global kd;  
global ki;  
kp=X(1); kd=X(2); ki=X(3);  
sim('pid2');  
z=sum(abs(simout-simout1));  
end
```

Переменные, по которым ведется поиск, должны быть объявлены глобальными в командной строке MatLab:

```
>> global kp;  
>> global kd;  
>> global ki;
```

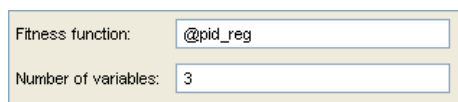
Заметим, что операции присваивания

```
kp=X(1); kd=X(2); ki=X(3);
```

соответствуют преобразованию «генотип – фенотип».

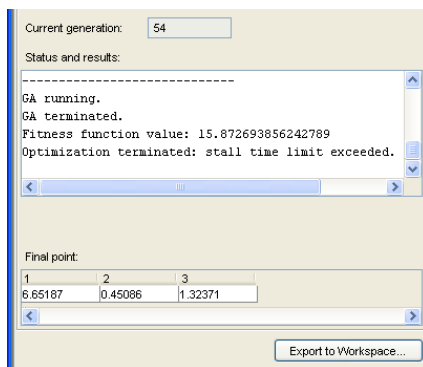
После запуска GaTool следует указать файл с описанием целевой функции и количество переменных, по которым ведется поиск (рис. 5.9). Остальные параметры приняты по умолчанию.

Фрагмент экрана GaTool после окончания процесса оптимизации показан на рис. 5.10.



The screenshot shows a window with two input fields. The first field is labeled "Fitness function:" and contains the text "@pid_reg". The second field is labeled "Number of variables:" and contains the number "3".

Рис. 5.9. Параметры запуска GaTool



The screenshot shows a window with the following content:

Current generation: 54

Status and results:

GA running.
GA terminated.
Fitness function value: 15.872693856242789
Optimization terminated: stall time limit exceeded.

Final point:

1	2	3
6.65187	0.45086	1.32371

Export to Workspace...

Рис. 5.10. Состояние после окончания оптимизации

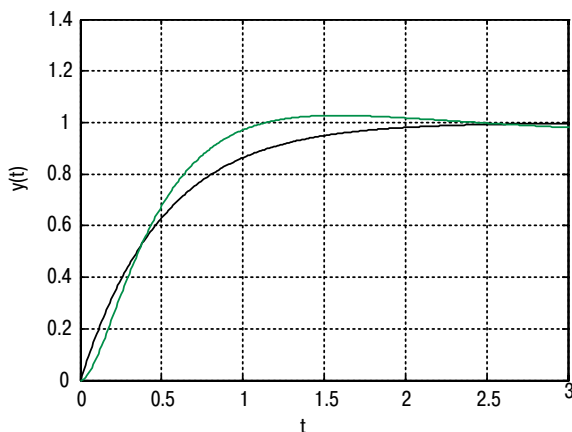


Рис. 5.11. Результат генетической настройки ПИД-регулятора

Для поиска решения потребовалось всего около 50 шагов, алгоритм остановился после того, как целевая функция перестала улучшаться.

На рис. 5.11 показано сравнение эталонного переходного процесса и процесса при полученных с помощью ГА коэффициентах. Очевидно, что качество решения удовлетворительное.

5.4. Настройка нечеткого регулятора

5.4.1. Настройка масштабирующих коэффициентов

Общая постановка задачи синтеза НЛР, которая допускает разнообразные конкретизации, описана в п. 3.1.3. Рассмотрим в качестве примера вариант генетического синтеза НЛР ПД-типа.

Нечеткий логический регулятор ПД-типа для каждой пары значений ошибки и ее производной $\{e, de/dt\}$ вычисляет значение управления:

$$u^*(t) = F_{\text{л}}(e^*(t), (de(t)/dt)^*),$$

где $u^*(t)$, $e^*(t)$, $(de(t)/dt)^*$ – нечеткие значения управления, ошибки и ее производной; $F_{\text{л}}$ – закон управления, заданный набором правил (рис. 5.12).

Если $g(t)$ – задающее воздействие (уставка), а $y(t)$ – сигнал на выходе объекта, то ошибка управления вычисляется по формуле

$$e(t) = g(t) - y(t).$$

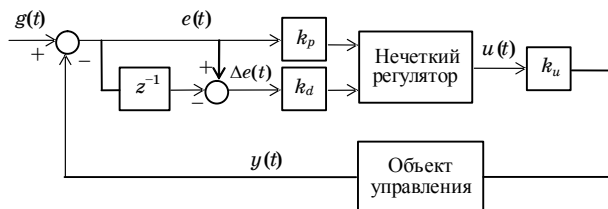


Рис. 5.12. Нечеткий логический регулятор ПД-типа

Отрицательное значение ошибки $e(t)$ означает, что выходной сигнал больше уставки $g(t)$, и его нужно уменьшать; положительное значение $e(t)$ означает, что выходной сигнал меньше уставки, и его нужно увеличивать. Нулевое значение $e(t)$ соответствует, очевидно, равенству $g(t)$ и $y(t)$.

Рассмотрим приращение ошибки управления:

$$\Delta e(t) = e(t) - e(t-1) = g(t) - y(t) - g(t) - y(t-1) = y(t-1) - y(t).$$

Отрицательный знак $\Delta e(t)$ означает, что выходной сигнал уменьшается; положительное значение $\Delta e(t)$ означает, что выходной сигнал увеличивается; нулевое значение $\Delta e(t)$ соответствует постоянному выходному сигналу.

Поведение ошибки в течение переходного процесса показано на рис. 5.13.

Сделанные замечания позволяют классифицировать все возникающие при управлении ситуации (табл. 5.1).

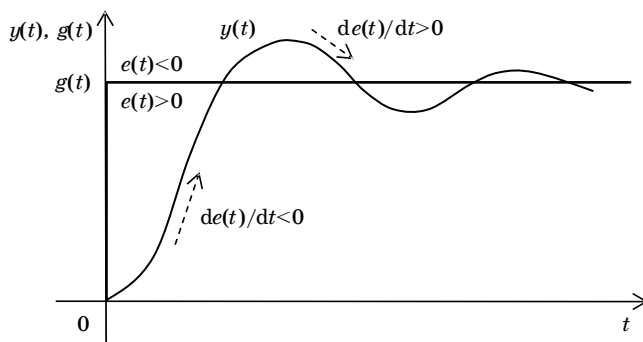


Рис. 5.13. Изменение ошибки при нормальном протекании переходного процесса

Таблица 5.1. Семантическое описание процесса управления

Описание ситуации		Характеристика ситуации	Управление
Знак $e(t)$	Знак $\Delta e(t)$		
–	+	$y(t) > g(t)$ и $y(t) \downarrow$	Не требуется (0)
–	–	$y(t) > g(t)$ и $y(t) \uparrow$	Уменьшение (–)
–	0	$y(t) > g(t)$ и $y(t) = \text{const}$	Уменьшение (–)
0	–	$y(t) = g(t)$ и $y(t) \uparrow$	Уменьшение (–)
0	+	$y(t) = g(t)$ и $y(t) \downarrow$	Увеличение (+)
0	0	$y(t) = g(t)$ и $y(t) = \text{const}$	Не требуется (0)
+	+	$y(t) < g(t)$ и $y(t) \downarrow$	Увеличение (+)
+	–	$y(t) < g(t)$ и $y(t) \uparrow$	Не требуется (0)
+	0	$y(t) < g(t)$ и $y(t) = \text{const}$	Увеличение (+)

На основании табл. 5.1 можно сформулировать закон управления в виде таблицы лингвистических правил (табл. 5.2), где «Н» – нулевое значение, «П» – положительное, «О» – отрицательное.

Таблица 5.2. Лингвистические правила при трех термах ЛП

Таблица правил		$\Delta e^*(t)$		
		О	Н	П
$e^*(t)$	О	О	О	Н
	Н	О	Н	П
	П	Н	П	П

Табл. 5.2 содержит 9 правил управления (лингвистическое значение управления находится на пересечении строки и столбца). Эти правила имеют универсальный характер, и могут использоваться для любого объекта управления невысокого порядка.

Может оказаться, что при трех термах закон управления будет слишком грубым, близким к релейному. Чтобы избежать этого, добавим еще по два терма, разбив каждое значение «О» и «П» на два: «ОМ», «ОБ» и «ПМ», «ПБ» («Отрицательный Малый», «Отрицательный Большой» и «Положительный Малый», «Положительный Большой»). Тогда описание правил может быть задано табл. 5.3, которая описывает, очевидно, уже 25 правил.

Поскольку нечеткие правила заданы, для синтеза НЛР требуется выбрать описание ЛП, входящих в правила.

Таблица 5.3. Лингвистические правила при пяти термах ЛП

Таблица правил		$\Delta e^*(t)$				
		ОБ	ОМ	Н	ПМ	ПБ
$e^*(t)$	ОБ	ОБ	ОБ	ОМ	ОМ	Н
	ОМ	ОБ	ОМ	ОМ	Н	ПМ
	Н	ОМ	ОМ	Н	ПМ	ПМ
	ПМ	ОМ	Н	ПМ	ПМ	ПБ
	ПБ	Н	ПМ	ПМ	ПБ	ПБ

Будем использовать при синтезе НЛР пакет Fuzzy Logic toolbox из библиотеки Simulink [72]. В этом пакете с помощью графического редактора FIS Edit, вызываемого командой

```
>> fuzzy
```

можно описать ЛП, правила управления, а также просмотреть поверхность отклика регулятора.

При описании ЛП следует учитывать две общие рекомендации:

– необходимо, чтобы термы ЛП A_i обеспечивали нечеткое разбиение базовой шкалы U так, чтобы примерно выполнялось условие

$$\sum_{i=1}^N \mu_{A_i}(x) = 1, \quad \forall x \in U;$$

– термы должны сгущаться в области нуля и разряжаться по краям базовой шкалы.

Первая из этих рекомендаций гарантирует отсутствие «белых пятен» в базе правил, а вторая обеспечивает повышение точности управления [73].

Таким образом, каждая из ЛП может быть единообразно описана в виде, показанном на рис. 5.14.

На рис. 5.14 базовая шкала нормирована – приведена к диапазону $[-1, 1]$. Однако в реальности каждая ЛП обладает собственной базовой шкалой, которую требуется определить или уточнить. Таким образом, в рассмотренной постановке задача генетического синтеза НЛР сводится к поиску значений масштабирующих коэффициентов k_p, k_d, k_u (см. рис. 5.12) для каждой ЛП.

Окно ввода лингвистических правил FIS Edit показано на рис. 5.15.

Поверхность отклика регулятора при нормированных значениях переменных показана на рис. 5.16. Очевидно, она является

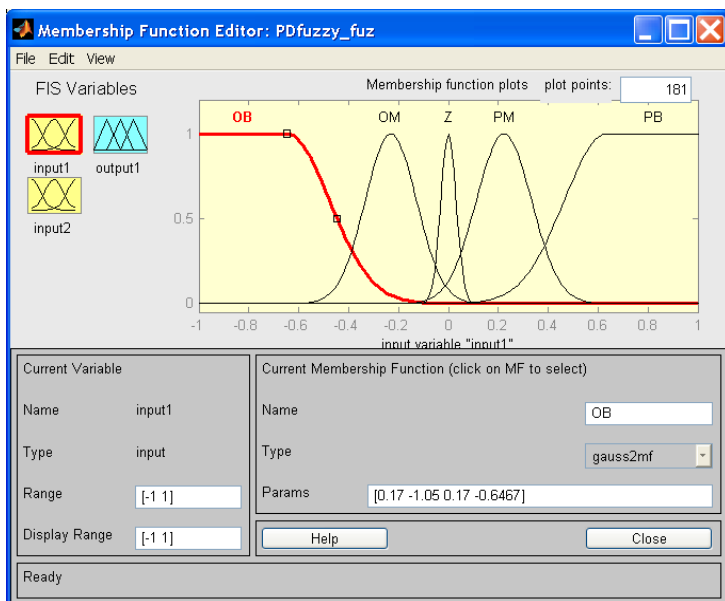


Рис. 5.14. Описание ЛП

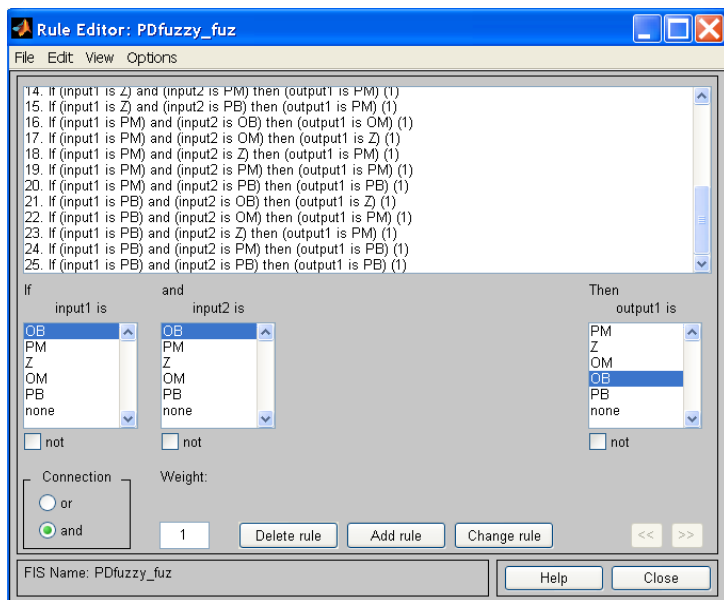


Рис. 5.15. Окно ввода лингвистических правил

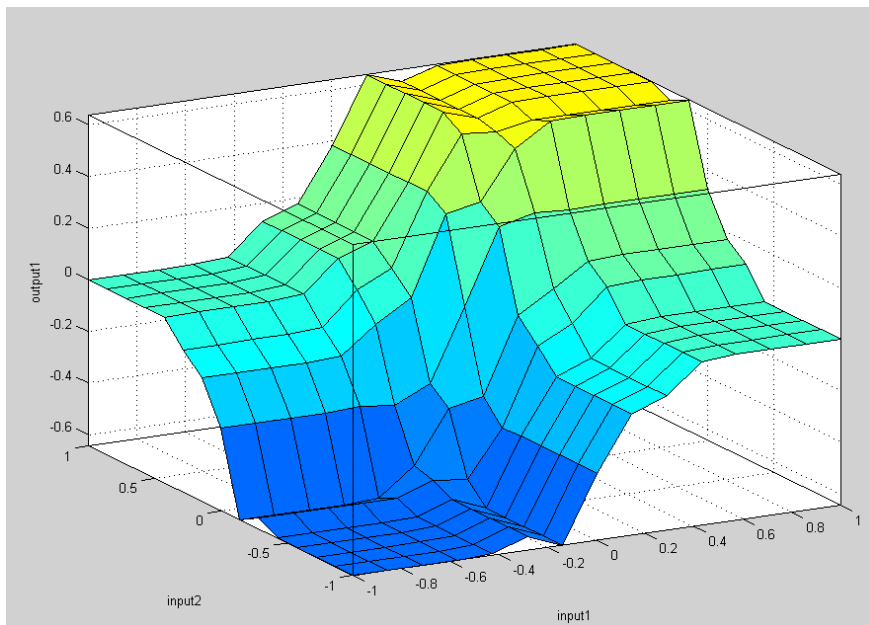


Рис. 5.16. Поверхность отклика НЛР

существенно нелинейной (для обычного ПД-регулятора это – плоскость).

Пусть объект управления описывается передаточной функцией (5.1). В разомкнутом состоянии этот объект неустойчив (см. рис. 5.6).

Схема генетического обучения НЛР показана на рис. 5.17. Задан более быстрый эталонный роцесс. Целевая функция при обучении НЛР будет почти такой же, как при обучении ПИД-регулятора:

```
function z=Fuzzy_reg(X);
global k1,
global k2;
global k3;
k1=X(1); k2=X(2); k3=X(3);
sim('Fuz_reg');
z=sum(abs(simout-simout1));
end
```

Параметры запуска GATool показаны на рис. 5.18. Остальные параметры приняты по умолчанию.

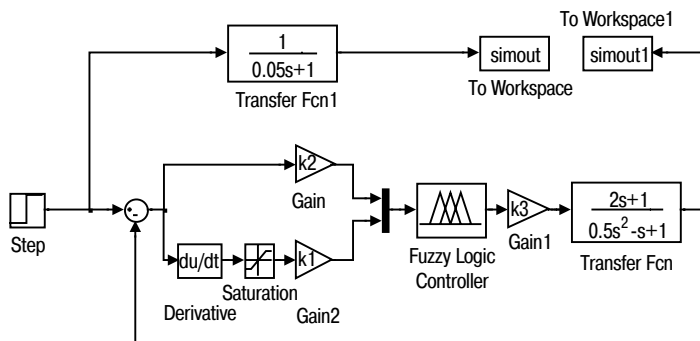


Рис. 5.17. Схема обучения НЛР в Simulink MatLab

Fitness function:	@Fuzzy_reg
Number of variables:	3

Рис. 5.18. Параметры запуска при настройке нечеткого контроллера

Фрагмент экрана GATool после окончания процесса оптимизации показан на рис. 5.19.

Получен несколько неожиданный результат – оказалось, что коэффициент k_1 почти равен нулю, т. е. при управлении не требуется учитывать значение производной.

Сравнение заданного и полученного переходных процессов показано на рис. 5.20. НЛР демонстрирует излишнюю чувствитель-

Current generation:	50						
Status and results:	<pre> ----- GA running. GA terminated. Fitness function value: 2.8602869813051477 Optimization terminated: maximum number of generations ex </pre>						
Final point:	<table border="1"> <thead> <tr> <th>1</th> <th>2</th> <th>3</th> </tr> </thead> <tbody> <tr> <td>-0.00551</td> <td>2.53749</td> <td>5.70175</td> </tr> </tbody> </table>	1	2	3	-0.00551	2.53749	5.70175
1	2	3					
-0.00551	2.53749	5.70175					

Рис. 5.19. Состояние после завершения поиска коэффициентов

ность в области малых отклонений от уставки. Этот недостаток может быть компенсирован более точным подбором параметров функций принадлежности (см. рис. 5.14).

Процесс изменения состояния популяции в процессе генетического поиска показан на рис. 5.21. Решение, близкое к оптималь-

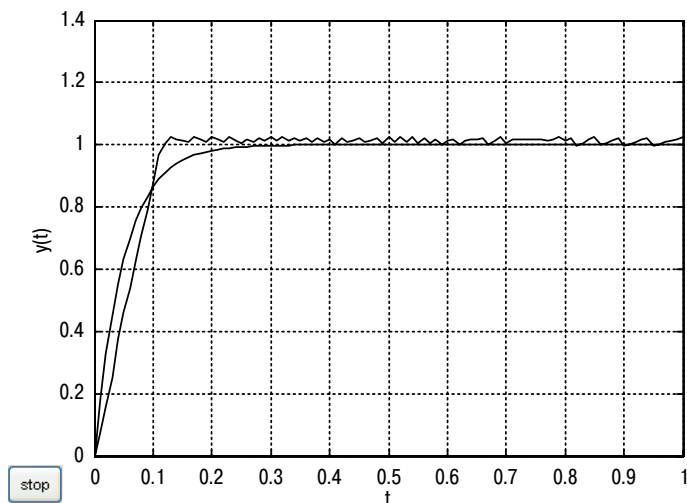


Рис. 5.20. Результат генетической настройки нечеткого регулятора

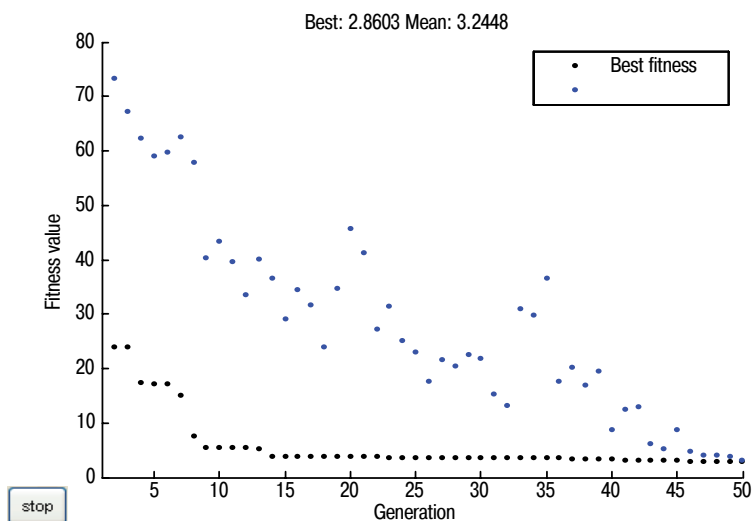


Рис. 5.21. Изменение пригодности хромосом при настройке НЛР

ному, было найдено достаточно быстро (в 13-м поколении). К 50-му поколению значение лучшей и средней пригодности практически сравнялось.

5.4.2. Настройка регулятора Сугэно

Нечеткий регулятор Сугэно отличается от НЛР Мамдани тем, что заключения нечетких правил представляют собой константы или линейные функции от входных переменных. Рассмотрим для простоты изложения вариант с константами.

Как было показано выше, заключения правил можно выбрать на основе эмпирических соображений. Однако такой подход не универсален, поскольку для объектов с нелинейностями, например, не всегда может быть понятно, какое управление необходимо в той или иной ситуации. Более универсальной можно считать следующую постановку задачи: имеется набор правил с заданными посылками (ситуациями), требуется определить (или уточнить) заключение каждого правила.

Пусть для описания входных переменных регулятора (ошибки и ее производной) используются ЛП, каждая из которых имеет три терма. Термы можно заранее расположить на базовых шкалах, обеспечив их нечеткое разбиение (рис. 5.22).

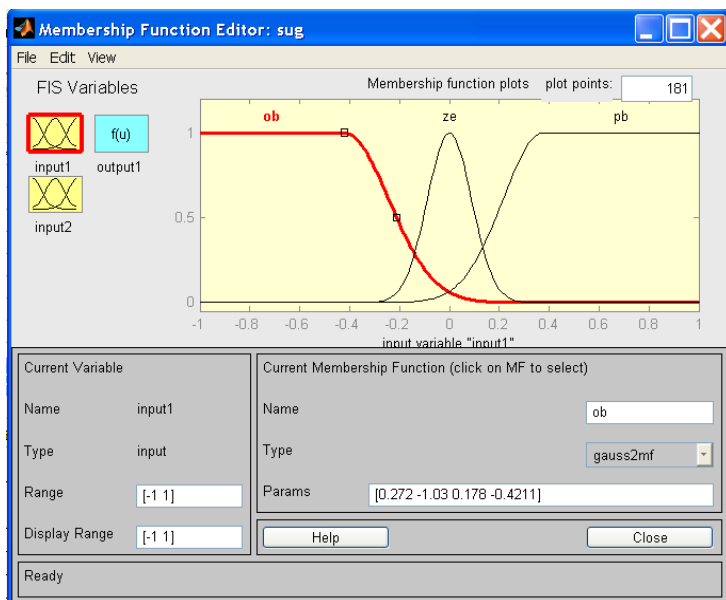


Рис. 5.22. Вариант нечеткого разбиения базовой шкалы

Тогда правила управления можно описать с помощью табл. 5.4, получающейся из табл. 5.2 при замене заключения правила константой (К1 – К6).

Табл. 5.4 описывает 9 управляющих правил, в каждое из которых входит одна неизвестная. Таким образом, ГА должен выполнить оптимизацию по 9 переменным, «правильно» расположив их на базовой шкале для управляющего сигнала.

Таблица 5.4. Правила для регулятора Сугэно

	Таблица правил	$\Delta e^*(t)$		
		О	Н	П
$e^*(t)$	О	К1	К2	К3
	Н	К4	К5	К6
	П	К7	К8	К9

На рис. 5.23 показано окно ввода правил для регулятора Сугэно.

Схема моделирования в Simulink приведена на рис. 5.24, где блоки saturation служат для приведения входных переменных в рамки базовой шкалы.

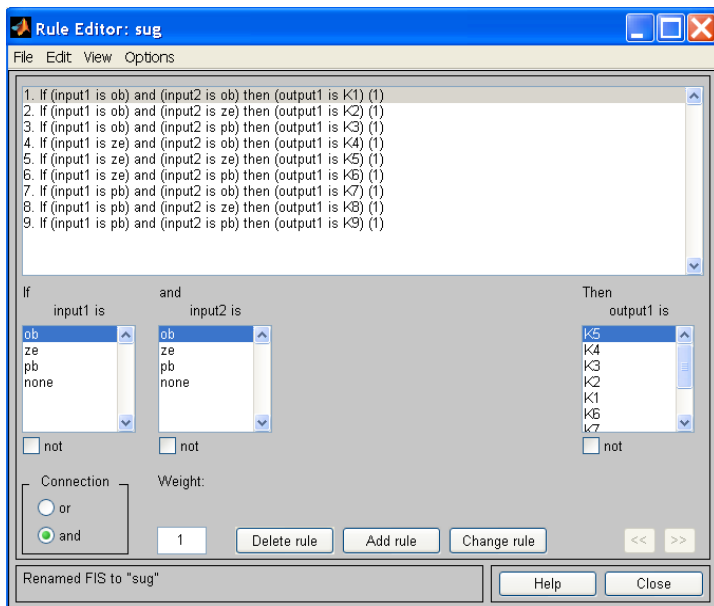


Рис. 5.23. Окно ввода правил для регулятора Сугэно

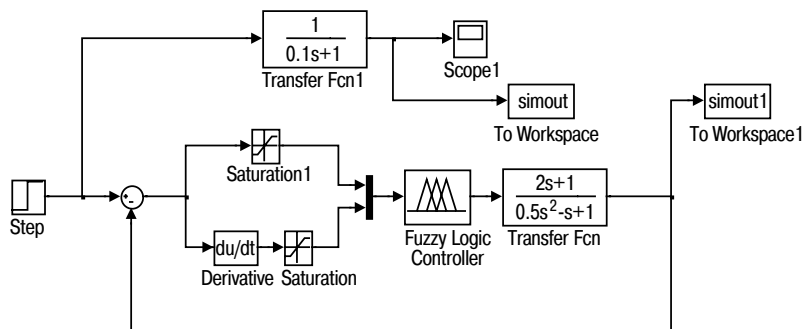


Рис. 5.24. Схема обучения НЛР Сугэно в Simulink MatLab

В данном случае настраиваемые переменные являются внутренними параметрами, входящими в структуру объекта Fuzzy logic controller. В приведенном ниже М-файле переменная sug – имя объекта Fuzzy logic controller, находящегося в рабочей памяти MatLab, а sug.output.mf(1,i).params – значение константы, входящей в правило с номером i:

```
function z=sug_reg(X);
global sug;
sug.output.mf(1,1).params=X(1);
sug.output.mf(1,2).params=X(2);
sug.output.mf(1,3).params=X(3);
sug.output.mf(1,4).params=X(4);
sug.output.mf(1,5).params=X(5);
sug.output.mf(1,6).params=X(6);
sug.output.mf(1,7).params=X(7);
sug.output.mf(1,8).params=X(8);
sug.output.mf(1,9).params=X(9);
sim('sugeno');
z=sum(abs(simout-simout1));
end
```

На рис. 5.25 показаны параметры запуска gatoool.

Графики рис. 5.26 дают возможность сравнить заданный и полученный переходный процесс. Для получения достаточно высокого качества настройки требуется всего несколько десятков поколений работы ГА.

Fitness function:	@sug_reg
Number of variables:	9

Рис. 5.25. Параметры запуска при обучении НЛР Сугэно

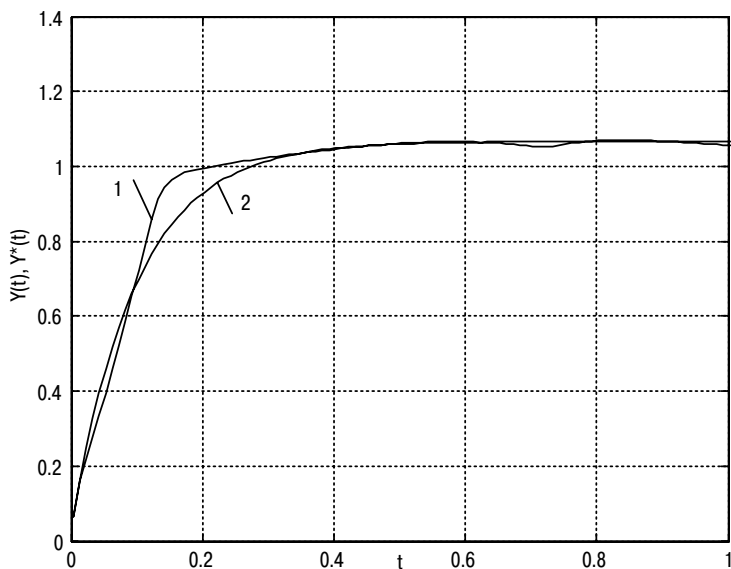


Рис. 5.26. *Результат генетической настройки НЛР Сугэно:
1 – полученный; 2 – заданный переходный процесс*

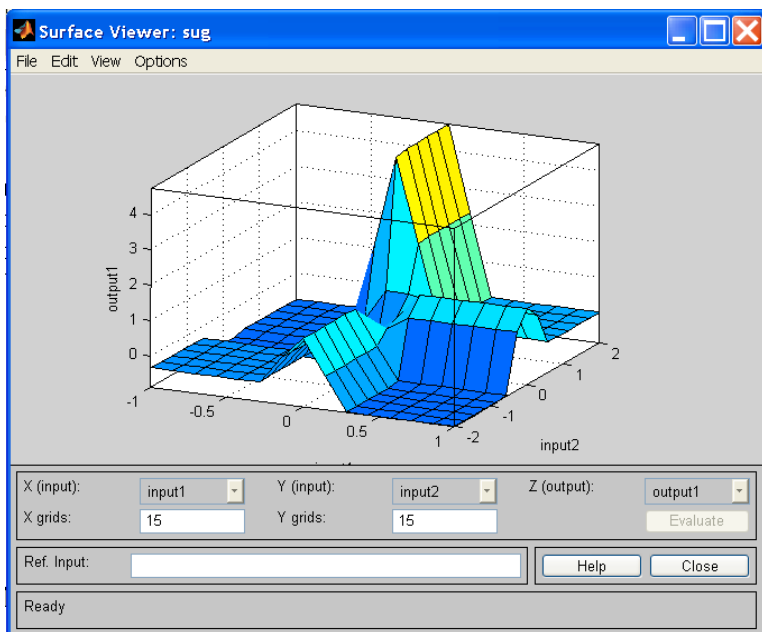


Рис. 5.27. *Поверхность отклика НЛР Сугэно*

Поверхность отклика синтезированного НЛР Сугэно (рис. 5.27) заметно отличается от поверхности отклика на рис. 5.16. Это является следствием общих принципов применения ГА:

- ищется не оптимальное, а субоптимальное (удовлетворительное) решение задачи;
- качество решения зависит от выбора параметров, по которым происходит генетическая настройка.

5.5. Настройка нейронной сети

Рассмотрим задачу настройки нейроэмулятора динамического объекта (см. п. 3.1.4). Пусть объект задан передаточной функцией

$$W(s) = \frac{1}{0,5s^2 + 0,01s + 1}.$$

Общая схема настройки была показана на рис. 3.21. Нейроэмулятор представляет собой двухслойную нейронную сеть прямого распространения. Первый слой содержит два нейрона, второй слой – один нейрон. Помимо входного сигнала, на нейроэмулятор поступает выходной сигнал, задержанный на один, два и три такта (рис. 5.28).

Выберем для нейронов линейные активационные функции. Нейроэмулятор будет иметь, таким образом, 10 весов ($W_1 + W_2$), подлежащих настройке. М-файл с описанием минимизируемой функции имеет следующий вид:

```
function z=set1(X);
global k1; global k2; global k3; global k4; global k5; global k6;
global k7; global k8; global k9; global k10;
k1=X(1); k2=X(2); k3=X(3); k4=X(4);
```

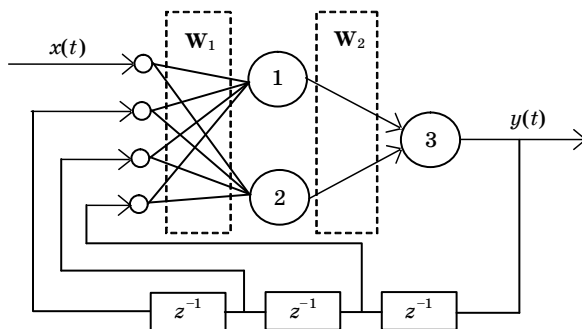


Рис. 5.28. Структура нейроэмулятора

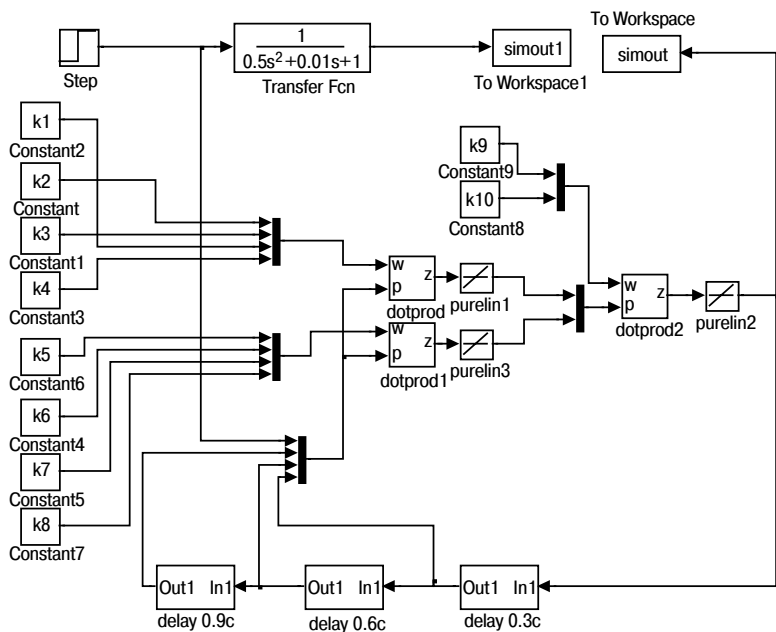


Рис. 5.29. Моделирование нейроэмулятора

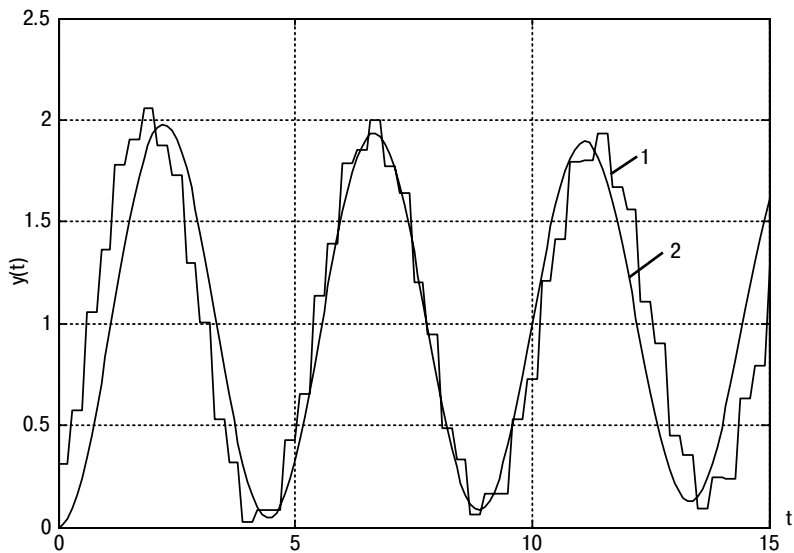


Рис. 5.30. Сравнение выходов нейроэмулятора и объекта:
1 – выход нейроэмулятора; 2 – выход объекта

```

k5=X(5); k6=X(6); k7=X(7); k8=X(8); k9=X(9); k10=X(10);
sim('setka');
z=sum(abs(simout-simout1));
end

```

Схема моделирования в Simulink MatLab показана на рис. 5.29.

Сравнение выхода объекта и его нейроэмулятора показано на рис. 5.30. Для рассмотренной постановки задачи этот результат является удовлетворительным.

5.6. Задачи для самостоятельной работы

5.6.1. Поиск экстремума функции одной переменной

1. Минимизировать заданный вариант функции (табл. 5.5).
2. Проверить полученный ответ визуально, построив график функции. Оценить количество локальных минимумов функции.
3. Оценить среднее количество шагов ГА, необходимых для достижения решения, выполняя запуск с разными начальными условиями.

Таблица 5.5. Функции одной переменной

№	Функция	Интервал
1	$f(x) = e^{0,1x} + \sin(x^2) - \cos(0,5x)$	$[-4, 4]$
2	$f(x) = x \sin(3x) - 5^{-x^2}$	$[-3, 9]$
3	$f(x) = x \sin(0,2x^2) + \frac{15}{x^3}$	$[1, 10]$
4	$f(x) = 40x \sin(x) - x^2 \cos(x)$	$[2, 20]$
5	$f(x) = tg(0,1x) - \sin(5x) \ln(0,1x)$	$[0, 10]$
6	$f(x) = x^3 - 34 \sin(2x)x^2 + 500 \cos(5x)$	$[0, 8]$
7	$f(x) = \sin(x^2) - \cos(x^3)$	$[0, 3]$
8	$f(x) = 3 \sin(x^3) - 2 \cos(x^2)$	$[-2, 3]$
9	$f(x) = 5 \sin(x^3) - 2 \cos(x)$	$[0, 3]$
10	$f(x) = \sin(2x) \cos(x^2) - 0,01x^3$	$[0, 6]$
11	$f(x) = x \cos(x) - (0,5x \cos(0,6x^2))$	$[5, 13]$

№	Функция	Интервал
12	$f(x) = 3\cos((0,5x)^2) + 0,01x^3$	$[0, 10]$
13	$f(x) = \cos(e^x)\cos(2x)$	$[1, 4]$
14	$f(x) = \sin(e^x)\sin(x)$	$[1, 4]$
15	$ x + \cos(x)$	$[-20, 20]$
16	$ x + \sin(x)$	$[-20, 20]$
17	$(x^2 + x)\cos(x)$	$[-10, 10]$

5.6.2. Поиск экстремума функции двух переменных

1. Минимизировать функцию двух переменных на заданном интервале (табл. 5.6).

2. Проверить полученный ответ визуально, построив график функции. Оценить количество локальных минимумов функции.

3. Оценить среднее количество шагов ГА, необходимых для достижения решения, выполняя запуск с разными начальными условиями.

Таблица 5.6. Функции двух переменных

№	Функция	Диапазон
1	$x \sin(4x) + 1,1y \sin(2y)$	$0 \leq x, y \leq 10$
2	$0,5 + \frac{\sin^2(\sqrt{x^2 + y^2}) - 0,5}{1 + 0,1(x^2 + y^2)}$	$-5 \leq x, y \leq 5$
3	$-e^{-0,2\sqrt{x^2 + y^2} + 3(\cos 2x + \sin 2y)}$	$-5 \leq x, y \leq 5$
4	$-x \sin(\sqrt{ x - (y + 9) }) - (y + 9) \sin(\sqrt{ y + 0,5x + 9 })$	$-20 \leq x, y \leq 20$
5	$z(x, y) = x^2 + y^2$	$-4 \leq x, y \leq 4$
6	$z(x, y) = \frac{x^2 + y^2}{\sin(x) + \cos(x)}$	$-4 \leq x, y \leq 4$
7	$z(x, y) = e^{-x^2 - y^2}$	$-2 \leq x, y \leq 2$
8	$z(x, y) = \sin(x)e^{-3y}$	$x \in [0, 2\pi], y \in [0, 1]$

№	Функция	Диапазон
9	$z(x, y) = \sin^2(x) \ln(y)$	$x \in [0, \pi], y \in [-1, 1]$
10	$z(x, y) = \sin^2(x - y) e^{- y }$	$x \in [0, \pi], y \in [-1, 1]$
11	$z(x, y) = \frac{\sin(xy)}{x}$	$x \in [0, 1.5], y \in [-\pi, \pi]$
12	$z(x, y) = \frac{x^2 y^2 + 2xy - 3}{x^2 + y^2 + 1}$	$x \in [-2, 2], y \in [-1, 1]$
13	$z(x, y) = (\sin(x^2) + \cos(y^2))^{xy}$	$x \in [-1, 1], y \in [-1, 1]$
14	$z(x, y) = (1 + xy)(3 - x)(4 - y)$	$x \in [0, 3], y \in [0, 4]$
15	$z(x, y) = (y^2 - 3) \sin \frac{x}{ y + 1}$	$x \in [-2\pi, 2\pi], y \in [-3, 3]$
16	$z(x, y) = e^{- x } (x^5 + y^4) \sin(xy)$	$x \in [-2, 2], y \in [-3, 3]$
17	$z(x, y) = \arctan(x + y)(\arccos(x) + \arcsin(y))$	$x \in [-1, 1], y \in [-1, 1]$
18	$z(x, y) = \sin^2(x) e^{ y }$	$x \in [0, \pi], y \in [-1, 1]$

5.6.3. Задачи линейного программирования

Решить задачу минимизации функции при линейных ограничениях (табл. 5.7).

Таблица 5.7. Задача минимизации при линейных ограничениях

№	Функция	Ограничения
1	$F = -2x_1 + x_3 - 3x_4 \rightarrow \min$	$\begin{cases} x_1 + 2x_2 - 3x_3 + 2x_4 \leq 8 \\ -x_1 - x_2 + 2x_3 - x_4 \leq 2 \\ 2x_2 - x_3 + 3x_4 = 4 \\ x_1 > 0 \\ x_2 > 0 \end{cases}$
2	$F = x_1 - 2x_2 \rightarrow \min$	$\begin{cases} x_1 - x_2 \leq 1 \\ x_1 + x_2 \geq 2 \\ x_1 - 2x_2 \leq 0 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{cases}$

Продолжение табл. 5.7

№	Функция	Ограничения
3	$F = 2x_1 - x_2 + 3x_3 + x_4 - 2x_5 \rightarrow \max$	$\begin{cases} x_1 - 2x_2 + x_3 + 3x_4 - 2x_5 \leq 4 \\ -x_1 - x_2 + 3x_3 - 2x_4 + 3x_5 \leq 10 \\ x_1 \geq 0 \\ x_2 \geq 0 \\ x_3 \geq 0 \\ x_4 \geq 0 \\ x_5 \geq 0 \end{cases}$
4	$F = 4x_1 + 2x_2 \rightarrow \max$	$\begin{cases} -x_1 + 3x_2 \leq 9 \\ 2x_1 + 3x_2 \leq 18 \\ 2x_1 - x_2 \leq 10 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{cases}$
5	$F = 2x_1 - x_2 + 3x_3 + 2x_4 - x_5 \rightarrow \max$	$\begin{cases} -x_1 + x_2 + x_3 = 1 \\ x_1 - x_2 + x_4 = 1 \\ x_1 + x_2 + x_5 = 2 \\ x_1 \geq 0 \\ x_2 \geq 0 \\ x_3 \geq 0 \\ x_4 \geq 0 \\ x_5 \geq 0 \end{cases}$
6	$F = 2x_1 + x_2 + 5x_4 \rightarrow \max$	$\begin{cases} -x_1 - 3x_2 + x_3 + 2x_4 = 5 \\ 2x_1 + x_2 - 2x_3 + x_4 = 2 \\ x_1 + 3x_3 - 3x_4 = 8 \\ x_1 \geq 0 \\ x_2 \geq 0 \\ x_3 \geq 0 \\ x_4 \geq 0 \end{cases}$

Продолжение табл. 5.7

№	Функция	Ограничения
7	$F = 2x_1 - x_2 + 3x_3 + x_4 - x_5 \rightarrow \max$	$\begin{cases} x_1 + x_2 - 3x_3 + 2x_4 - x_5 \leq 4 \\ x_2 - 3x_4 + 2x_5 \geq 2 \\ 2x_1 - 3x_2 - 2x_3 + x_4 - x_5 = 5 \\ x_1 + x_3 - 3x_4 + 2x_5 = 8 \\ x_1 \geq 0 \\ x_2 \geq 0 \\ x_3 \geq 0 \\ x_4 \geq 0 \\ x_5 \geq 0 \end{cases}$
8	$F = 8x_1 - 6x_2 - 5x_3 + 2x_4 \rightarrow \max$	$\begin{cases} x_1 + 4x_2 - x_3 + x_4 = 16 \\ 4x_1 - 6x_2 + 3x_3 - 7x_4 = 20 \\ x_1 \geq 0 \\ x_2 \geq 0 \\ x_3 \geq 0 \\ x_4 \geq 0 \end{cases}$
9	$F = 40x_1 + 36x_2 \rightarrow \min$	$\begin{cases} x_1 \leq 8 \\ x_2 \leq 10 \\ 5x_1 + 3x_2 \geq 45 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{cases}$
10	$F = x_1 + 2x_2 \rightarrow \min$	$\begin{cases} x_1 + 2x_2 \leq 10 \\ x_1 + x_2 \geq 1 \\ x_2 \leq 4 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{cases}$
11	$F = x_1 - 2x_2 + 3x_3 \rightarrow \max$	$\begin{cases} x_1 + x_2 + x_3 \leq 7 \\ x_1 - x_2 + x_3 \geq 2 \\ 3x_1 - x_2 - 2x_3 = -5 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{cases}$

№	Функция	Ограничения
12	$F = 3x_1 + 2x_2 \rightarrow \max$	$\begin{cases} -x_1 + 2x_2 \leq 4 \\ 3x_1 + 2x_2 \leq 14 \\ x_1 - x_2 \leq 3 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{cases}$
13	$F = -3x_1 + x_2 + x_3 \rightarrow \max$	$\begin{cases} x_1 - 2x_2 + x_3 \leq 11 \\ -4x_1 + x_2 + 2x_3 \geq 3 \\ 2x_1 - x_3 = -1 \\ x_1 \geq 0 \\ x_2 \geq 0 \\ x_3 \geq 0 \end{cases}$
14	$F = 10x_1 + 6x_2 + 4x_3 \rightarrow \max$	$\begin{cases} x_1 + x_2 + x_3 \leq 100 \\ 10x_1 + 4x_2 + 5x_3 \leq 600 \\ 2x_1 + 2x_2 + 6x_3 \leq 300 \\ x_1 \geq 0 \\ x_2 \geq 0 \\ x_3 \geq 0 \end{cases}$
15	$F = x_1 + 3x_2 \rightarrow \max$	$\begin{cases} x_1 \leq 5 \\ x_1 + 2x_2 \leq 10 \\ x_2 \leq 4 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{cases}$
16	$F = (x_1 - 3)^2 + (x_2 + 4)^2 + e^{5x_3} \rightarrow \min$	$\begin{cases} x_1 + x_2 + x_3 \leq 1 \\ x_1 \geq 0 \\ x_2 \geq 0 \\ x_3 \geq 0 \end{cases}$
17	$F = x_1^2 + x_2^2 + x_3^2 \rightarrow \min$	$\begin{cases} x_1 + 2x_2 + 3x_3 = 7 \\ 2x_1 + 2x_2 + x_3 = 4,5 \end{cases}$

№	Функция	Ограничения
18	$F = 4x_1 + 2x_2 + 5x_3 \rightarrow \max$	$\begin{cases} x_1 + 2x_2 + x_3 \leq 430 \\ 3x_1 + 2x_3 \leq 460 \\ x_1 + 4x_2 \leq 450 \\ x_1 \geq 0 \\ x_2 \geq 0 \\ x_3 \geq 0 \end{cases}$

5.6.4. Решение задачи нелинейного программирования

Решить задачу минимизации функции при нелинейных ограничениях (табл. 5.8).

Таблица 5.8. Задача минимизации при нелинейных ограничениях

№	Функция	Ограничения
1	$F = x_1^2 + 2x_2^2 \rightarrow \min$	$\begin{cases} x_1^2 + x_2^2 \leq 5 \\ 2x_1 - 2x_2 = 1 \end{cases}$
2	$F = 1 - 2x_1 - 4x_1x_2 \rightarrow \min$	$\begin{cases} x_1 + 4x_1x_2 \leq 4 \\ x_1 \geq 0 \\ x_2 \geq 0 \end{cases}$
3	$F = 3x_1^2 - 2x_2 \rightarrow \max$	$\begin{cases} 2x_1 + x_2 = 4 \\ x_1^2 + x_2^2 \leq 40 \\ x_1 \geq 0 \end{cases}$
4	$F = x_1^2 + x_2^2 \rightarrow \min$	$(x_1 - 1)^3 - x_2^2 = 0$
5	$F = x_1^2 + x_2^2 + x_3^2 \rightarrow \min$	$\begin{cases} 1 - \frac{x_3}{x_2} \geq 0 \\ x_1 - x_3 \geq 0 \\ x_1 - x_2^2 + x_2x_3 - 4 = 0 \\ 0 \leq x_1 \leq 5 \\ 0 \leq x_2 \leq 3 \\ 0 \leq x_3 \leq 3 \end{cases}$

Продолжение табл. 5.8

№	Функция	Ограничения
6	$F = x_1^2 + 4x_2^2 + x_3^2 \rightarrow \min$	$\begin{cases} x_1 + x_2^2 - 2 = 0 \\ -1 \leq x_1 \leq 1 \\ 0 \leq x_2 \leq 2 \\ 0 \leq x_3 \leq 2 \end{cases}$
7	$F = \frac{-x_1 x_2^2 x_3}{81} \rightarrow \min$	$\begin{cases} x_1^3 + x_2^2 + x_3 - 13 = 0 \\ x_2^2 \sqrt{x_3} - 1 = 0 \\ x_1 - x_2^2 + x_2 x_3 - 4 = 0 \\ 0 \leq x_1 \\ 0 \leq x_2 \\ 0 \leq x_3 \end{cases}$
8	$F = 3x_1^2 + 2x_2 \rightarrow \min$	$\begin{cases} 2x_1 + x_2 \geq 4 \\ x_1^2 + x_2^2 \leq 40 \end{cases}$
9	$F = (x_1 - 1)^2 + (x_1 - x_2)^2 + (x_2 - x_3)^2 \rightarrow \min$	$\begin{cases} x_1(1 + x_2^2) + x_2^4 - 4 - 3\sqrt{2} = 0 \\ 0 \leq x_1 \leq 3 \\ 0 \leq x_2 \leq 3 \\ 0 \leq x_3 \leq 3 \end{cases}$
10	$F = x_1^2 + x_2^2 \rightarrow \min$	$\begin{cases} 2 - x_1 - x_2^2 = 0 \\ x_1^2 - x_2 \geq 0 \\ 0,5 \leq x_2 \leq 2,5 \\ 0 \leq x_3 \leq 3 \end{cases}$
11	$F = x_1^4 + x_2 \rightarrow \min$	$\begin{cases} 9 - 2x_1^2 - 3x_2 \geq 0 \\ 0 \leq x_1 \\ 0 \leq x_2 \end{cases}$
12	$F = x_1 - x_2 \rightarrow \min$	$\begin{cases} 2x_1 - x_2^2 - 1 \geq 0 \\ 9 - 0,8x_1^2 - 2x_2 \geq 0 \\ 0 \leq x_1 \\ 0 \leq x_2 \end{cases}$

№	Функция	Ограничения
13	$F = 3x_1 + x_2 \rightarrow \min$	$1 - (x_1 - 2)^2 - (x_2 - 5)^2 \geq 0$
14	$F = (x_1 - 2)^2 + (x_2 - 2)^2 \rightarrow \min$	$\begin{cases} 3 - x_1 - x_2 \geq 0 \\ 10x_1 - x_2 - 2 \geq 0 \\ 0 \leq x_2 \end{cases}$
15	$F = 2x_1 - 3(x_2 - 4)^2 \rightarrow \max$	$\begin{cases} 10 - x_1^2 - x_2^2 \geq 0 \\ 9 - x_1^2 - (x_2 - 4)^2 \geq 0 \\ 0 \leq x_1 \end{cases}$
16	$F = x_1 + x_2^2 \rightarrow \max$	$\begin{cases} 25 - x_1^2 - x_2^2 \geq 0 \\ 9 - x_1^2 - x_2 \geq 0 \\ 0 \leq x_1 \leq 5 \\ 0 \leq x_2 \leq 10 \end{cases}$

5.6.5. Генетический синтез параметров регулятора

Для заданного передаточной функцией объекта управления (табл. 5.9) синтезировать регулятор. Работу выполнять в несколько этапов:

- выполнить анализ разомкнутой системы управления с целью выяснения устойчивости системы и параметров переходного процесса;
- попытаться синтезировать наиболее простой вариант регулятора (П-типа), в случае неудачи усложнить закон управления, используя регулятор ПИ-, ПД- или ПИД-типа;
- синтезировать НЛР со структурой, заданной преподавателем.

В качестве варианта можно сначала синтезировать нейроэмулятор, а затем синтезировать регулятор, используя нейроэмулятор в контуре настройки.

Таблица 5.9. Описание объекта управления

№	Передаточная функция разомкнутой системы	№	Передаточная функция разомкнутой системы
1	$W = \frac{2s+1}{5s^3+50s^2-3s+1}$	10	$W = \frac{s+1}{-s^3+0,1s^2+0,1s+1}$
2	$W = \frac{30s+1}{0,01s^4+0,1s^3-s^2-s+1}$	11	$W = \frac{10s^2+4s+1}{-3s^3+0,5s^2-0,5s+1}$

Окончание табл. 5.9

№	Передаточная функция разомкнутой системы	№	Передаточная функция разомкнутой системы
3	$W = \frac{10s^2 - 5s + 1}{0,1s^3 - 10s^2 + 2s + 1}$	12	$W = \frac{5s^2 + 2s + 1}{30s^3 - 5s^2 + 0,5s + 1}$
4	$W = \frac{100s^2 + 25s + 1}{10s^3 + 20s^2 - 20s + 1}$	13	$W = \frac{4s + 1}{-6s^3 + 10s^2 - 0,1s + 1}$
5	$W = \frac{s^2 + 10s + 1}{8s^3 - 4s^2 + 2s + 1}$	14	$W = \frac{-15s + 1}{20s^3 + 20s^2 - s + 1}$
6	$W = \frac{10s + 1}{-2s^3 - 2s^2 + s + 1}$	15	$W = \frac{5s^2 + 5s + 1}{4s^3 - 7s^2 + 3s + 1}$
7	$W = \frac{30s^2 - 25s + 1}{0,1s^3 + 0,01s^2 - 0,1s + 1}$	16	$W = \frac{10s^2 + 15s + 1}{10s^3 - 5s^2 - 2s + 1}$
8	$W = \frac{10s + 1}{-2s^3 + 2s^2 + s + 1}$	17	$W = \frac{2s + 1}{5s^2 - 3s + 1}$
9	$W = \frac{-2s + 1}{-5s^2 + 3s + 1}$	18	$W = \frac{s^2 - 15s + 1}{10s^3 + 5s^2 + 2s + 1}$

ЗАКЛЮЧЕНИЕ

Современная вычислительная техника позволяет ученым и инженерам-разработчикам моделировать разнообразные сложные объекты и процессы. Однако чем сложнее объект, тем труднее решить задачи оптимизации, идентификации и синтеза.

Генетический алгоритм может работать с любыми функциями: разрывными, не имеющими производных, целочисленными, в задачах с ограничениями и без ограничений, при любом размере пространства поиска. Такая универсальность привела к тому, что генетический алгоритм в настоящее время широко применяется в самых разных областях науки и техники. В учебном пособии описаны некоторые важные приложения генетического алгоритма, дающие возможность понять, как возникают типовые оптимизационные задачи в областях, на первый взгляд, далеких друг от друга.

Появление библиотеки GADS toolbox в составе Simulink MatLab открыло новый этап в распространении генетического алгоритма как методики решения задач оптимизации. В сочетании с другими широко известными пакетами, такими как Control System toolbox, Fuzzy Logic toolbox и Neural Net toolbox библиотека GADS дает новые возможности для синтеза регуляторов и моделирования сложных систем.

Приведенные в учебном пособии примеры позволяют получить опыт использования библиотеки GADS MatLab для решения практических задач. Конечно, успех использования генетического алгоритма в конкретной задаче не может быть гарантирован, он возникает при глубоком понимании разработчиком как механизмов действия генетического алгоритма, так и особенностей задачи, для решения которой генетический алгоритм привлекается.

Важное достоинство библиотек MatLab заключается в открытости их кода. Пользователь может модифицировать стандартные функции или использовать собственные функции, написанные на языке MatLab. Знание системы MatLab становится в последние годы одним из необходимых требований к выпускнику технического вуза. Для первоначального знакомства с MatLab можно воспользоваться пособием [74].

Успех генетического алгоритма стимулировал поиски других биологических аналогий, таких как «муравьиная оптимизация» (ants colony optimization), «ройный интеллект» (swarm intelligence) и т. д. Возникают направления, в которых эти подходы плодотворно используются совместно с нечеткой логикой, нейронными сетями и другими областями технического искусственного интеллекта.

Наблюдая за полетами птиц, человек создал турбореактивные лайнеры, весьма далекие от своего биологического прототипа, но умеющие летать. Одним из результатов размышлений о процессе эволюции стал генетический алгоритм. И уже не важно, что он является только грубой (а возможно – и неверной) моделью эволюции. Важно, что он решает практические задачи.

Библиографический список

1. *Fraser A. S.* Simulation of genetic systems // *J. of Theor. Biol.* 1962. Vol. 2. P. 329–346.
2. *Holland J.* Adaptation in Natural and Artificial Systems: An Introductory Analysis with application to Biology, Control and Artificial Intelligence. University of Michigan Press, 1975.
3. *Goldberg D. E.* Genetic Algorithms in Search, Optimization and Machine Learning. Reading, MA: Addison-Wesley, 1989.
4. Handbook of Genetic Algorithms / Ed. L. Davis. N.Y.: Van Nostrand Reinhold, 1991.
5. *Chambers L. D.* (Ed.) Practical Handbook of Genetic Algorithms. CRC Press, Boca Raton FL, 1995. Т. 1. 560 с.; Т. 2. 448 с.
6. *Fogel D. B.* Evolutionary computation: Toward a New Philosophy of Machine Intelligence // IEEE Press. Piscataway NJ, 1995.
7. *Растрюгин Л. А.* Адаптация сложных систем. Методы и приложения. Рига: Зинатне, 1981. 375 с.
8. *Букатова И. Л.* Эволюционное моделирование и его приложения. М.: Наука, 1979. 231 с.
9. *Букатова И. Л., Михасев Ю. И., Шаров А. М.* Эвоинформатика: Теория и практика эволюционного моделирования. М.: Наука, 1991. 206 с.
10. *Неймарк Ю. И.* Поисковые и оптимизационные возможности коллектива автоматов // Самоорганизация и адаптивные информационно-управляющие системы. М., 1979. С. 21–24.
11. *Батищев Д. И.* Генетические алгоритмы решения экстремальных задач. Воронеж, 1995. 65 с.
12. *Рутковская Д., Пилиньский М., Рутковский Л.* Нейронные сети, генетические алгоритмы и нечеткие системы. М.: Горячая линия – Телеком, 2004. 452 с.
13. *Емельянов В. В., Курейчик В. В., Курейчик В. М.* Теория и практика эволюционного моделирования. М.: Физматлит, 2003. 432 с.
14. *Гладков Л. А., Курейчик В. В., Курейчик В. М.* Генетические алгоритмы. М.: Физматлит, 2006. 320 с.
15. *Стецюра Г. Г.* Эволюционные методы в задачах управления, выбора и оптимизации // Приборы и системы управления. 1998. № 3. С. 54–62.
16. *Скурихин А. Н.* Генетические алгоритмы // Новости искусственного интеллекта. 1995. № 4. С. 6–46.
17. *Курейчик В. М.* Генетические алгоритмы. Состояние, проблемы, перспективы // Изв. РАН. Сер. Теория и системы управления. 1999. № 1. С. 144–160.

18. Бураков М. В., Коновалов А. С. Проектирование нейронных и нечетких регуляторов с помощью генетического алгоритма // Управление в условиях неопределенности / Под ред. А. Е. Городецкого; СПб.: Изд-во СПбГТУ, 2002. 399 с.
19. Genetic Algorithm and Direct Search Toolbox User's Guide // www.mathworks.com.
20. Поляк Б. Т. Введение в оптимизацию. М.: Наука, 1983.
21. Реклейтис Г., Рейвиндран А., Рэгсдел К. Оптимизация в технике: В 2 кн. М.: Мир, 1986.
22. Horst R., Pardalos P. M. (eds) Handbook of Global Optimization // Dordrecht: Kluwer, 1995.
23. Pint'r J. D. Global Optimization in Action. London: Kluwer, 1996.
24. Жигляевский А. А., Жулинская А. Г. Методы поиска глобального экстремума. М.: Наука, 1991.
25. Жулинская А., Шалтянис В. Поиск оптимума: компьютер расширяет возможности. М.: Наука, 1989. 128 с.
26. Орлянская И. В. Современные подходы к построению методов глобальной оптимизации // Электронный журнал «Исследовано в России» <http://zhurnal.apc.relarn.ru/articles/2002/189.pdf> 2101.
27. Horst R., Tuy H. Global Optimization, Deterministic Approaches. Berlin: Springer-Verlag, 1990.
28. Расстригин Л. А. Статистические методы поиска. М.: Наука, 1968. 376 с.
29. Price W. L. A Controlled Random Search Procedure for Global Optimization // The Computer Journal. 1977. Vol. 20. P. 367–370.
30. Rinnoy Kan A. H. G., Timmer G. T. Stochastic Global Optimization Methods // Mathematical programming. 1987. N 39. P. 27–78.
31. Törn A. Global Optimization as a Combination of Global and Local Search. Turku: Abo Akademi University, ННÅА А:13, 1974.
32. Aarts E. H. L., van Laarhoven P. J. M. Simulated Annealing: Theory and Applications. London: Kluwer, 1987.
33. Ali M., Storey C. Aspiration Based Simulated Annealing Algorithm // J. of Global Optimization. 1996. N 11. P. 181–191.
34. Darwin Ch. The Origin of Species. London: John Murray, 1859.
35. Darwin Ch. The Descent of Man and Selection in Relation to Sex. John Murray. London, 1871.
36. Чайковский Ю. В. Эволюция / ИИЕТ РАН. М., 2003. Вып. 22. 472 с.
37. Бураков М. В. Синтез нейронного регулятора // Изв. РАН. Сер. Теория и системы управления. 1999. № 3. С. 140–145.
38. Бураков М. В. Структура нейронечеткого регулятора // Изв. РАН. Сер. Теория и системы управления. 2001. № 6. С. 160–165.

39. Бураков М. В., Попов О. С. Совместное использование имитационного моделирования и экспертных процедур для управления динамическими объектами // Изв. вузов. Сер. Приборостроение. 1994. № 5–6. С. 11–13.

40. Бураков М. В., Попов О. С. Элементы искусственного интеллекта в проблеме управления сложным динамическим объектом // Автоматика и телемеханика. 1997. № 8. С. 118–124.

41. *Astrom, K. J. and Hagglund T.* Advanced PID control // ISA – The Instrumentation, System and Automation Society, 2006. 460 p.

42. Бураков М. В. Микроконтроллерные системы управления: Метод. указ. для курсового и дипломного проектирования / ГУАП. СПб., 2003. 54 с.

43. *Ziegler J. G., Nichols N. B.* Optimum settings for automatic controllers // Trans. ASME. 1942. Vol. 64. P. 759–768.

44. *Zadeh L. A.* Fuzzy sets // Information and Control. 1965. Vol. 8. P. 338–353.

45. *Mamdani E. H.* Application of fuzzy algorithms for control of simple dynamic plant // IEEE Proc. 1974. Vol. 121. N 12. P. 1585–1588.

46. Бураков М. В., Коновалов А. С., Шумилов П. Е. Интеллектуальные системы авиационной антиюзовой автоматики. СПб.: Изд-во политехнического университета, 2005. 242 с.

47. Заде Л. Понятие лингвистической переменной и его применение для принятия приближенных решений. М.: Мир, 1976. 165 с.

48. *Sinn-Cheng Lin, Yung-Yaw Chen.* Design of self-learning fuzzy sliding mode controllers based on genetic algorithms // Fuzzy sets and systems. 1997. Vol. 86. P. 139–153.

49. *Herrera F., Losano M., Verdegay J. L.* A learning process for fuzzy control rules using genetic algorithms // Fuzzy sets and systems. 1998. Vol. 100. P. 143–158.

50. *Gurocak H. B.* A genetic-algorithm-based method for tuning fuzzy logic controllers // Fuzzy sets and systems. 1999. Vol. 108. P. 39–47.

51. *Warwick K., Irwin G. W., Hint K. J.* Neural Networks for Control and Systems. London: Peter Peregrinus, 1988.

52. *Miller W. T., Sutton R. S., Werbos P. J.* Neural Networks for Control. Cambridge, MA: MIT Press, 1990.

53. Уоссермен Ф. Нейрокомпьютерная техника. М.: Мир, 1992.

54. *Hecht-Nielsen R.* Neurocomputing: Reading. MA: Addison – Wesley, 1989.

55. *Rumelhart D. E., Hinton G. E., Williams R. J.* Learning internal representations by error propagation: Parallel distributed processing. Cambridge, MA: MIT Press, 1986. Vol. 1. P. 318–362.

56. Терехов В. А., Ефимов Д. В., Тюкин И. Ю., Антонов В. Н. Нейросетевые системы управления. СПб.: Изд-во СПбГУ, 1999. 265 с.

57. Тарасов В. Б. От многоагентных систем к интеллектуальным организациям. М., 2002. 352 с.
58. Holland J. Adaptation in Natural and Artificial Systems: An Introductory Analysis with application to Biology, Control and Artificial Intelligence. University of Michigan Press, 1975.
59. Koza J. R. Genetic Programming. Cambridge, MA: MIT Press, 1992.
60. Michalewicz Z. Genetic Algorithms + Data Structures = Evolution Programs. Springer Verlag, 1994.
61. Хювенен Э., Сеппянен Й. Мир Лиспа: Введение в язык Лисп и функциональное программирование. М.: Мир, 1990. 447 с.
62. Майерс Г. Надежность программного обеспечения. М.: Мир, 1980.
63. Лунаев В. В. Качество программного обеспечения. М.: Финансы и статистика, 1983. 261 с.
64. Холстед М. Х. Начала науки о программах. М.: Финансы и статистика, 1981. 128 с.
65. Бейзер Б. Тестирование черного ящика. Технология функционального тестирования программного обеспечения и систем. СПб.: Питер, 2004. 318 с.
66. White L. J. and Cohen E. I. A domain strategy for computer program testing // IEEE Transactions on Software Engineering. May 1980. Vol. SE-6. N 3. P. 247–257.
67. Шалыто А. А., Туккель Н. И. Программирование с явным выделением состояний // Мир ПК. 2001. № 8, 9.
68. Казаков М. А., Корнеев Г. А., Шалыто А. А. Разработка логики визуализаторов алгоритмов на основе конечных автоматов // Телекоммуникации и информатизация образования. 2003. № 6.
69. Бураков М. В., Коновалов А. С., Подзорова А. В. Модель надежности программного обеспечения // Современные проблемы социально-экономического развития и информационных технологий: Сб. докл. Междунар. НТК. Баку, 2004. С. 51–54.
70. Genetic Algorithm and Direct Search Toolbox User's Guide. www.mathworks.com.
71. Общий курс высшей математики для программистов: Учебник / Под ред. В. И. Ермакова. М.: ИНФРА-М, 2001. 656 с.
72. Дьяконов В., Круглов В. Математические пакеты расширения Matlab. Специальный справочник. СПб.: Питер, 2001. 480 с.
73. Lee C. C. Fuzzy logic in control systems: fuzzy logic controller // IEEE Transaction on System, Man and Cybernetics. 1990. N 20(2). P. 404–435.
74. Бураков М. В. Основы работы в MatLab: Учеб. пособие / ГУАП. СПб., 2006. 67 с.

Учебное издание

Бураков Михаил Владимирович

**ГЕНЕТИЧЕСКИЙ АЛГОРИТМ:
ТЕОРИЯ И ПРАКТИКА**

Учебное пособие

Редактор *А. Г. Ларионова*
Верстальщик *С. Б. Мацапура*

Сдано в набор 28.12.07. Подписано к печати 27.02.08.
Формат 60×84 1/16. Бумага офсетная. Печать офсетная.
Усл.-печ. л. 10,3. Уч.-изд. л. 10,0. Тираж 150 экз. Заказ №

Редакционно-издательский центр ГУАП
190000, Санкт-Петербург, Б. Морская ул., 67