МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
(МИНОБРНАУКИ)

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»
(СПБГУ ИТМО)

УДК 004.4’242
№ госрегистрации 0120.0 710295
Инв. №

УТВЕРЖДАЮ
Ректор СПбГУ ИТМО,
докт. техн. наук, профессор
В. Н. Васильев

«____» _____ 2007 г.

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

ТЕХНОЛОГИЯ ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ ДЛЯ ГЕНЕРАЦИИ АВТОМАТОВ УПРАВЛЕНИЯ СИСТЕМАМИ СО СЛОЖНЫМ ПОВЕДЕНИЕМ

ПРОМЕЖУТОЧНЫЙ ОТЧЕТ ПО II ЭТАПУ «ТЕОРЕТИЧЕСКИЕ ИССЛЕДОВАНИЯ ПОСТАВЛЕННЫХ ПЕРЕД НИР ЗАДАЧ»

ЛИСТОВ 100

Декан факультета «Информационные технологии и программирование»
докт. техн. наук, профессор
_____ В. Г. Парфенов

Руководитель темы
заведующий кафедрой «Технологии программирования»,
докт. техн. наук, профессор
_____ А. А. Шалыто

Ответственный исполнитель
доцент кафедры «Компьютерные технологии»,
канд. техн. наук
_____ Г. А. Корнеев

Санкт-Петербург
2007

СПИСОК ИСПОЛНИТЕЛЕЙ

Руководитель темы докт. техн. наук, профессор	А. А. Шалыто	Отчет в целом
Ответственный исполнитель канд. техн. наук	Г. А. Корнеев	Отчет в целом
Ведущий научный сотрудник, докт. техн. наук, профессор	В. В. Антипов	Разделы 1.1. и 1.2.
Ведущий научный сотрудник, канд. техн. наук	В. В. Киселев	Разделы 2.1. и 2.3.
Ведущий научный сотрудник, канд. техн. наук	Р. Н. Котляр	Разделы 2.2. и 2.3.
Ведущий научный сотрудник, канд. техн. наук	Ю. П. Московцев	Разделы 2.1. и 2.4.
Ведущий научный сотрудник, канд. техн. наук, доцент	В. А. Третьяков	Разделы 1.3. и 1.4.
Ведущий научный сотрудник, канд. техн. наук	Г. М. Файкин	Разделы 1.1. и 1.4.
Канд. техн. наук	Е. М. Кузнецова	Разделы 2.1. и 2.2.
Канд. техн. наук	А. С. Пресняк	Разделы 2.3. и 2.4.
Канд. техн. наук, доцент	Э. А. Опалева	Разделы 3.2. и 3.3.
Руководитель временного трудового коллектива (ВТК), ассистент кафедры КТ	А. П. Мельничук	Раздел 2.1.
Член ВТК, профессор кафедры ИС	Е. Ю. Михайлова	Раздел 2.1.1.
Член ВТК, доцент кафедры КТ	В. Д. Наумчик	Раздел 2.1.1.
Член ВТК, доцент кафедры КТ	М. Ю. Осипов	Раздел 2.1.4.
Член ВТК, доцент кафедры КТ	А. Н. Воробьев	Раздел 2.1.2.
Член ВТК, доцент кафедры КТ	Ю. А. Щупак	Раздел 2.1.4.
Член ВТК, доцент кафедры КТ	С. В. Чириков	Раздел 2.1.2.
Член ВТК, доцент кафедры КТ	А. С. Сегаль	Раздел 2.1.1.
Член ВТК, доцент кафедры КТ	Д. Г. Шопырин	Раздел 2.1.4.
Член ВТК, доцент кафедры ИС	Д. А. Зубок	Раздел 2.1.2.
Член ВТК, ассистент кафедры ИС	В. В. Повышев	Раздел 2.1.3.
Член ВТК, ассистент кафедры ИС	В. В. Ильин	Раздел 2.1.1.
Член ВТК, ассистент кафедры ИС	М. Г. Холин	Раздел 2.1.3.
Аспирант	П. Г. Лобанов	Разделы 1.3, 2.1.3, 2.2.3, 2.3.3 и 3.3
Аспирант	К. В. Рубинов	Разделы 1.2 и 1.4.
Магистрант	Н. И. Поликарпова	Разделы 1.1, 2.1.1, 2.2.1, 2.3.1 и 3.1.
Магистрант	В. Н. Точилин	Разделы 1.1., 2.1.1., 2.2.1, 2.3.1 и 3.1.
Магистрант	Ф. Н. Царев	Разделы 3.1 и 3.2.
Магистрант	Ю. Д. Бедный	Разделы 1.4, 2.1.4, 2.2.4, 2.3.4 и 3.4.
Магистрант	В. Р. Данилов	Разделы 1.2, 2.1.2, 2.2.2, 2.3.2 и 3.2.
Студент	Е. А. Мандриков	Разделы 3.3 и 3.4.
Студент	В. А. Кулев	Разделы 3.3 и 3.4.

РЕФЕРАТ

Отчет 100 с., 3 гл., 85 рис., 12 табл., 55 источников.

ТЕОРЕТИЧЕСКИЕ ИССЛЕДОВАНИЯ ПОСТАВЛЕННЫХ ПЕРЕД НИР ЗАДАЧ

Объектом исследования являются методы генерации автоматов управления системами со сложным поведением.

Цель этапа — разработка эффективных методов генерации автоматов управления системами со сложным поведением.

В результате выполнения второго этапа работы был предложен набор методов генетического программирования, предназначенных для построения автоматов управления системами со сложным поведением. Для каждого метода приведены их функциональные особенности и характеристики. На основе предложенных методов разработана технология генетического программирования для генерации автоматов управления системами со сложным поведением. Описываются созданные прототипы инструментальных средств генетического программирования, осуществляющих построение автоматов управления системами со сложным поведением.

В первой главе отчета описаны разработанные методы генетического программирования, предназначенные для построения автоматов управления системами со сложным поведением.

Во второй главе исследованы области применимости и функциональные особенности предложенных методов.

В третьей главе разработаны прототипы программных средств, реализующих предложенные методы, проведены их экспериментальные исследования.

Таким образом, были получены решения всех задач, поставленных в техническом задании на проведение второго этапа работы.

Экономическую значимость второго этапа работы до завершения работ в целом оценить затруднительно.

ОГЛАВЛЕНИЕ

СПИСОК ИСПОЛНИТЕЛЕЙ 2
РЕФЕРАТ 3
ОГЛАВЛЕНИЕ 4
ВВЕДЕНИЕ 6
1. МЕТОДЫ АВТОМАТИЧЕСКОЙ ГЕНЕРАЦИИ АВТОМАТОВ УПРАВЛЕНИЯ СИСТЕМАМИ СО СЛОЖНЫМ ПОВЕДЕНИЕМ 8
1.1. Метод сокращенных таблиц переходов 9
1.1.1. Постановка задачи 9
1.1.2. Представление состояний: полные таблицы 10
1.1.3. Представление состояний: сокращенные таблицы 13
1.1.4. Мутация, зависящая от пригодности 17
1.1.5. Формирование нового поколения 20
1.1.5.1. Выбор особей для скрещивания 20
1.1.5.2. Модификация оценочной функции 20
1.2. Метод представления автоматов деревьями решений 20
1.2.1. Представление автоматов деревьями решений 21
1.2.2. Генетические операции 23
1.2.3. Задание ограничений на целевой автомат 25
1.2.4. Возможные модификации метода 26
1.2.5. Простой пример применения 26
1.3. Методы оптимизации генетических алгоритмов для построения автоматов 30
1.3.1. Использование автоматов с флагами 31
1.3.2. Алгоритм восстановления связей между состояниями 35
1.3.3. Алгоритм сортировки состояний в порядке использования 39
1.4. Композитное генетическое программирование 41
1.4.1. Идея метода композитного генетического программирования 43
1.4.1.1. Задача классификации плотности 43
1.4.1.2. Постановка проблемы, возникающей при решении задачи классификации плотности с помощью эволюционных алгоритмов 45
1.4.2. Описание метода 45
1.5. Выводы 47
2. ФУНКЦИОНАЛЬНЫЕ ОСОБЕННОСТИ И ХАРАКТЕРИСТИКИ МЕТОДОВ ГЕНЕРАЦИИ АВТОМАТОВ 48
2.1. Функциональные особенности методов генерации автоматов 48
2.1.1. Метод сокращенных таблиц переходов 48
2.1.2. Метод представления автоматов деревьями решений 50
2.1.3. Методы оптимизации генетических алгоритмов для построения автоматов 52
2.1.4. Метод композитного генетического программирования 53
2.2. Характеристики методов генерации автоматов 54
2.2.1. Метод сокращенных таблиц переходов 54
2.2.2. Метод представления автоматов деревьями решений 58
2.2.3. Методы оптимизации генетических алгоритмов для построения автоматов 58
2.2.4. Метод композитного генетического программирования 58
2.3. Условия и ограничения функционирования методов генерации автоматов 59
2.3.1. Метод сокращенных таблиц переходов 59

2.3.2. Метод представления автоматов деревьями решений 61
2.3.3. Методы оптимизации генетических алгоритмов для построения автоматов 63
2.3.4. Метод композитного генетического программирования 63
2.4. Выводы 63
3. ТЕХНОЛОГИЯ ГЕНЕТИЧЕСКОГО ПРОРАММИРОВАНИЯ ДЛЯ ГЕНЕРАЦИИ АВТОМАТОВ УПРАВЛЕНИЯ СИСТЕМАМИ СО СЛОЖНЫМ ПОВЕДЕНИЕМ 64
3.1. Программное средство *AutoGen* 64
3.1.1. Модель программного средства 64
3.1.2. Прототип программного средства 67
3.2. Программное средство, основанное на представление автоматов деревьями решений 70
3.2.1. Модель программного средства 70
3.2.2. Прототип программного средства 72
3.2.3. Результаты экспериментов 74
3.3. Программное средство для оценки эффективности методов оптимизации генетических алгоритмов для построения автоматов 75
3.3.1. Модель программного средства 75
3.3.2. Прототип программного средства 78
3.3.2.1. Эксперименты по использованию автоматов с флагами 78
3.3.2.2. Эксперименты по применению алгоритма восстановления связей между состояниями 81
3.3.2.3. Эксперименты по применению алгоритма сортировки состояний в порядке использования 84
3.4. Программное средство для поддержки композитного генетического программирования 85
3.4.1. Модель программного средства 86
3.4.2. Прототип программного средства 86
3.4.3. Результаты экспериментов 89
3.4.3.1. Постановка задачи 89
3.4.3.2. Методы решения задачи эволюционными алгоритмами 90
3.4.3.3. Представление автоматов в виде графа переходов 91
3.4.3.4. Представление автоматов в виде *Karva*-деревьев 91
3.4.4. Универсальная функция приспособленности и универсальная форма решений 93
3.5. Методические рекомендации и указания по разработке автоматов управления систем со сложным поведением 94
ЗАКЛЮЧЕНИЕ 97
ИСТОЧНИКИ 98

ВВЕДЕНИЕ

Технология генетического программирования для генерации автоматов управления системами со сложным поведением разрабатывается в рамках проведения научно-исследовательской работы по *лоту «Разработки в области языков программирования и моделирования программного обеспечения, технологий и инструментальных средств проектирования программ» шифр «2007-4-1.4-18-01-033» по теме «Технология генетического программирования для генерации автоматов управления системами со сложным поведением»*, выполняемой в рамках Федеральной целевой научно-технической программы «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007−2002 годы» по *государственному контракту № 02.514.11.4044 от 18.05.2007*, заключенному между Федеральным агентством по науке и инновациям и Государственным образовательным учреждением высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» на основании решения Конкурсной комиссии Роснауки № 14 (протокол от 28.04.2007 г. № 14).

Целями настоящего этапа работы является разработка эффективных методов генерации автоматов управления системами со сложным поведением. Задачами этапа являются:

1. Разработка новых методов генерации автоматов.
2. Определение функциональных особенностей и характеристик разработанных методов.
3. Разработка прототипов программных средств генерации автоматов, основанных на предложенных методах.

Достижение цели настоящего этапа работы позволит перейти к экспериментальным исследования разработанных методов.

В настоящее время работы по построению автоматов на основе генетического программирования проводятся в ряде зарубежных университетов, в том числе в Массачусетском технологическом институте и Университете Южной Калифорнии. Знакомство с результатами этих работ показало, что в них строятся автоматы, которые не могут быть использованы в системах со сложным поведением.

Обычно генетические алгоритмы в рамках эволюционного моделирования используются для настройки нейронных сетей. Однако, если настроенная нейронная сеть функционирует в рассматриваемой среде недостаточно эффективно, то вручную ее перенастроить невозможно. Поэтому приходится вновь и вновь настраивать ее при помощи генетических алгоритмов. При этом человеку весьма трудно направить процесс в нужном направлении, а также при необходимости понять полученный результат.

При применении генетических алгоритмов для настройки автоматов ситуация принципиально изменяется, так как построив с помощью генетических алгоритмов автоматы, их в дальнейшем обычно удается модифицировать вручную.

Поиск приемлемого по выбранным критериям управляющего автомата перебором практически невозможен из-за огромного размера пространства, в котором осуществляется поиск. Например, в такой простой задаче как задача об «Умном муравье», число возможных автоматов с семью состояниями около $3{,}2\times10^{18}$.

Применение генетического программирования позволяет сделать перебор направленным, однако и в этом случае трудоемкость построения автоматов с требуемыми свойствами остается большой.

Указанная проблема решается за счет учета специфики автоматов, применяемых в системах управления, состоящей в том, что состояния декомпозируют входные воздействия на группы, в каждую из которых обычно входит небольшое число переменных. Это позволяет строить хромосомы

только для подмножества всех автоматов, что существенно сокращает пространство возможных решений, и как следствие, время поиска. Эти идеи развиты в настоящем отчете.

Метрологическое обеспечение НИР не требуется.

Дополнительные патентные исследования, проведенные в рамках второго этапа работы (отчет о патентных исследованиях № 2007.12.29-1 входит в состав отчетной документации по этапу), позволяют утверждать, что в настоящее время отсутствуют патенты и иные охранные документы, которые могут препятствовать применению в Российской Федерации результатов научных исследований, проводимых по контракту.

Изложенное позволяет утверждать, что результаты выполнения научно-исследовательской работы будут соответствовать мировому уровню разработок в рассматриваемой области.

1. МЕТОДЫ АВТОМАТИЧЕСКОЙ ГЕНЕРАЦИИ АВТОМАТОВ УПРАВЛЕНИЯ СИСТЕМАМИ СО СЛОЖНЫМ ПОВЕДЕНИЕМ

В данной главе описываются разработанные методы генерации автоматов управления системами со сложным поведением.

Из литературы известны методы построения конечных автоматов с помощью генетических алгоритмов, генетического программирования и других эволюционных подходов.

Большинство работ в этой области посвящено построению автоматов-*распознавателей*, описывающих грамматику некоторого языка. Задача такого автомата состоит в определении принадлежности заданной строки языку. Распознаватель не производит выходных воздействий, а результат определяется состоянием автомата после обработки входной последовательности. В данном направлении как наиболее значимые можно выделить работы [4, 18].

Более сложная форма конечного автомата – *преобразователь* – отображает множество входных строк на множество выходных, возможно, над другим алфавитом. Примером преобразователя является компилятор, а примером распознавателя – синтаксический анализатор, определяющий соответствие кода программы грамматике языка программирования. Эволюционному построению преобразователей посвящена работа [31].

В распознающих и преобразующих автоматах все условия переходов имеют вид сравнения входного символа с заданным. Таким образом, подмножество входных воздействий, удовлетворяющее условию конкретного перехода, всегда состоит из единственного входного символа.

В области генетического программирования традиционно используются модели вычислений в виде графов, которые можно интерпретировать как графы переходов конечных автоматов. Построению программ в виде графов посвящено большое число работ, например [41, 42]. Применение автоматов в играх описано в работах [3, 33].

Небольшое число работ затрагивают вопросы построения *управляющих автоматов* (автоматов, описывающих логику сложного поведения сущности или системы). Например, в статьях [17, 37] рассматривается автоматическое построение компонент логических контроллеров в виде автоматов, а в работе [36] оценивается эффективность автоматных моделей применительно к различным задачам.

Практически во всех рассмотренных исследованиях автомат в каждый момент времени обрабатывает только одну входную переменную. Исключения составляют лишь работы [40] (четыре параллельных троичных входа) и [36] (в качестве условия перехода допускается сравнение значений двух регистров). Теоретически, любое число параллельных входов сводится к одному, в качестве воздействий которого выступают комбинации сигналов исходных параллельных входов. Однако размер алфавита, полученного таким образом входа, растет экспоненциально с увеличением числа исходных параллельных входов. В обоих упомянутых работах параллельные входы не приводят к недопустимо большому алфавиту, но для реальных систем эта проблема крайне актуальна.

Также в рассмотренных работах автомат на каждом шаге может выполнить не более одного действия из заданного множества. В таком случае любая комбинация действий, которые может потребоваться выполнить одновременно, также должна считаться элементарным действием. При этом требуется априорная информация обо всех возможных комбинациях действий, либо задание вместе с элементарными действиями всех их наборов, что приводит к экспоненциальному росту числа действий. Отметим, что в работе [36] допускаются действия с аргументами, а также параллельно выполняемые автоматы, отвечающие за различные действия, что значительно ослабляет проблему.

В настоящей работе рассматривается задача построения управляющих автоматов. Из всех перечисленных работ наилучшие результаты в этом аспекте получены в статьях [37, 38]

применительно к созданию управляющей программы робота. Однако и эти работы не лишены упомянутых выше недостатков, относящихся к входным и выходным воздействиям. Насколько известно авторам, исследования в области эволюционного построения логики систем со сложным поведением, отличных от роботов, ранее не проводились.

Для генерации автоматов применяются различные модификации генетических алгоритмов [49]. Однако, для большинства из этих методов размер хромосомы, требуемый для хранения автомата, экспоненциально зависит от количества входных переменных. В следствие этого, размер пространства поиска при описании автоматов виде битовых строк растет как повторная экспонента количества состояний, что не позволяет использовать такое представление на практике. Таким образом, необходимо разработать новые методы представления автоматов (разд. 1.1, 1.2) и оптимизировать методы генерации за счет учета специфики автоматных моделей (разд. 1.3).

Вторым следствием огромного числа возможных автоматов является необходимость комбинирования методов, эффективных на различных этапах решения задачи. В частности, некоторые методы позволяют довольно быстро получать хороших особей, но не могут довести их до оптимума, в то время, как другие методы позволяют получать оптимальные решение, но эволюция в них происходит очень медленно. Вопросы комбинации таких методов рассмотрены в разд. 1.4.

1.1. Метод сокращенных таблиц переходов

При разработке метода сокращенных таблиц авторами наибольшее внимание уделяли специфике управляющих автоматов — возможности построения автоматов с произвольным числом параллельных входов и выходов. Кроме того, для сокращения пространства поиска метод использует концепцию автоматизированного объекта управления [54]: логика сложного поведения описывается автоматом на высоком уровне абстракции, а объект управления задается в качестве входного данного и оптимизации не подвергается. Объект управления может быть произвольным (и, вообще говоря, сколь угодно сложным). Таким образом, предлагаемый метод решает задачу об использовании сложных структур данных в рамках генетического программирования, поставленную основателем генетического программирования в работе [28].

В разд. 1.1.2 рассматривается упрощенная (наивная) версия метода, соответствующая по своим характеристикам большинству аналогов. Далее показаны недостатки наивного подхода, и разд. 1.1.3 приведено описание усовершенствованной версии метода, лишенной этих недостатков.

1.1.1. Постановка задачи

Сформулируем *задачу построения управляющего автомата*. Пусть задан объект управления $O = \langle V, v_0, X, Z \rangle$, где V – множество вычислительных состояний (или *значений*), v_0 – *начальное значение*, $X = \{x_i : V \to \{0,1\}\}_{i=1}^{n}$ – множество предикатов, $Z = \{z_i : V \to V\}_{i=1}^{m}$ – множество действий. Также задана оценочная функция $\varphi : V \to \mathbf{R}^+$ и натуральное число k.

Объект O может управляться автоматом вида $A = \langle S, s_0, \Delta \rangle$, где S – конечное множество управляющих состояний, s_0 – *стартовое состояние*, $\Delta : S \times \{0,1\}^n \to S \times Z^*$ – *управляющая функция*. Управляющую функцию можно разложить на две компоненты: функцию выходов $\zeta : S \times \{0,1\}^n \to Z^*$ и функцию переходов $\delta : S \times \{0,1\}^n \to S$.

Пусть перед началом работы объекту управления соответствует начальное значение v_0, а автомат находится в стартовом состоянии s_0. Назовем *шагом работы* автоматизированного объекта следующую последовательность операций.

1. Объект управления вызывает все предикаты из множества X и формирует из их значений вектор *входного воздействия* $in \in \{0,1\}^n$.
2. Автомат вычисляет значение вектора *выходного воздействия* $out = \zeta(s, in)$, где s – текущее состояние автомата, и переходит в новое управляющее состояние $s_{new} = \delta(s, in)$.
3. Объект управления по очереди вызывает действия $z \in out$, тем самым, изменяя текущее вычислительное состояние.

Задача построения управляющего автомата состоит в том, чтобы найти автомат заданного вида такой, что за k шагов работы под управлением этого автомата объект O перейдет в вычислительное состояние с максимальной пригодностью ($\varphi(v) \to \max$).

В связи с использованием генетического программирования для этой задачи возникают следующие **подзадачи**: выбор представления конечного автомата в виде особи; адаптация генетических операторов (мутации и скрещивания) для выбранного представления; настройка параметров генетической оптимизации.

В классической интерпретации генетического алгоритма особь представляется в виде набора хромосом. Управляющий автомат легко представить как набор состояний, в каждом из которых его поведение определяется сужением управляющей функции $\Delta_s : \{0,1\}^n \to S \times Z^*$, $s \in S$. Таким образом, удобно сопоставить каждому состоянию хромосому. Таким образом, от задачи представления автомата в виде особи перейдем к более конкретной постановке проблемы: представлению управляющего состояния автомата в виде хромосомы.

1.1.2. Представление состояний: полные таблицы

Естественный способ записи хромосомы состояния – это табличное представление функции Δ_s. Таблица содержит 2^n строк (по одной для каждой возможной комбинации значений n предикатов) и $m+1$ столбцов, в первом из которых записано значение функции переходов (номер целевого состояния), а совокупность остальных столбцов обозначает множество действий m, которые необходимо выполнить на переходе. Пример полной таблицы для одного состояния приведен на рис. 1 (контуром обведена информативная часть таблицы, именно она и заносится в хромосому). Отметим, что в каждой строке таблицы записано множество действий, а не их последовательность. Задание значения функции действий в виде множества значительно упрощает оператор скрещивания и повышает эффективность процесса эволюции. Выполнение на переходе последовательности действий эквивалентно осуществлению нескольких переходов, на которых выполняются множества действий. Таким образом, автомат в исходной модели всегда может быть записан в рассмотренной выше табличной форме, возможно с добавлением нескольких состояний и переходов. После получения результата оптимизации лишние элементы автомата можно устранить, преобразовав множества действий в последовательности.

x_0	x_1	s	z_0	z_1	z_2
0	0	0	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	1	2	1	1	1

Рис. 1. Хромосома состояния: полная таблица (n = 2, m = 3, $|S|$ = 3)

Все таблицы, соответствующие состояниям одного автомата, имеют одинаковую размерность, так как число предикатов и действий объекта управления задано по условию задачи. Что касается управляющих состояний, то их число также должно быть известно: эта информация используется при случайной генерации значений функции переходов. Авторами реализован подход, при котором число управляющих состояний задается перед началом оптимизации и далее не изменяется. При этом число состояний можно задавать исходя из априорных представлений о сложности задачи, причем с некоторым «запасом»: в процессе оптимизации лишние состояния станут недостижимыми, и их можно будет автоматически исключить. Однако неоправданно большое число состояний негативно влияет на скорость эволюции. В этом смысле более эффективным является постепенное наращивание числа управляющих состояний в процессе оптимизации. Этот вариант также несложно реализуется в рамках предлагаемого подхода к представлению конечных автоматов в виде особей.

Опишем теперь генетические операторы над хромосомами состояний, записанными предложенным выше способом (в виде полных таблиц).

Алгоритм 1. Мутация полных таблиц. Алгоритм мутации состояния, представленного полной таблицей, описан на псевдокоде в листинге 1 и проиллюстрирован на рис. 2 (здесь и далее на стрелках, обозначающих изменения значений в ячейках таблицы, написаны вероятности этих изменений). При мутации состояния с некоторой вероятностью может мутировать каждый элемент таблицы. При этом номер целевого состояния изменяется на любой из допустимых, а вместо исходного набора действий генерируется новый набор, вероятность появления единиц в котором равна доле единиц в исходном наборе.

Листинг 1. Мутация полных таблиц

```
State Mutate(State state)
{
      State mutant = state;
      for (для всех i: строк таблицы)
      {
            if (с вероятностью p1) {
                  mutant[i].targetState = случайное число от 0 до nStates - 1;
            }
            if (с вероятностью p2) {
                  int nActsPresent = число единиц в mutant[i].output;
                  if ((nActsPresent == 0) || (nActsPresent == nActions)) {
                        Index j = случайное число от 0 до nActions - 1;
                        mutant[i].output[j] = !mutant[i].output[j];
                  } else {
                        for (для всех j: номеров действий) {
                              mutant[i].output[j] = 1 с вероятностью
                        nActsPresent/nActions и 0 иначе;
                        }
                  }
            }
      }
      return mutant;
}
```

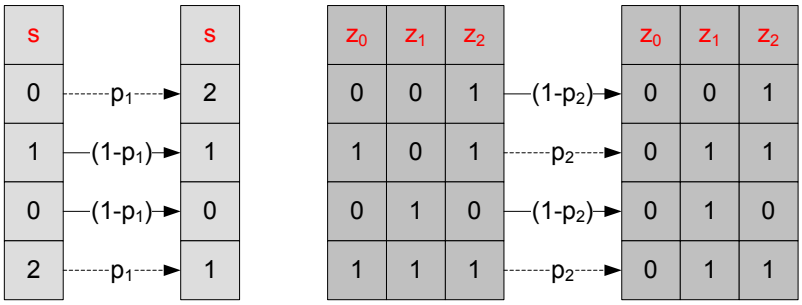


Рис. 2. Пример мутации полных таблиц

Алгоритм 2. Скрещивание полных таблиц. В настоящей работе рассматривается адаптация к предложенному представлению состояний одного способа скрещивания, известного как *одноточечное*. Руководствуясь схожими идеями, нетрудно адаптировать к табличному представлению и другие способы скрещивания.

Алгоритм одноточечного скрещивания полных таблиц представлен в листинге 2 и проиллюстрирован на рис. 3. Специфика данного алгоритма обусловлена тем, что таблицы состояний двумерны, в отличие от традиционного одномерного представления хромосом в виде битовых строк. При скрещивании полных таблиц предлагается по очереди выполнять традиционное одноточечное скрещивание соответствующих столбцов этих таблиц.

Листинг 2. Скрещивание полных таблиц

```
pair<State, State> Cross(State state1, State state2)
{
      State child1 = state1;
      State child2 = child1;
      int tableSize = размер таблицы;

      for (для всех j: столбцов таблицы)
      {
            int crossPoint = случайное число от 0 до tableSize;
            for (для всех i: строк таблицы от 0 до crossPoint - 1) {
                  child1[i][j] = state1[i][j];
                  child2[i][j] = state2[i][j];
            }
            for (для всех i: строк таблицы от crossPoint до tableSize - 1) {
                  child1[i][j] = state2[i][j];
                  child2[i][j] = state1[i][j];
            }
      }
      return make_pair(child1, child2);
}
```

Основная проблема, возникающая при использовании полных таблиц рассмотренного вида – это экспоненциальный рост размерности хромосомы с увеличением числа предикатов объекта управления (напомним, что число строк в таблице 2^n, где n – число предикатов). Опыт показывает, что в реальных задачах управляющие автоматы, построенные вручную, имеют гораздо меньше переходов, чем $|S| \cdot 2^n$. Причина, по мнению авторов, состоит в том, что в большинстве задач предикаты имеют «локальную природу» по отношению к управляющим состояниям. В каждом

состоянии *значимым* является лишь определенный, небольшой поднабор предикатов, остальные же не влияют на значение управляющей функции. Именно это свойство позволяет существенно сократить размер описания состояний. Кроме того, использование этого свойства в процессе оптимизации позволяет получить результат, более похожий на автомат, построенный вручную, а следовательно, и более понятный человеку.

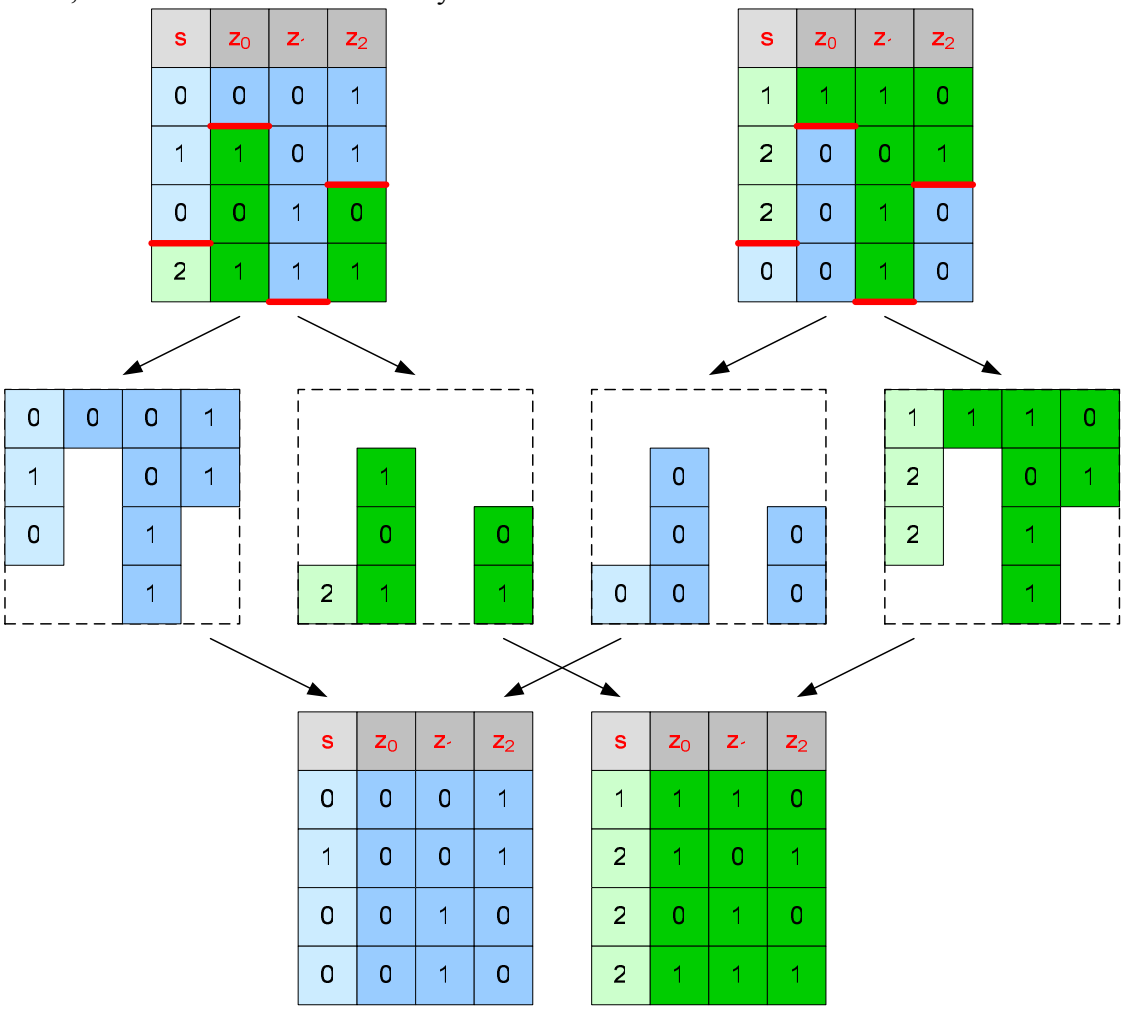


Рис. 3. Пример скрещивания полных таблиц

1.1.3. Представление состояний: сокращенные таблицы

Свойство локальности предикатов можно применять для сокращения описания управляющего состояния разными способами. Авторами выбран один из подходов, при котором число значимых в состоянии предикатов ограничивается некоторой константой r. К таблице, задающей сужение управляющей функции на данное состояние, в этом случае добавляется битовый вектор, описывающий множество значимых предикатов (рис. 4).

x_0	x_1	x_2	x_3	x_4	x_5
0	1	0	1	0	0

x_1	x_3	s	z_0	z_1	z_2
0	0	0	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	1	2	1	1	1

Рис. 4. Хромосома состояния: сокращенная таблица (n = 6, m = 3, r = 2, $|S|$ = 3)

Число строк таблицы в этом случае 2^r, однако, константа r обычно невелика. Ее выбор зависит от сложности задачи. Как показывает опыт, для большинства автоматизированных объектов среднее по всем состояниям значение r не больше пяти.

Опишем генетические операторы над хромосомами состояний, записанными в виде сокращенных таблиц.

Алгоритм 3. Мутация сокращенных таблиц. По сравнению с представлением в виде полной таблицы, добавилась возможность мутации множества значимых предикатов. При этом каждый из значимых предикатов с некоторой вероятностью заменяется другим, который не принадлежит этому множеству (рис. 5).

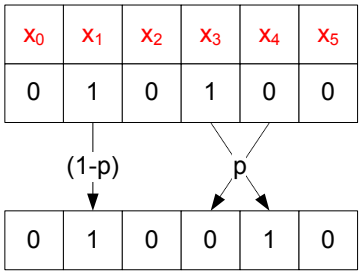


Рис. 5. Пример мутации множества значимых предикатов

Мутация сокращенной таблицы происходит так же, как мутация полной таблицы. Описание алгоритма приведено в листинге 3.

Листинг 3. Мутация сокращенных таблиц

```
State Mutate(State state)
{
      State mutant = state;
      if (с вероятностью p) {
            int from, to;
            случайно выбрать from и to, так что
                  (mutant.predicates[from] == 1) && (mutant.predicates[to] == 0);
            mutant.predicates[from] = 0;
            mutant.predicates[to] = 1;
      }
      // Остальная часть хромосомы мутирует, как в случае полных таблиц
      mutant.table = Mutate(mutant.table);
      return mutant;
}
```

Алгоритм 4. Скрещивание сокращенных таблиц. Это наиболее сложный из предлагаемых алгоритмов. Основная последовательность его шагов отражена в листинге 4.

Листинг 4. Скрещивание сокращенных таблиц

```
pair<State, State> Cross(State state1, State state2)
{
      State child1 = state1;
      State child2 = child1;

      ChoosePreds(state1.predicates, state2.predicates,
            child1.predicates, child2.predicates);

      int crossPoint = случайное число от 0 до tableSize;
      FillChildTable(state1, state2, child1, crossPoint);
      FillChildTable(state1, state2, child2, crossPoint);
      return make_pair(child1, child2);
}
```

Поскольку родительские хромосомы, представленные сокращенными таблицами, могут иметь разные множества значимых предикатов, сначала необходимо выбрать, какие из этих предикатов будут значимы для хромосом детей. Функция `ChoosePreds`, осуществляющая этот выбор, представлена в листинге 5.

Листинг 5. Выбор значимых предикатов детей при скрещивании сокращенных таблиц

```
void ChoosePreds(Predicates p1, Predicates p2, Predicates ch1, Predicates ch2)
{
      for (для всех i: номеров предикатов) {
            if (p1[i] && p2[i]) {          // Предикат от обоих родителей
                  ch1[i] = ch2[i] = true; // достается обоим детям
                  запоминаем, что в наборах предикатов детей стало на 1 меньше
            места;
            }
      }
      for (для всех i: номеров предикатов) {
            if (p1[i] != p2[i]) {
                  Predicates* pCh;
                  if (у обоих детей есть место) {
                        pCh = равновероятно любой ребенок;
                  } else {
                        pCh = тот ребенок, у которого еще есть место;
                  }
                  (*pCh)[i] = true;
                  запоминаем, у кого стало меньше места;
            }
      }
}
```

В результате работы функции `ChoosePreds` предикаты, значимые для обоих родителей, наследуются обоими детьми, а каждый из тех предикатов, которые были значимы лишь для одной родительской особи, равновероятно достанется любому из двух детей. Пример работы функции для родительских хромосом, представленных на рис. 6, проиллюстрирован на рис. 7.

x_0	x_1	x_2	x_3	x_4	x_5
0	1	0	1	0	0

x_0	x_1	x_2	x_3	x_4	x_5
1	1	0	0	0	0

x_1	x_3	s	z_0	z_1	z_2
0	0	0	0	0	1
0	1	1	1	0	1
1	0	0	0	1	0
1	1	2	1	1	1

x_0	x_1	s	z_0	z_1	z_2
0	0	1	1	1	0
0	1	2	0	0	1
1	0	2	0	1	0
1	1	0	0	1	0

Рис. 6. Родительские хромосомы, представленные сокращенными таблицами

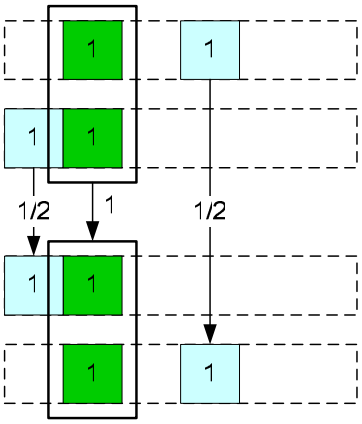


Рис. 7. Пример выбора значимых предикатов детей

После выбора значимых предикатов заполняются таблицы обоих детей. Алгоритм заполнения представлен в листинге 6.

Листинг 6. Заполнение таблиц детей при скрещивании сокращенных таблиц

```
void FillChildTable(State s1, State s2, State& child, int crossPoint)
{
      for (для всех i: строк таблицы child) {
            vector<int> lines1 = выбрать строки таблицы s1, в которых предикаты,
                  значимые для child, имеют те же значения, что в строке i,
                  причем, если предикат значим для обоих родителей и i >=
            crossPoint, то его значение не учитывается;
            vector<int> lines2 = выбрать строки таблицы s2, в которых предикаты,
                  значимые для child, имеют те же значения, что в строке i,
                  причем, если предикат значим для обоих родителей и i <
            crossPoint, то его значение не учитывается;

            vector<Probability> p1(nStates);
            vector<Probability> p2(nStates);
            for (для всех j из lines1) {
                  p1[целевое состояние у s1 в строке j] += 1.0;
```

```
                    }
                    for (для всех j из lines2) {
                          p2[целевое состояние у s2 в строке j] += 1.0;
                    }
                    Поделить значения p1 на число строк из lines1;
                    Поделить значения p2 на число строк из lines2;
                    vector<Probability> p = p1 + p2;
                    child[i].targetState = выбрать случайно с распределением вероятностей
                    p;

                    for (для всех k: номеров действий) {
                          Probability q1, q2;
                          for (для всех j из lines1) {
                                q1 += s1[j].output[k];
                          }
                          for (для всех j из lines2) {
                                q2 += s2[j].output[k];
                          }
                          Поделить q1 на число строк из lines1;
                          Поделить q2 на число строк из lines2;
                          child[i].output[k] = 1 с вероятностью (q1 + q2)/2 и 0 иначе;

                    }
              }
        }
```

Иллюстрация примера заполнения первой строки таблицы одного из детей приведена на рис. 8. В данной реализации оператора скрещивания на значения каждой строки таблицы ребенка влияют значения нескольких строк родительских таблиц. При этом конкретное значение, помещаемое в ячейку таблицы ребенка, определяется «голосованием» всех влияющих на нее ячеек родительских таблиц.

В описанном выше варианте алгоритма все состояния автомата имеют равное число значимых предикатов (r – константа для всего процесса оптимизации). Однако предложенный алгоритм скрещивания легко расширяется на случай разного числа значимых предикатов у пары родителей.

1.1.4. Мутация, зависящая от пригодности

В классическом генетическом алгоритме мутации применяются к хромосомам с некоторой вероятностью, постоянной в процессе оптимизации. Авторы считают целесообразным введение зависимости вероятности мутации особи от ее пригодности.

Применение интенсивной мутации к плохо приспособленным особям повышает эффективность эволюции, позволяет покидать локальные оптимумы и ослабляет проблему преждевременной сходимости популяции, тогда как для особей с высокой пригодностью оптимальны менее интенсивные мутации.

В метрическом пространстве оптимальный модуль вектора мутации можно с высокой точностью оценить сверху расстоянием между особью и глобальным оптимумом оценочной функции. Это расстояние в задаче не известно, однако можно ожидать тенденции к его сокращению с ростом пригодности особи.

Для иллюстрации рассмотрим частный случай. Пусть пространство поиска евклидово, а функция пригодности непрерывна и возрастает с приближением к оптимуму. В этом случае можно проанализировать зависимость эффективности мутации от ее модуля.

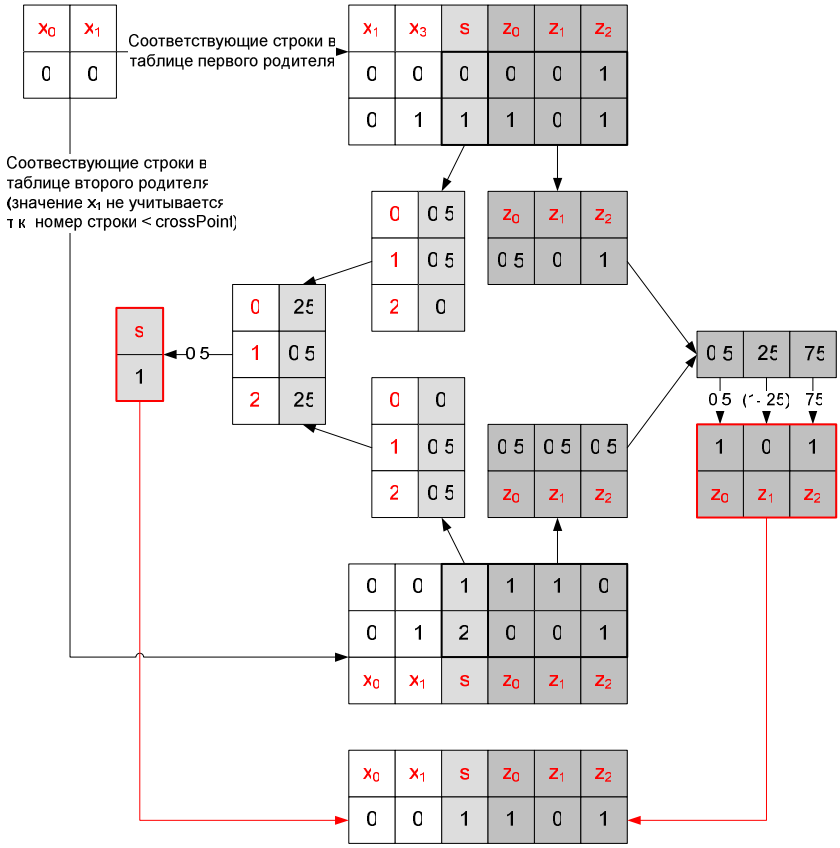


Рис. 8. Пример заполнения строки таблицы ребенка при скрещивании сокращенных таблиц

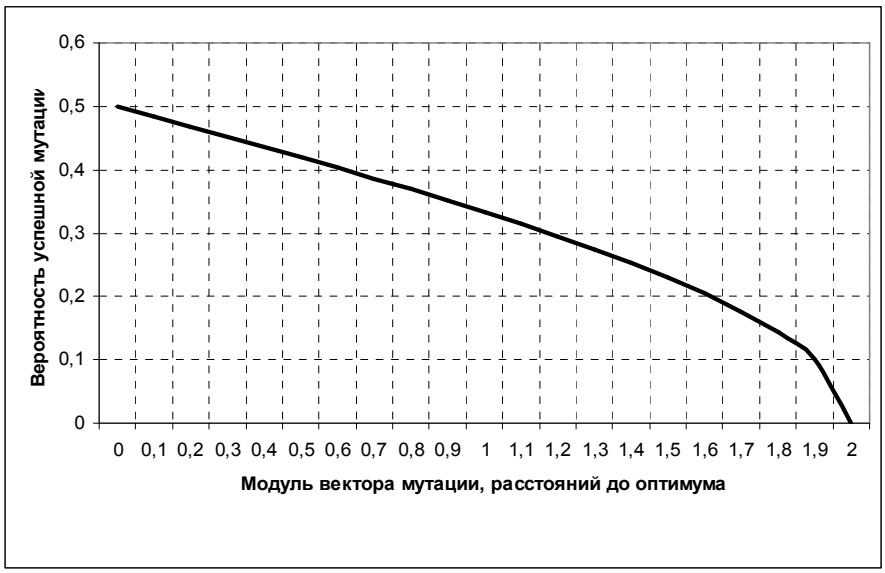


Рис. 9. Вероятность улучшения пригодности после мутации

График на рис. 9 отражает снижение вероятности улучшения пригодности особи после мутации с увеличением модуля вектора мутации. График на рис. 10 показывает характерное изменение расстояния до оптимума при успешной мутации.

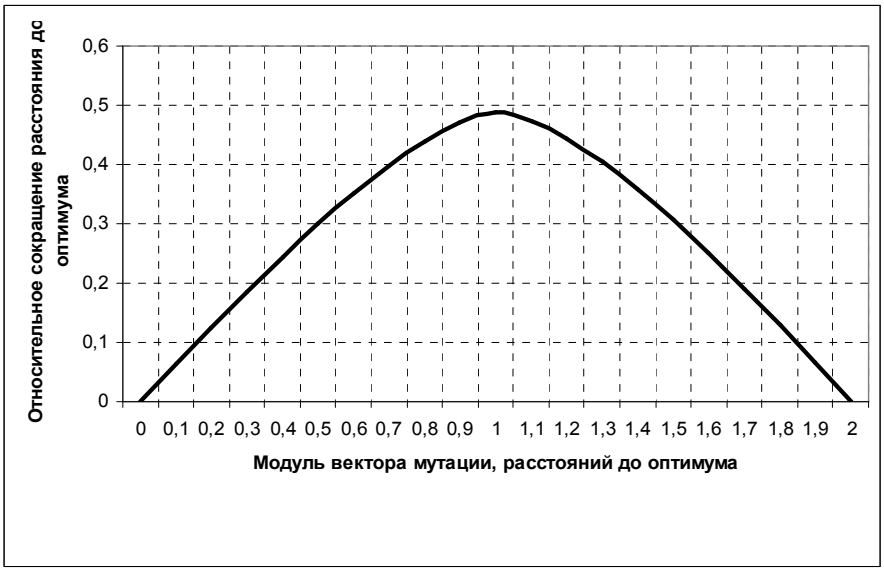


Рис. 10. Математическое ожидание относительного сокращения расстояния до оптимума в результате успешной мутации

Приведенная выше диаграмма показывает, что максимальная эффективность успешной мутации достигается в том случае, когда ее модуль равен (неизвестному) расстоянию до оптимума.

Из двух предыдущих диаграмм получена диаграмма, представленная на рис. 11. Она отражает зависимость математического ожидания относительного сокращения расстояния до оптимума в результате попытки применения мутации.

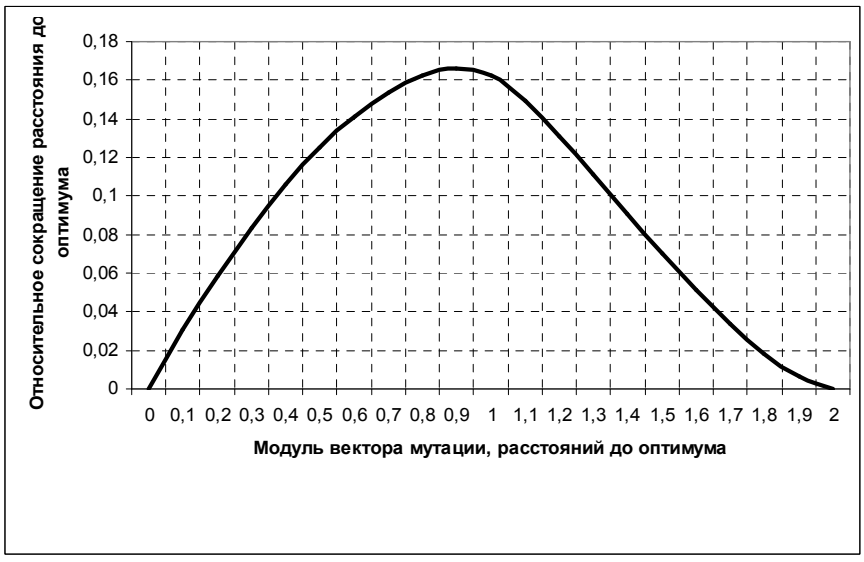


Рис. 11. Математическое ожидание сокращения расстояния до оптимума за одну операцию оценки пригодности

Оптимальная интенсивность мутации зависит от неизвестного расстояния до оптимума. С другой стороны, пригодность имеет тенденцию к возрастанию по мере приближения к оптимуму. Из этого можно сделать вывод о склонности оптимальной интенсивности мутации к обратной зависимости от значения оценочной функции. В экспериментах авторы использовали интенсивность мутации, обратно пропорциональную пригодности.

1.1.5. Формирование нового поколения

Для учета зависимости интенсивности мутации от пригодности требуется наличие оценки пригодности особи перед ее мутацией. В методах генетической оптимизации стандартный подход к формированию нового поколения предполагает следующий порядок действий: выбор пары особей из популяции родителей, их скрещивание, мутации детей, и, наконец, добавление детей, возможно мутировавших, в новую популяцию. Однако в этом случае требуется дополнительная оценка пригодности детей перед мутацией, что приводит к удвоению объема вычислений оценочной функции. Для сохранения эффективности оптимизации необходимо внести изменения в процесс формирования нового поколения. Например, можно добавить новую особь в популяцию два раза – до и после мутации – или выбрать лучшую из них.

В рамках использования метода сокращенных таблиц предлагается другой подход, предполагающий применение к каждой особи поколения не более одного генетического оператора. При этом часть особей с лучшей пригодностью переносится в новую популяцию без изменений, а каждое из оставшихся в новой популяции мест заполняется либо результатом скрещивания пары особей из родительской популяции, либо результатом мутации особи из родительской популяции.

1.1.5.1. Выбор особей для скрещивания

В процессе экспериментальной проверки был использован подход, при котором вероятность выбора особи в качестве родителя пропорциональна ее пригодности. Этот метод известен под названием *fitness proportional selection*.

1.1.5.2. Модификация оценочной функции

Задание оценочной функции на множестве вычислительных состояний объекта управления позволяет оптимизировать *поведенческие* свойства управляющего автомата. Однако, в целях улучшения понятности найденного результата оптимизации для человека, авторы считают целесообразным оптимизировать также *структурные* свойства автомата. Предлагается ввести набор стандартных структурных характеристик (например, число достижимых управляющих состояний, суммарное число действий на переходах), которые, по желанию разработчика, будут влиять на оценку пригодности особей.

1.2. Метод представления автоматов деревьями решений

Для генерации автоматов применяются различные модификации эволюционных алгоритмов [49]. Однако, для большинства из этих методов размер хромосомы, требуемый для хранения автомата, экспоненциально растет от числа входных переменных.

При решении задачи классификации плотности Дж. Козой [1] применялось представление функции переходов автомата с помощью деревьев разбора [52], реализующих булевы функции. Это представление обладает существенно большей выразительностью по сравнению с традиционными табличными методами, что позволило эффективно решить задачу. Однако обычно требуется строить автоматы, число состояний в которых больше двух, и поэтому функция переходов не может быть представлена как булева.

Поэтому разработаем метод генерации автоматов, являющийся обобщением метода Дж. Козы. Этот метод должен допускать высокоуровневое представление автомата, позволяющее уменьшить размер хромосомы.

Наиболее перспективным является использование такого метода в рамках *композитного генетического программирования* [46]. Оно позволяет быстро находить хорошие приближения решений, которые потом могут быть улучшены с помощью других эволюционных алгоритмов.

В этом разделе излагается метод генерации автоматов с использованием деревьев решений [55]. Этот метод позволяет эффективно представлять автоматы с дискретными входными переменными.

1.2.1. Представление автоматов деревьями решений

Дерево решений является удобным способом задания дискретной функции, зависящей от конечного числа дискретных переменных.

Оно представляет собой помеченное дерево, метки в котором расставлены по следующему правилу:

- внутренние узлы помечены символами переменных;
- ребра – значениями переменных;
- листья – значениями искомой функции.

Для определения значения функции по значениям переменных необходимо спуститься от корня до листа, и сформировать значение, которым помечен полученный лист. При этом из вершины, помеченной переменной x, переход производится по тому ребру, которое помечено тем же значением, что и значение переменной x.

Пример дерева решений изображен на рис. 12. Здесь для простоты показано дерево, реализующее булеву функцию от переменных a, b и c.

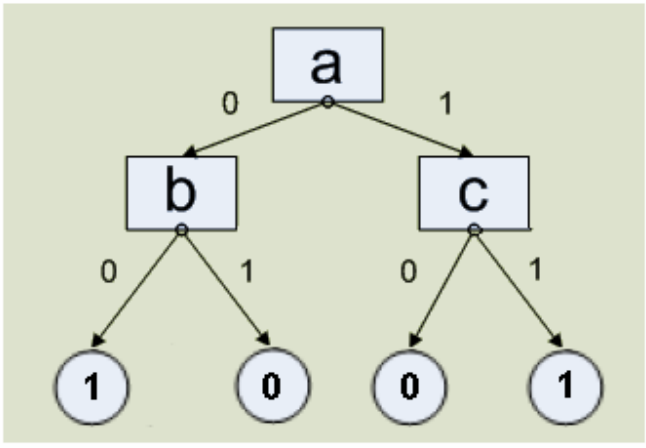


Рис. 12. Пример дерева решений

Как правило, в большинстве практических задач дерево решений требует значительно меньше памяти по сравнению с заданием функции на всех наборах значений входных переменных (с помощью таблиц истинности).

Опишем предлагаемый метод представления автомата с помощью деревьев решений. Для задания автомата необходимо выразить его функции переходов и выходов с помощью деревьев

решений. Возможно осуществить следующее преобразование автомата: вместо задания функций переходов и выходов для автомата в целом, представим эти функции для каждого состояния.

Более формально: зададим для каждого состояния $q \in Q$ функцию $\sigma_q : X \to Q \times Y$, такую что $\sigma_q(x) = (\delta(q, x), \lambda(q, x))$ для $\forall x \in X$.

Функции σ_q соответствуют функциям переходов и выходов из состояния q. Каждая из этих функций может быть выражена с помощью дерева решений. В этих деревьях переменными являются входные переменные автомата, а множеством значений — все возможные пары (*Новое Состояние*, *Действие*). **Таким образом, автомат в целом задается упорядоченным набором деревьев решений.**

На рис. 2 приведен пример автомата Мили. При этом *a*, *b*, *c* — входные переменные булевого типа, а *L*, *R*, *F* — выходные действия.

Этот автомат с использованием деревьев решений приведен на рис. 14.

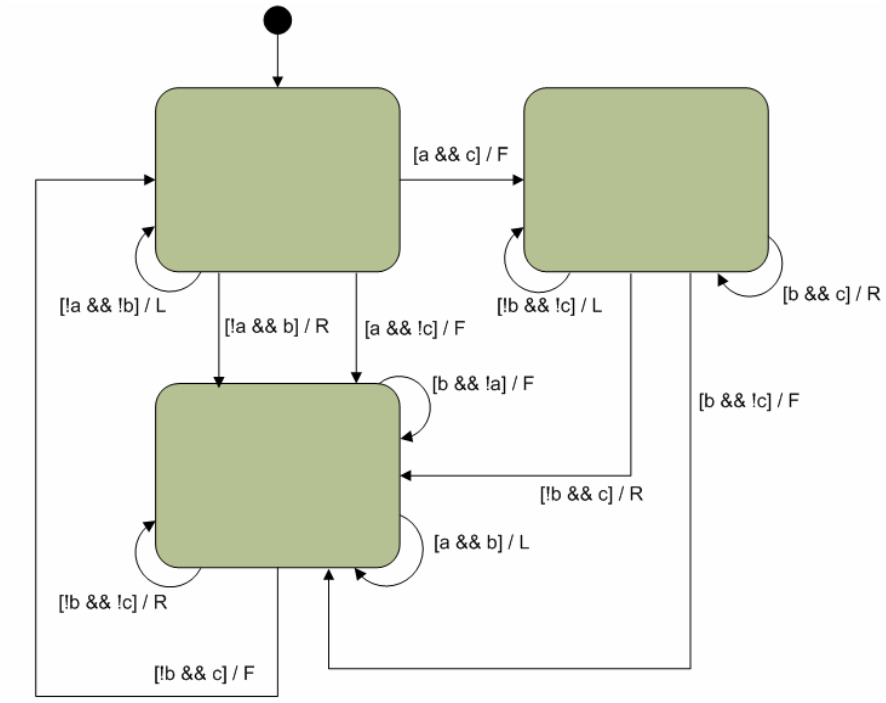


Рис. 13. Пример автомата Мили

На рисунке показано, что выбор переходов из каждого состояния осуществляется с помощью некоторого дерева решений.

При таком представлении функции переходов из каждого состояния можно ожидать значительного уменьшения длин хромосом по сравнению с хранением ее в виде таблицы. Как правило, в автоматном программировании [54] в каждом состоянии важен лишь сравнительно небольшой набор из всего множества входных переменных автомата. Таким образом, представление автомата с помощью набора деревьев решений должно значительно сократить длину хромосомы, а, следовательно, и ускорить генерацию автоматов.

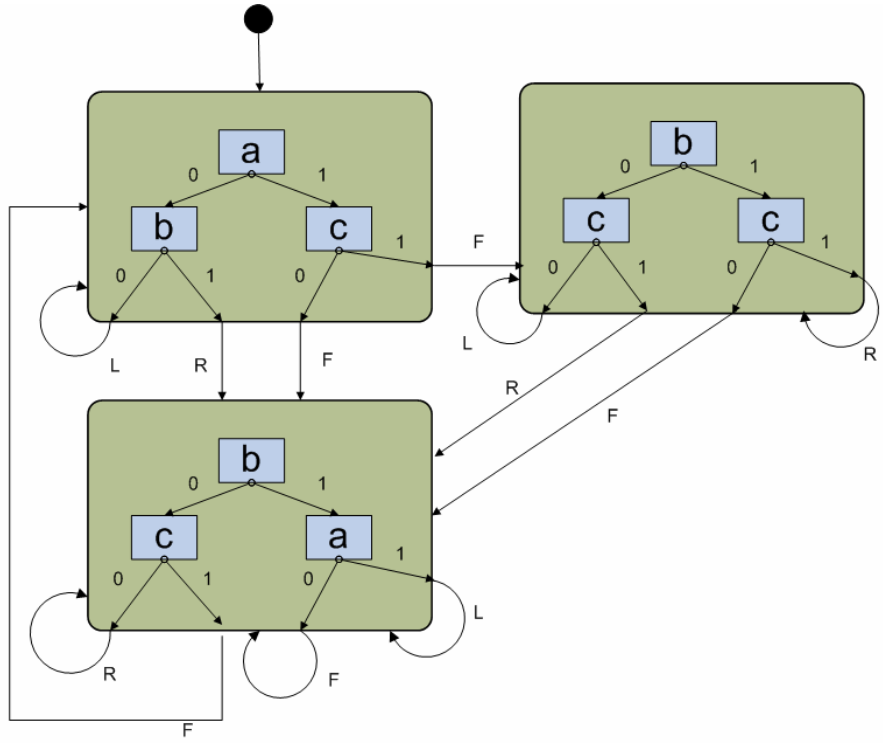


Рис. 14. Представление автомата Мили с помощью деревьев решений

1.2.2. Генетические операции

Для использования представления автоматов в виде набора деревьев решений в генетических алгоритмах определим следующие операции:

- случайное порождение автомата — в каждом состоянии создается случайное дерево решений;
- скрещивание автоматов — скрещиваются деревья решений в соответствующих состояниях;
- мутация автомата — в случайном дереве решений производится мутация.

Здесь считается, что число состояний в автомате фиксировано, и поэтому противоречий при выполнении определенных таким образом операций не возникнет.

После определения операций над набором деревьев решений, определим генетические операции над собственно деревьями решений. Это можно сделать с помощью операций, аналогичных операциям, используемым в генетическом программировании над деревьями разбора [15, 52]. В деревьях решений переменные являются нетерминалами, а значения функции — терминалами. При этом арность нетерминала, соответствующего переменной x, равна мощности множества возможных значений переменной. Приведем описание генетических операций над деревьями решений:

- Случайное порождение дерева решений — случайным образом выбирается метка: либо одно из возможных значений функции переходов, либо одна из переменных. После этого создается вершина, помеченная выбранной меткой. При этом если была выбрана переменная, то рекурсивно генерируются дети вершины, иначе вершина становится листом дерева.
- Мутация — выбирается случайный узел в поддереве. После этого поддерево, соответствующее выбранному узлу, заменяется на случайно сгенерированное (рис. 15).

- Скрещивание — в скрещиваемых деревьях выбираются два случайных узла. После этого поддерево, соответствующее выбранному узлу в первом дереве, заменяется поддеревом, соответствующим узлу второго дерева (рис. 16).

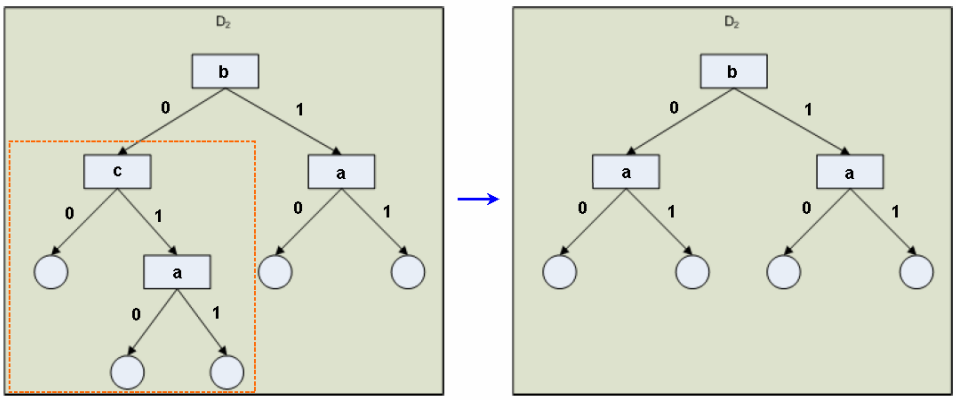


Рис. 15. Пример мутации деревьев решений

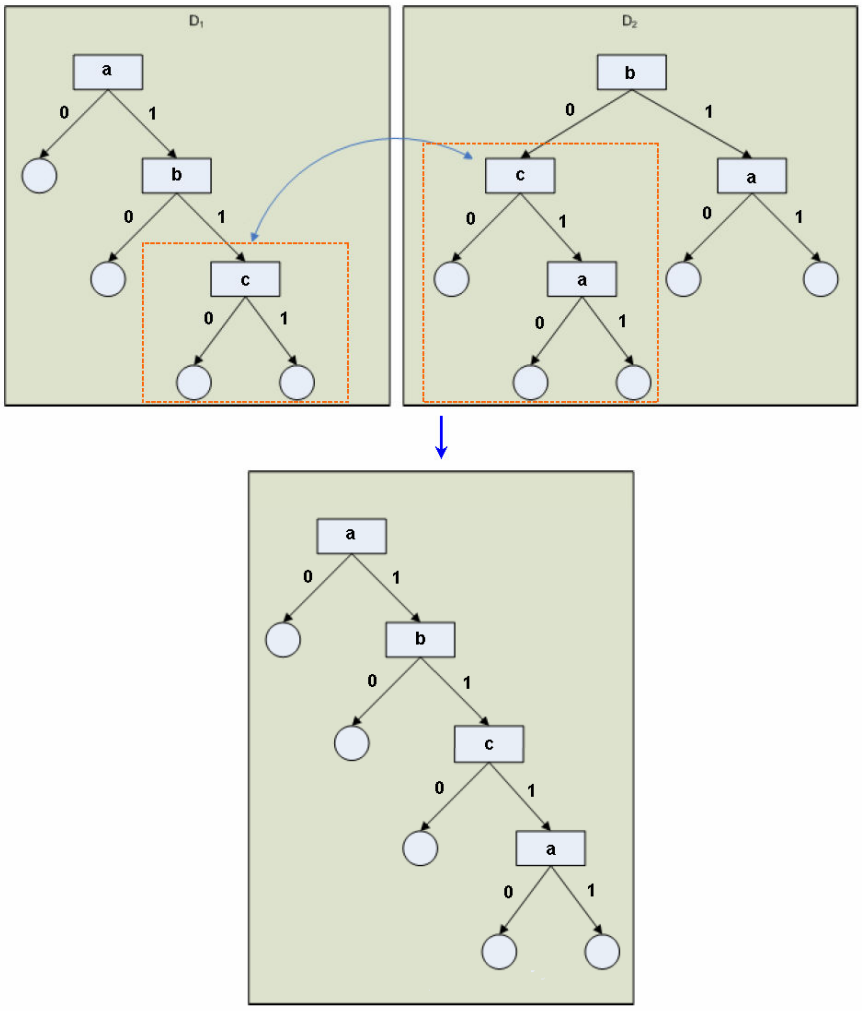


Рис. 16. Пример скрещивания деревьев решений

Отметим, что заданные таким образом операции могут порождать деревья, в которых некий атрибут встречается на пути от корня до листа дважды (например, дерево, полученное в результате скрещивания на рис. 16). Такие деревья являются корректными — определяют функцию на любом наборе значений атрибутов единственным образом, однако имеют недостижимые вершины. Действительно, если атрибут встречается на пути дважды, то его значение уже известно, следовательно, одна из ветвей недостижима ни при каких значениях предикатов. Появление недостижимых ветвей может влиять на эволюцию отрицательным образом по следующим причинам:

- Появление недостижимых ветвей ведет к увеличению высоты деревьев, экспоненциально увеличивая память, необходимую на хранение популяции. В условиях ограниченной памяти это ведет к уменьшению популяции.
- Недостижимые ветви дерева могут являться плохими приближениями искомой функции (так как не учитываются при подсчете функции приспособленности), но при этом, однако, копироваться в потомков, замедляя генерацию.

Таким образом, необходимо ввести операцию обрезки — удаление недостижимых ветвей. Операция может быть выполнена следующим образом: узел, одна из дочерних вершин которого недостижима, заменяется на свою достижимую дочернюю вершину. На рис. 17 приведен пример обрезки недостижимых ветвей. Операция обрезки должна выполняться после генетических операций скрещивания и мутации.

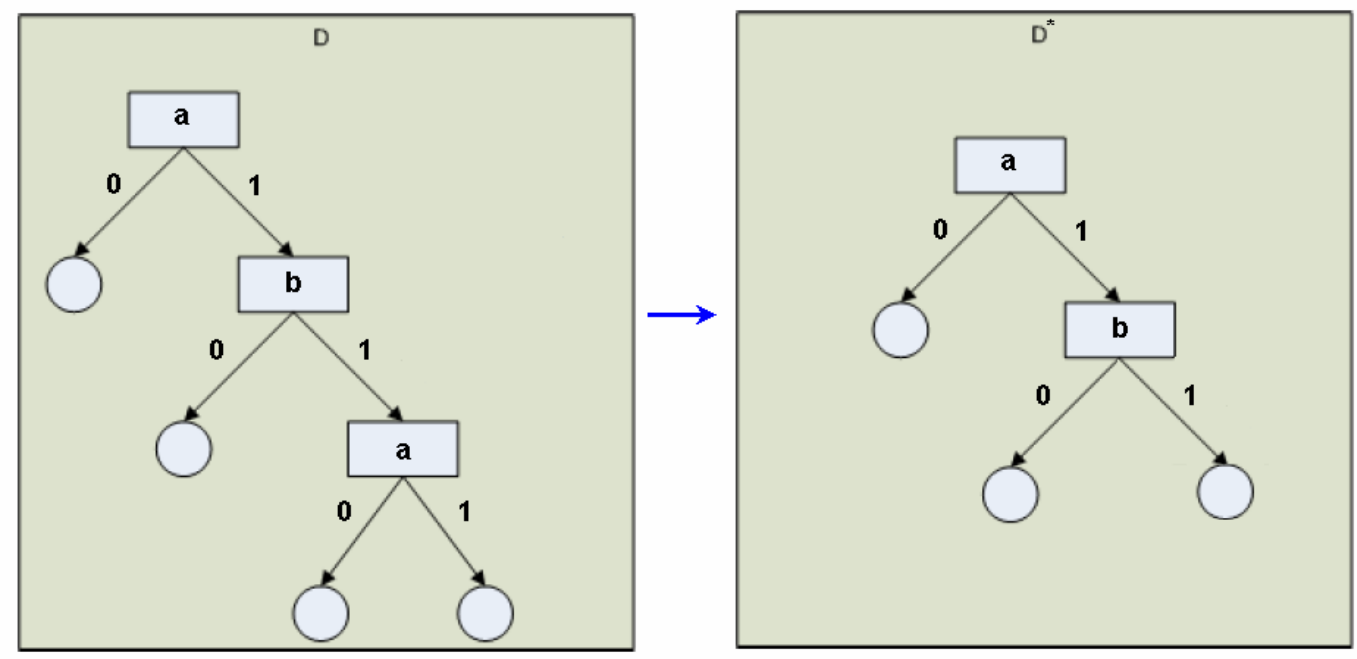


Рис. 17. Пример обрезки недостижимых ветвей

1.2.3. Задание ограничений на целевой автомат

Заметим, что каждый лист дерева решений задает ровно один переход из состояния. Таким образом, можно считать, что число листьев во всех деревьях равно числу переходов автомата. Здесь

не учитывается тот факт, что два листа могут задавать один и тот же переход. Вместе с тем, число листьев в дереве является довольно естественной оценкой числа переходов и может быть легко вычислено. Высота деревьев, задающих автомат, может быть естественной оценкой сложности логики автомата.

Таким образом, разработанный метод позволяет накладывать ограничения на следующие характеристики генерируемых автоматов:

- суммарное число переходов автомата;
- максимальное число переходов из состояния;
- сложность логики (максимальная высота деревьев решений).

Для учета ограничений можно применить следующие классические способы задания ограничений на хромосомы, используемые в генетических алгоритмах:

- повторение генетических операций до получения результата, соответствующего ограничениям;
- внесение штрафных функций за нарушение наложенных ограничений в подсчет функции приспособленности.

1.2.4. Возможные модификации метода

Разработанный метод обеспечивает возможность построения модификаций. Приведем некоторые из них:

- Возможно раздельное хранение функций переходов и выходов. В этом случае в каждом состоянии будет храниться не одно дерево, соответствующее функции σ_q, а два дерева, соответствующие функциям δ_q и λ_q. Генетические операции над автоматами, заданными таким образом, определяются аналогично.
- Предлагаемый метод выше был изложен для генерации автоматов Мили. Отметим, что он может быть адаптирован и для генерации автоматов Мура. В таком случае, для каждого из переходов, хранимых в узле дерева решений, необходимо хранить только вершину, в которую ведет рассматриваемый переход. Кроме того, в каждом состоянии необходимо хранить ассоциированное выходное воздействие, совершаемое при входе в вершину. Генетические операции над автоматами дополняются скрещиванием наборов выходных воздействий в состояниях любым из способов, используемых в генетических алгоритмах.

1.2.5. Простой пример применения

Ниже демонстрируется простой пример применения разработанного метода к решению задачи «*Умный муравей*» [21]. В данной задаче используется одна входная переменная булевого типа. Обозначим его *X*. Выходные воздействия поворота налево, направо и шага вперед обозначим через *L*, *R* и *F* соответственно. Если дерево решений, соответствующее состоянию, не использует никаких входных переменных, то будем изображать один переход, а не дерево.

Для простоты продемонстрируем работу метода на примере популяции, размером четыре. Число состояний в каждом из автоматов положим равным двум.

Допустим, что на определенном шаге генерации получена следующая популяция (рис. 18). Здесь под каждым из автоматов популяции приведено значение фитнесс-функции — количество еды, съеденной за отведенное время.

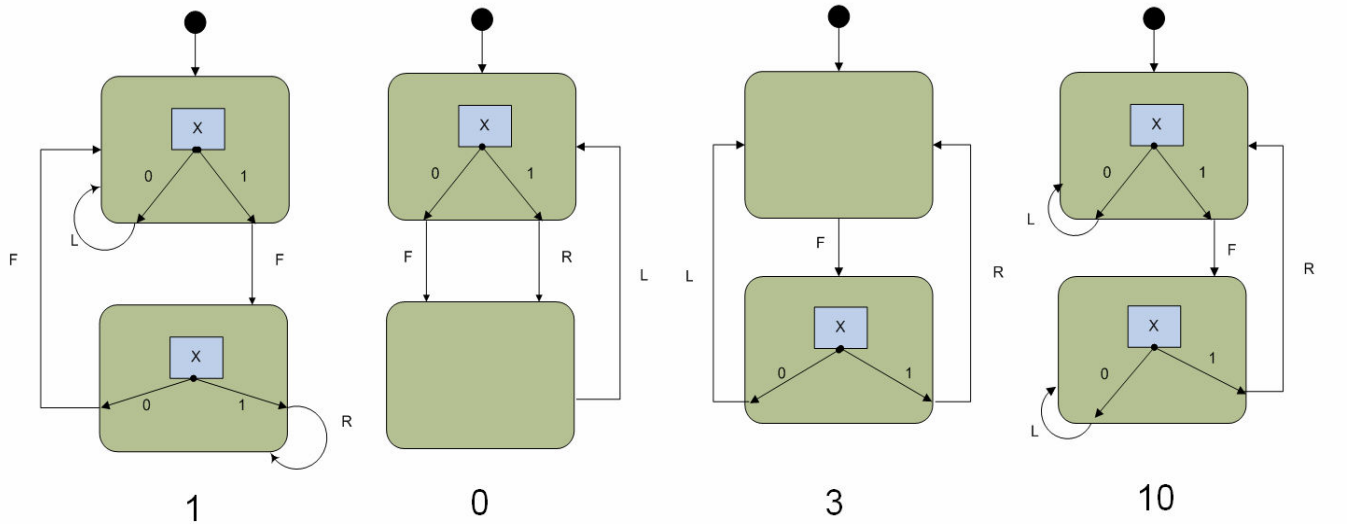


Рис. 18. Популяция автоматов

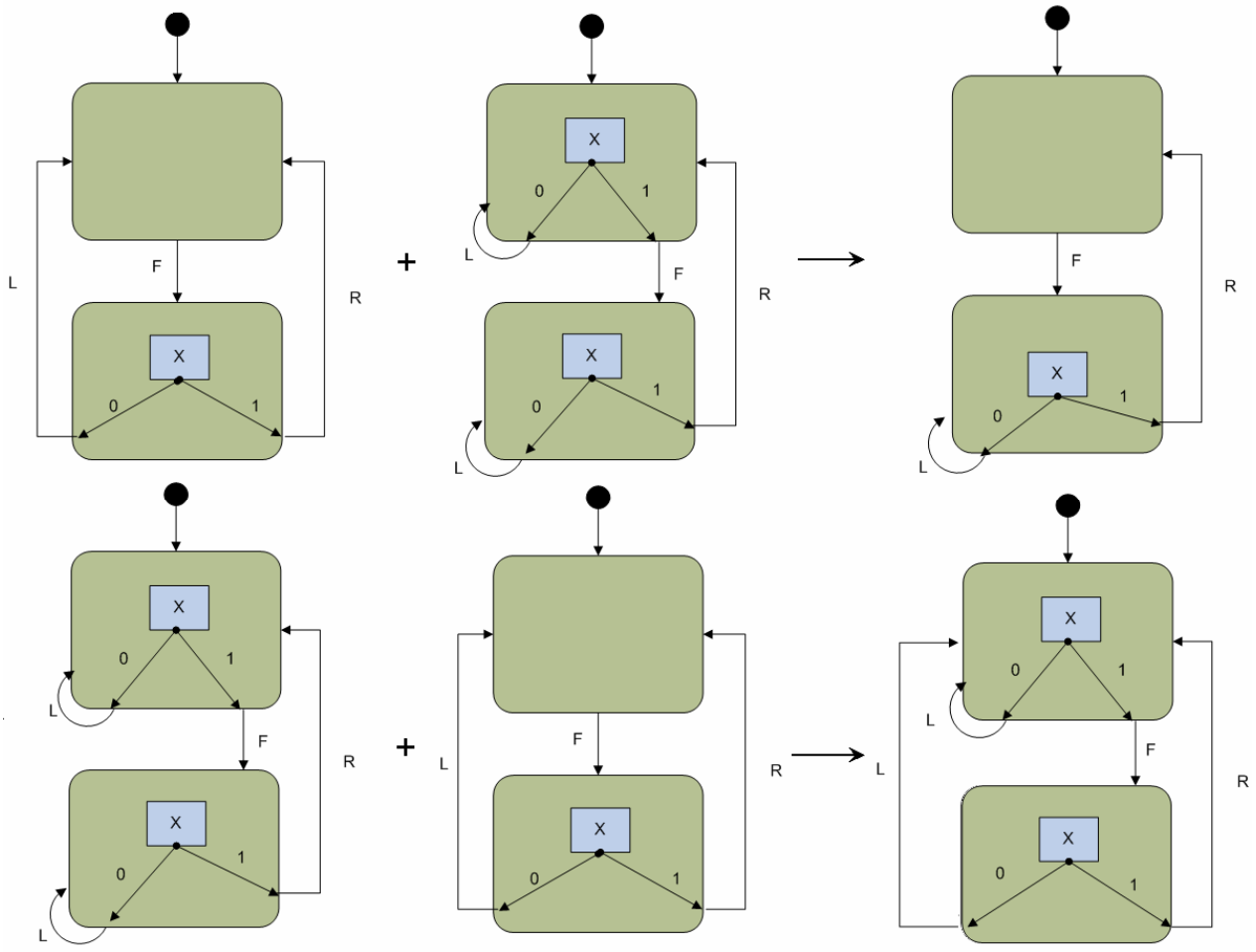


Рис. 19. Порождение потомства

В случае, когда в качестве метода отбора применяется метод рулетки [49], для производства потомства выбирается несколько лучших представителей. Пусть в рассматриваемом примере отбираются два лучших автомата популяции. Тогда два оставшихся автомата покидают популяцию, а их место занимают потомки отобранных для размножения представителей. Выбор очередной пары особей для скрещивания происходит случайным образом. Порождение потомства путем скрещивания показано на рис. 19.

После этого порожденное скрещиванием потомство подвергается действию мутаций. Операция совершается над каждым из порожденных автоматов. С определенной долей вероятности происходит мутация в одном из состояний автомата — над соответствующим деревом решений. На рис. 20 приведен пример действия мутаций на порожденное потомство.

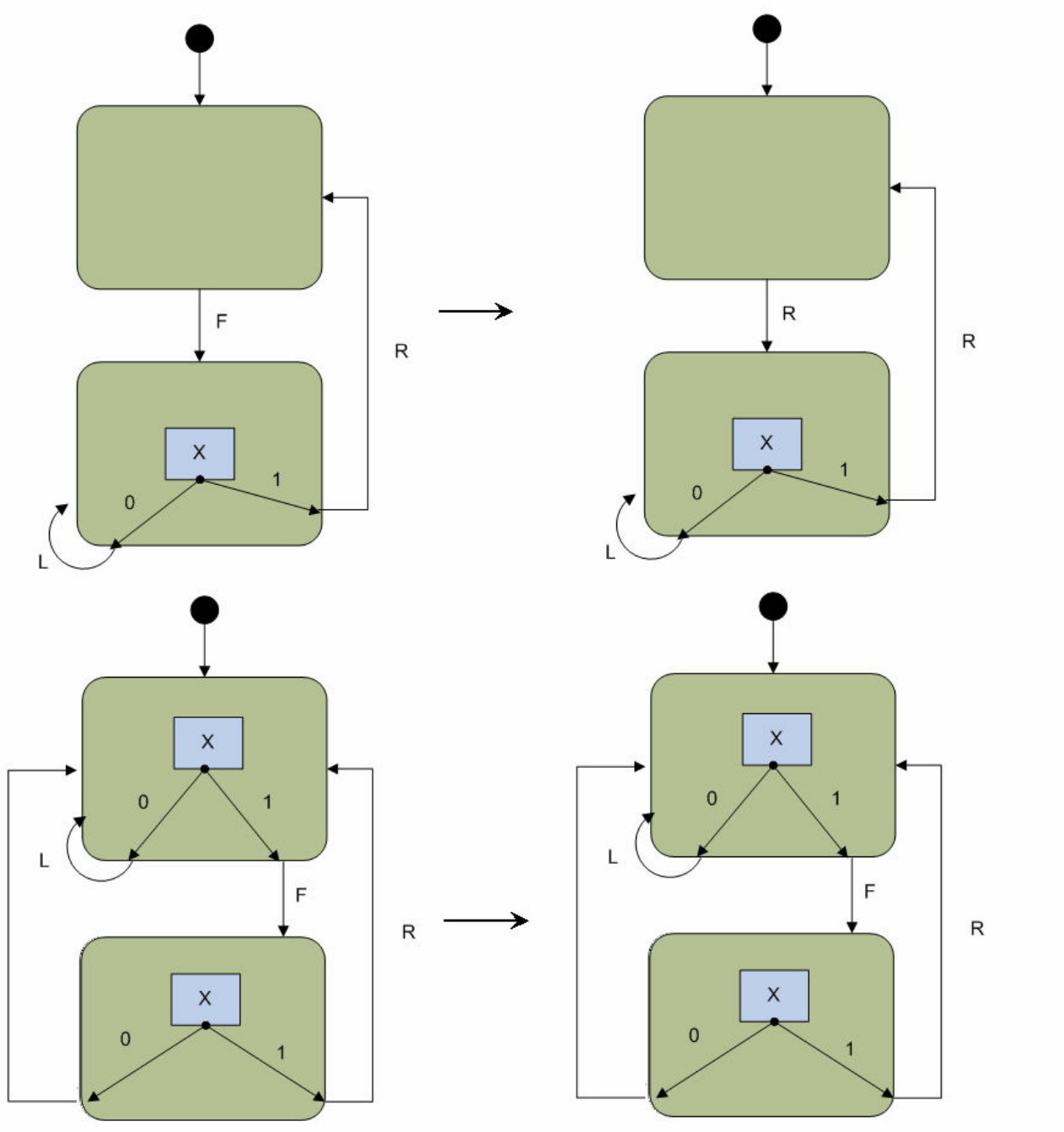


Рис. 20. Мутация потомства

Таким образом, популяция преобразовалась, как показано на рис. 21.

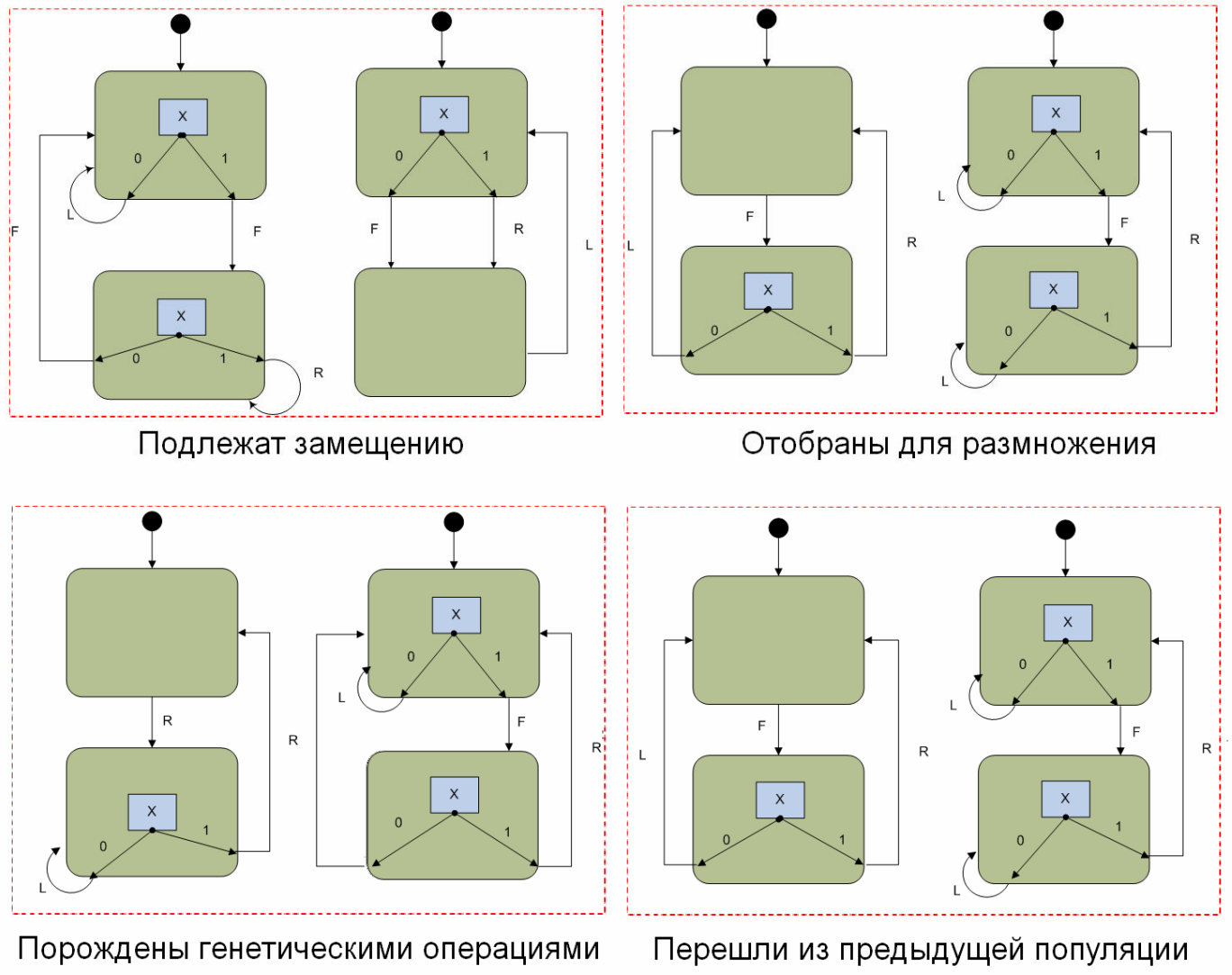


Рис. 21. Преобразование популяции

Многократное повторение подобных шагов генерирует искомый автомат.

1.3. Методы оптимизации генетических алгоритмов для построения автоматов

Генетические алгоритмы применяются при решении широкого круга задач. Однако при использовании классических генетических алгоритмов часто требуются очень большие вычислительные ресурсы для того, чтобы получить решение с необходимой степенью точности. В ряде случаев классические алгоритмы могут не справиться с поставленной задачей за разумное время.

Для повышения скорости сходимости и улучшения устойчивости генетического алгоритма, его, как правило, модифицируют с учетом особенностей задачи, к которой он применяется. В работах [20, 30] рассматриваются оценочные функции и способы их адаптации для улучшения эффективности генетических алгоритмов. В работе [9] на примере задачи коммивояжера рассматривается как зависит эффективность эволюционных методов от выбора структуры хромосом. Необходимо отметить, что выбор структуры хромосом определяет какие классы операторов скрещивания и мутации могут использоваться.

В данном разделе рассматривается три новых модификации генетического алгоритма для класса задач, в которых решением является конечный автомат. Эти модификации могут использоваться как независимо друг от друга, так и совместно.

1.3.1. Использование автоматов с флагами

Поведение конечного автомата зависит от его состояний и переходов между ними. Каждый переход определяет в какое состояние перейдет автомат при некотором условии (функция, принимающая значения входных переменных и возвращающая булевское значение) и какие действия выполнятся при переходе. Иногда в качестве входных переменных на переходе используются значения выходных переменных, генерируемых самим автоматом. Такие переменные в работе [54] названы флагами. Значения флагов также влияют на окружение, в котором работает автомат.

Чем более сложную задачу решает автомат, тем, как правило, больше число его состояний. В книге [54] определен класс автоматов с флагами, в которых используется вектор многозначных флаговых переменных `F`, который несет информацию о предыдущих состояниях автомата. Если автоматы без флагов зависят от предыстории (предыдущего состояния), то автоматы с флагами зависят от глубокой предыстории (не только от предыдущего состояния). Автоматы данного класса взаимодействуют с внешними ячейками многозначной памяти.

На рис. 22 изображена структурная схема для автомата без флагов. При этом поведение автомата зависит только от значений входных переменных `x` и его внутреннего состояния.

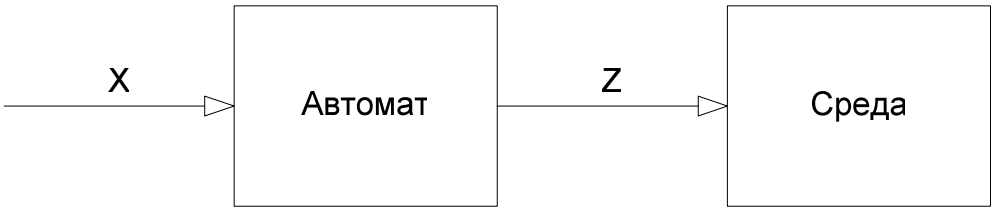


Рис. 22. Структурная схема для автомата без флагов

Из изложенного выше следует, что флаговые переменные одновременно используются и в качестве входных, и в качестве выходных переменных. Вместе с входными переменными и текущим состоянием автомата, они определяют, в какое состояние перейдет автомат в следующий момент времени и какие выходные переменные он сформирует. На рис. 23 изображена структурная схема для автомата с флагами.

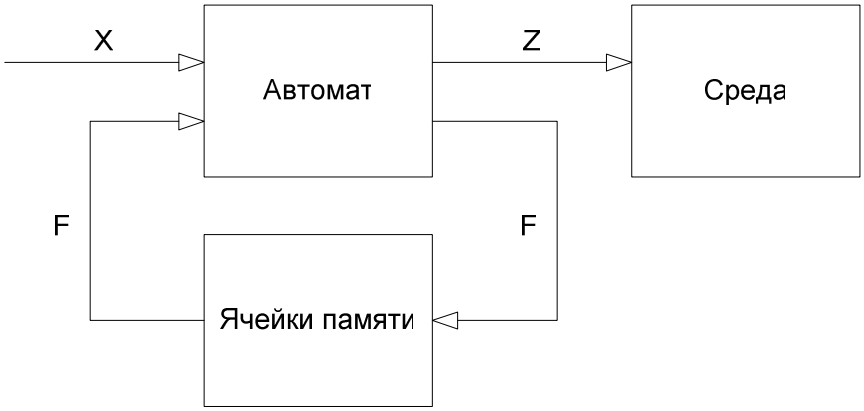


Рис. 23. Структурная схема для автомата с флагами

Как показано в работе [54], использование флаговых переменных часто позволяет уменьшить число состояний в автомате, но при этом становится труднее понять алгоритм работы автомата. Однако, этот недостаток не является существенным для случая автоматической генерации автоматов.

Рассмотрим, как можно использовать автоматы с флагами при решении задачи о флибах [50]. Поведение флиба описывается с помощью таблицы переходов и выходов. В табл. 1 в качестве примера приведен флиб с тремя состояниями A, B и C.

Таблица 1. Таблица переходов и выходов для флиба с тремя состояниями

Состояние	Значение входной переменной			
	0		1	
	Значение выходной переменной	Следующее состояние	Значение выходной переменной	Следующее состояние
A	1	B	0	A
B	0	C	0	A
C	1	A	0	B

На рис. 24 изображен граф переходов для флиба, таблица переходов и выходов для которого приведена в табл. 1.

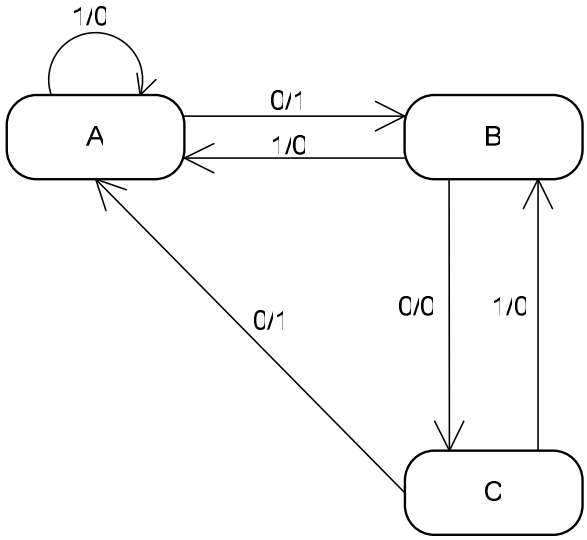


Рис. 24. Граф переходов для флиба с тремя состояниями

Во флибе флаговые переменные можно использовать для хранения предыдущих состояний окружающей среды. Во флаговой переменной с номером один будем хранить состояние среды в предыдущий момент времени, а во флаговой переменной с номером n будем хранить состояние среды на n шагов назад.

Пусть x и $F = (f_1, \ldots, f_n)$ – состояние среды и вектор флаговых переменных в текущий момент времени, соответственно. Тогда вектор флаговых переменных для следующего момента времени будет иметь вид $F' = (f'_1 = x, f'_2 = f_1, \ldots, f'_n = f_{n-1})$. При программной реализации флаговые переменные можно хранить в массиве. В этом случае переход от F к F' осуществляется с помощью удаления последнего элемента массива и вставки в начало массива текущего состояния среды x.

Введение каждой новой флаговой переменной увеличивает число переходов из каждого состояния флиба вдвое. При n флаговых переменных число переходов из каждого состояния флиба будет равно 2^{1+n}. Номер перехода, который должен использоваться в текущий момент времени, можно вычислить по формуле $x + 2^1 f_1 + 2^2 f_2 + \ldots + 2^n f_n$. Также номер перехода может быть выражен числом $f_n \ldots f_2 f_1 x$, записанным в двоичном виде.

Далее флиб, который моделируется с помощью автомата с флагами, будем называть флибом с флагами. В табл. 2 приведен пример таблицы переходов для флиба с тремя состояниями и одной флаговой переменной. Это переменная, сформированная автоматом на предыдущем шаге, на этом шаге используется в качестве одной из входных переменных.

Таблица 2. Таблица переходов и выходов для флиба с тремя состояниями одной флаговой переменной

Состояние	Значение входной переменной *s* и флаговой переменной *f*							
	f = 0, s = 0		f = 0, s = 1		f = 1, s = 0		f = 1, s = 1	
	Значение выходной переменной и следующее состояние		Значение выходной переменной и следующее состояние		Значение выходной переменной и следующее состояние		Значение выходной переменной и следующее состояние	
A	1	A	1	B	0	B	0	A
B	1	B	1	C	0	C	0	A
C	0	B	0	C	1	A	1	B

На рис. 25 изображен граф переходов для флиба, таблица переходов которого приведена в табл. 2.

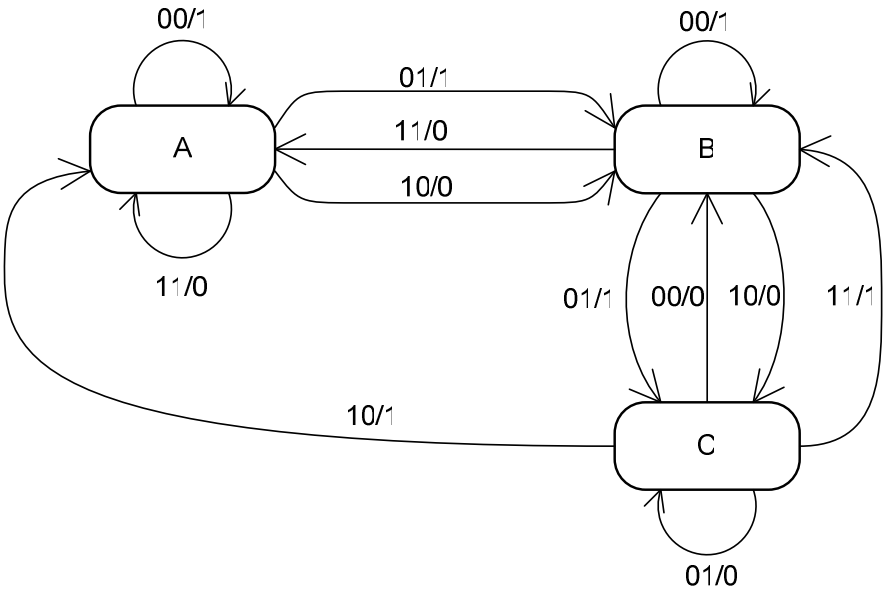


Рис. 25. Граф переходов для флиба с тремя состояниями и одной флаговой переменной

В общем случае, входная переменная может принимать произвольное число значений. Пусть существует m допустимых комбинации значений входных переменных и все эти комбинации пронумерованы от 0 до $m-1$. Тогда при n флаговых переменных число переходов из каждого состояния автомата будет равно m^{1+n}. Номер перехода, который должен использоваться в текущий момент времени, можно вычислить по формуле $x + m^1 f_1 + m^2 f_2 + \ldots + m^n f_n$, где x – номер текущей комбинации значений входных переменных, а $F = (f_1, \ldots, f_n)$ – вектор флаговых переменных.

Флаговые переменные позволяют существенно уменьшить число состояний в автомате. Рассмотрим эту особенность на простом примере. Пусть требуется построить флиб для простейшей среды, заданной с помощью повторяющейся битовой маски: 11100. При использовании обычного автомата понадобится, по крайней мере, три состояния для того, чтобы флиб смог предсказывать изменения среды со сто процентной точностью. Пример такого флиба приведен на рис. 26.

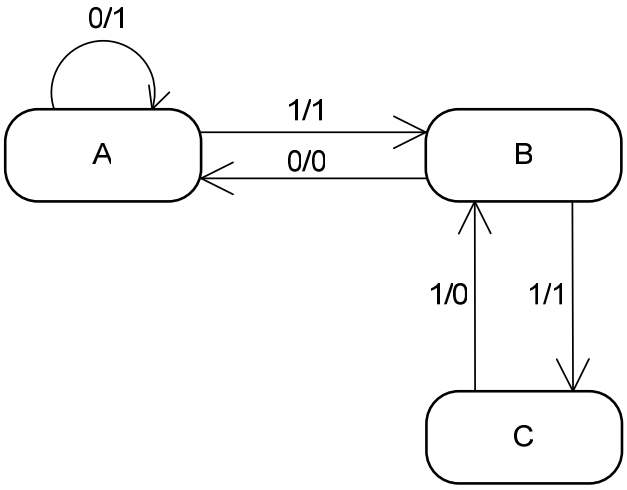


Рис. 26. Граф переходов флиба для среды, заданной битовой маской 11100

Начальное состояние автомата **A**. Для упрощения понимания переходы между состояниями, которые не используются в случае рассматриваемой среды, были удалены.

При использовании двух флаговых переменных достаточно одного состояния, для того, чтобы построить флиб с точностью предсказания равной ста процентам. Граф переходов для такого флиба с флагами изображен на рис. 27. Как и в предыдущем случае, все неиспользуемые переходы были удалены для того, чтобы упростить граф.

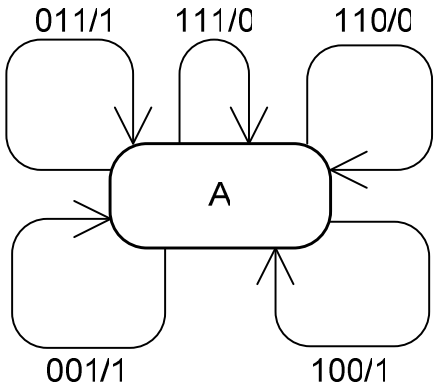


Рис. 27. Граф переходов флиба с двумя флаговыми переменными для среды заданной, битовой маской 11100

Как видно из приведенного примера, использование флаговых переменных позволяет уменьшить число состояний автомата и увеличить число переходов между состояниями.

1.3.2. Алгоритм восстановления связей между состояниями

При генерации генетическим алгоритмом нового поколения решений и применении операторов скрещивания и мутации, переходы в автоматах изменяются случайным образом. При

таком изменении переходов, в автомате, как правило, возникают состояния, в которые невозможно попасть из начального состояния при любой последовательности значений входных переменных. Будем называть такие состояния **недоступными**. Состояния, в которые можно попасть из начального состояния при некоторой последовательности значений входных переменных, будем называть **доступными**. Алгоритм восстановления связей между состояниями изменяет переходы в автомате таким образом, чтобы в нем не было **недоступных состояний**.

Рассмотрим алгоритм восстановления связей между состояниями на примере задачи о флибах. Предлагаемый алгоритм имеет следующий вид.

1. Формируется список **доступных состояний**. Для этого можно, например, использовать функцию рекурсивного обхода графа.
2. Выполняется цикл по всем состояниям. Если текущее состояние не входит в число **доступных состояний**, то для него выполняются следующие операции:
    a. Случайным образом выбирается состояние из списка **доступных состояний**.
    b. Выбирается случайным образом один переход из выбранного состояния.
    c. В текущем состоянии заменяется один переход из этого состояния на переход, который ведет в то же состояние, что и переход, выбранный в пункте b.
    d. Выбранный в пункте b переход заменяется переходом, который ведет в текущее состояние.
    e. Обновляется список **доступных состояний**. В него добавляется текущее состояние и все состояния, в которые можно попасть из него.

Рассмотрим работу описанного выше алгоритма на примере. На рис. 28 изображен граф переходов для флиба, полученного в результате работы генетического алгоритма. Начальное состояние автомата – **A**.

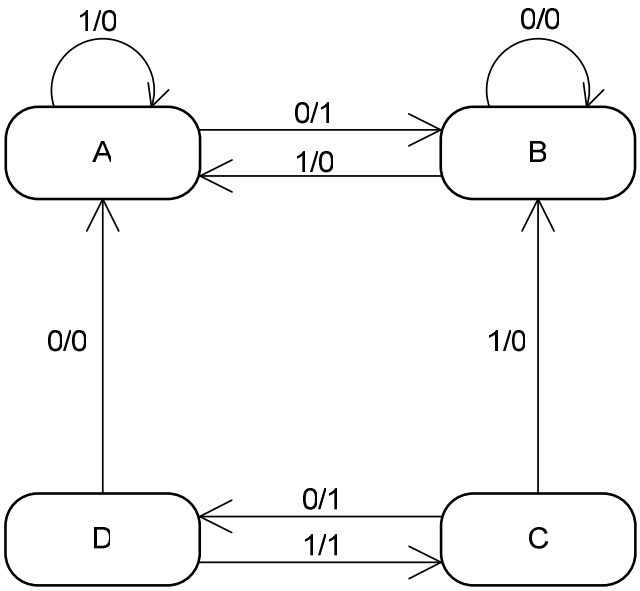


Рис. 28. Граф переходов для флиба, полученного в результате работы генетического алгоритма

Для автомата, граф переходов которого изображен на рис. 28, состояния **A** и **B** являются **доступными состояниями**, в то время, как состояния **C** и **D** – **недоступные**. На рис. 29 **доступные состояния** закрашены светло-серым цветом, а **недоступные** – темно-серыми.

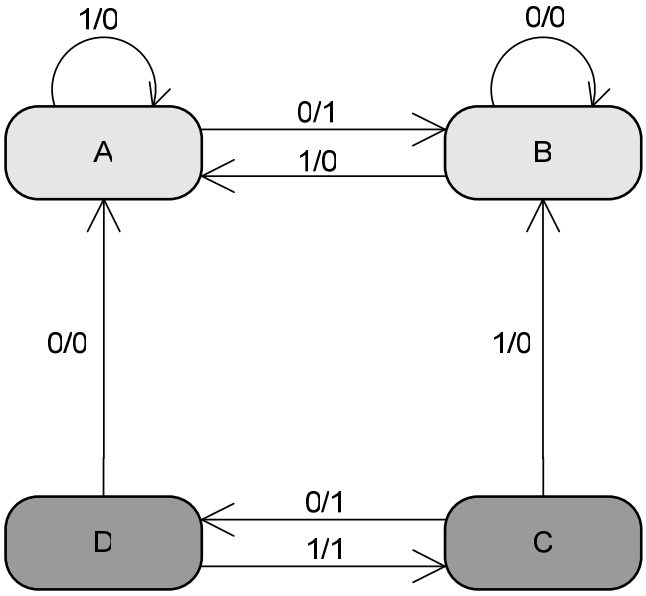


Рис. 29. **Доступные** (светло-серые) и **недоступные** (темно-серые) состояния автомата

Алгоритм восстановления связей между состояниями перебирает все недоступные состояния. Первым рассматривается состояние **C**. Случайным образом выбирается состояние из списка **доступных состояний** (например, состояние **B**). Выбирается случайным образом переход из состояния **B**. В качестве примера выберем переход, который соответствует единичному значению входной переменной и ведет в состояние **A**. Этот переход заменяется переходом, соответствующим такому же значению выходной переменной и ведущим в состояние **C**. Один из переходов, ведущих из состояния **C** (для примера возьмем переход, соответствующий единичному значению входной переменной), заменяется аналогичным переходом, который ведет в состояние **A**.

На рис. 30 показан результат добавления состояния **C** в список **доступных состояний**. Серым пунктиром показаны переходы, которые были удалены в результате этой операции. Черным пунктиром обозначены новые переходы. В состояние **C** автомат попадет из начального состояния **A**, если на его вход, например, будет подана последовательность значений входной переменной вида `01`.

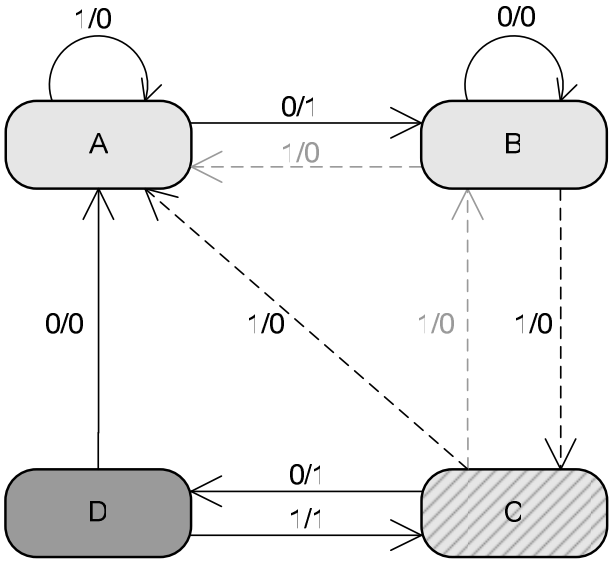


Рис. 30. Добавление состояние **C** в список **доступных состояний**

После замены переходов состояние **C** стало **доступным**. В состояние **D** можно попасть из состояния **C**, следовательно, состояние **D** также стало **доступным**. Действительно, в состояние **D** можно попасть из начального состояния **A**, если подать, например, последовательность значений входной переменной вида `0011`.

Так как после обновления списка **доступных состояний** все состояния стали доступными, то алгоритм завершает свою работу. Граф переходов для автомата, получившегося в результате работы алгоритмам восстановления связей между состояниями, изображен на рис. 31.

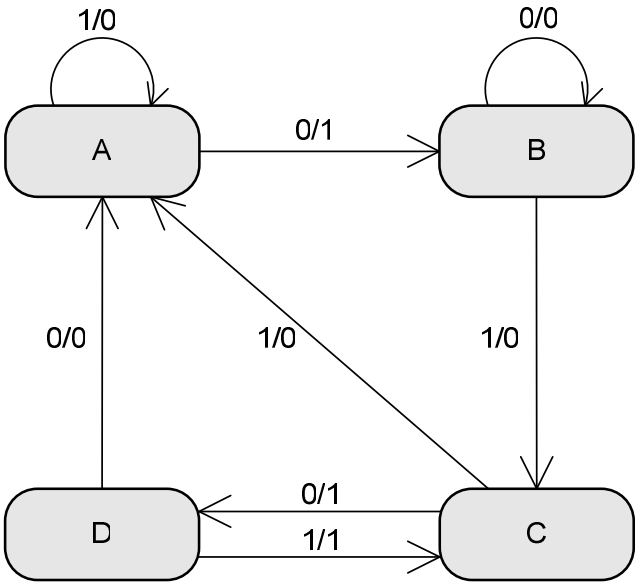


Рис. 31. Результат работы алгоритма восстановления связей между состояниями

Алгоритм восстановления связей между состояниями можно использовать как в качестве дополнительного, так и в качестве основного оператора мутации. Иногда может оказаться целесообразным проверять, не ухудшилась ли приспособленность особи после выполнения восстановления связей между состояниями. Если приспособленность особи ухудшилась, то граф переходов возвращается в исходное состояние.

Часто может оказаться целесообразным увеличить число **доступных состояний** автомата, оставив часть состояний недоступными. Такой подход требует меньше вычислительных ресурсов и вносит меньше изменений в поведение автомата.

1.3.3. Алгоритм сортировки состояний в порядке использования

Во многих задачах в большинстве решений, которые перебирает генетический алгоритм, автоматы в процессе вычисления значения оценочной функции не достигают части состояний. Далее состояния, из которых выполняется переход хотя бы на одном шаге моделирования (вычисления оценочной функции), будем называть **используемыми состояниями**, а все остальные состояния – **неиспользуемыми состояниями**.

Если при вычислении оценочной функции на вход автомата подаются все возможные последовательности входных значений, то **неиспользуемые состояния** и переходы, связанные с ними, не влияют на поведение автомата. Примерами таких задач могут служить задача об умном муравье [53] и задача о флибах [50].

Приведем алгоритм сортировки состояний в порядке их использования:

1. Создается пустой словарь пар номеров: [«старый номер состояния» – «новый номер состояния»].
2. Моделируется работа автомата. Перед каждым переходом выполняем следующее: если в словаре нет пары, в которой первый элемент равен текущему номеру состояния, то в него добавляется пара [текущий номер состояния – количество пар в словаре].

3. Выполняется цикл по всем состояниям автомата. Для каждого состояния: если в словаре нет пары, в которой первый элемент равен номеру состояния, то в него добавляется пара [номер состояния – количество пар в словаре].
4. Согласно словарю, изменяется порядок состояний и номера состояний, в которые ведут переходы.

Состояния, для которых в словарь добавляются пары в пункте 2 – это и есть **используемые состояния**. Состояния, для которых в словарь добавляются пары в пункте 3 – это **неиспользуемые состояния**.

Рассмотрим работу алгоритма сортировки состояний в порядке использования на примере задачи о флибах. Пусть среда задана с помощью повторяющейся битовой маски `11100`. В процессе работы генетического алгоритма был получен флиб, граф переходов для которого изображен на рис. 32.

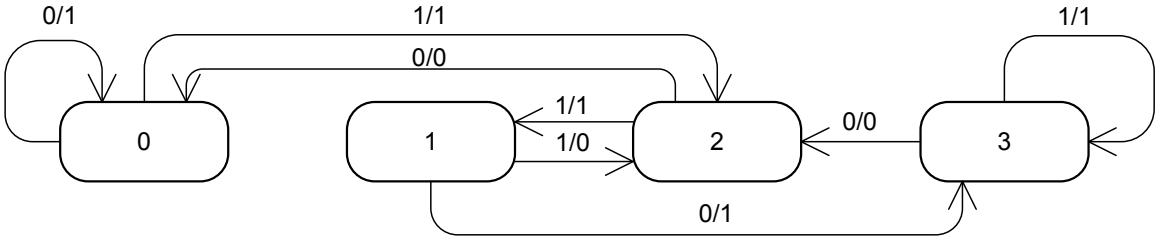


Рис. 32. Граф переходов флиба полученного в процессе работы генетического алгоритма (среда заданна битовой маской 11100)

Состояния флиба пронумерованы цифрами от нуля до трех. Начальное состояние – нулевое.

Создается пустой словарь пар [«старый номер состояния» – «новый номер состояния»].

Начинается моделирование работы флиба. На вход автомату подается **1**. Автомат переходит в состояние **2** и в словарь добавляется пара **[0, 0]**. На следующем шаге моделирования автомат переходит в состояние **1** и в словарь добавляется пара **[2, 1]**. Далее автомат переходит в состояние **2** и в словарь добавляется пара **[1, 2]**. При переходе из состояния **2** в состояние **0** в словарь ничего не добавляется, так как для состояний **2** в словаре уже есть пара.

При дальнейшем моделирование работы флиба, пары в словарь не добавляются, так как при среде, заданной битовой маской `11100` флиб никогда не попадет в состояние **3**. Для всех остальных состояний пары в словаре уже есть.

После моделирования работы флиба словарь пар номеров будет иметь следующий вид:

{[0, 0], [1, 2], [2, 1]}

Выполняется цикл по всем состояниям. Так как в словаре нет пары только для состояний, с первым элементом **3**, то в него добавляется пара **[3, 3]**. Теперь словарь принимает следующий вид:

{[0, 0], [1, 2], [2, 1], [3, 3]}

Осталось изменить порядок состояний и номера состояний в переходах согласно словарю. Состояния **1** и **2** меняются местами и номерами. Граф переходов для флиба, получившегося в результате работы алгоритма, приведен на рис. 33.

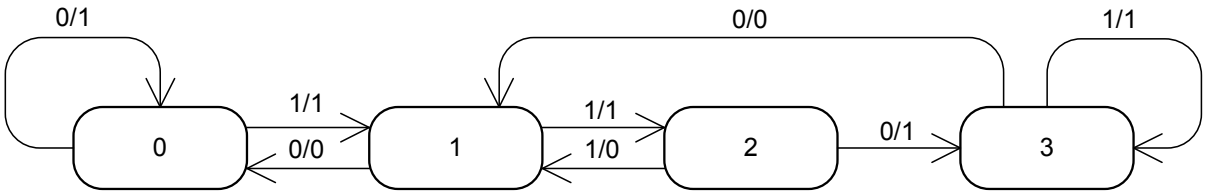


Рис. 33. Граф переходов флиба после применения алгоритма сортировки состояний (среда задана битовой маской 11100)

После применения алгоритма сортировки состояний легко построить автомат, имеющий такое же поведение, как и автомат, найденный с помощью классического генетического алгоритма, но который имеет меньшее число состояний. Для этого в автомате, состояния которого отсортированы в порядке использования, достаточно удалить все **неиспользуемые состояния** и заменить переходы в них переходами в начальное состояние.

В нашем случае одно **неиспользуемое состояние** – **3**. В состояние **3** ведет только один переход из состояния **2**. Удалим состояние **3** и заменим переход в него на переход в состояние **0**. Результат изображен на рис. 34.

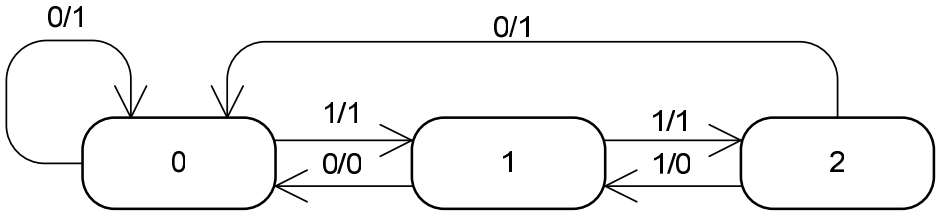


Рис. 34. Граф переходов флиба после удаления **неиспользуемых состояний** (среда задана битовой маской 11100)

Так как при моделировании работы автомата переход из состояния **2,** соответствующий значению входной переменной **0,** ни разу не выполнялся, то поведение автомата не изменилось.

1.4. Композитное генетическое программирование

Как правило, при решении оптимизационной задачи при помощи эволюционных алгоритмов существует множество подходов, каждый из которых характеризуется представлением решения в виде хромосомы, заданием функции приспособленности, определением используемых эволюционных алгоритмов, их параметров, размером популяции. «Угадывание» подхода, как можно быстрее приводящего к желаемому результату, является эвристической задачей. Представляется маловероятным, что эта задача может быть существенно автоматизирована. Из работ в этом направлении можно выделить лишь, так называемое, «метагенетическое программирование» [17], позволяющее параметрам генетических операции (например, таким как вероятность мутации или способ селекции) эволюционировать вместе с популяцией решений. Справедливо также, что различные подходы решения оптимизационной задачи, основанные на эволюционных алгоритмах, как правило, различаются по следующим показателям:

- выразительностью выводимых решений;
- скоростью поиска;

- множеством локальных экстремумов, к которым наиболее вероятно сходится процесс эволюции.

Выразительность выводимых решений определяется способом представления решения оптимизационной задачи в виде хромосомы. Например, при поиске максимума некоторой вещественной функции, определенной на множестве R^2, можно осуществлять поиск не на плоскости, а на прямой, зафиксировав значение одной из координат. На рис. 35 приведен график некоторой функции, имеющий несколько экстремумов. При фиксированной координате пространство поиска сокращается до R – на рис. 35 ему соответствует прямая линия.

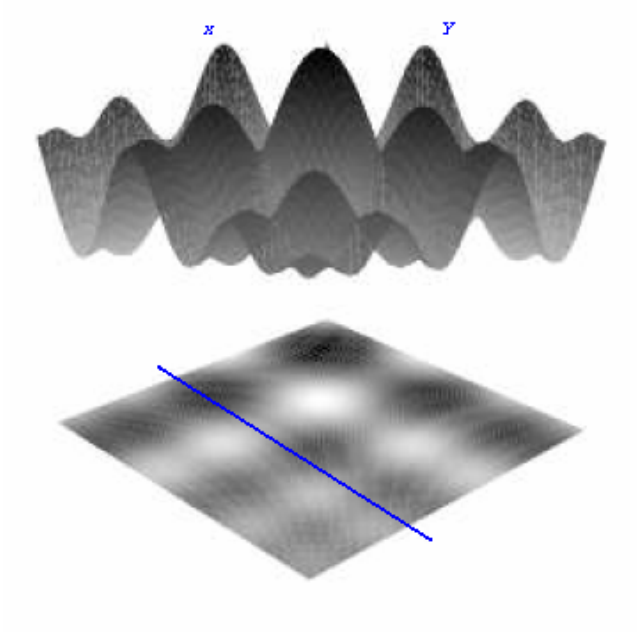


Рис. 35. Поиск экстремума

В общем случае, поиск, организованный таким образом, будет выполняться значительно быстрее. Однако максимум на выбранной прямой не будет являться максимумом на плоскости. При этом если передать алгоритму поиска максимума на плоскости найденное решение на прямой, то это решение может быть в дальнейшем улучшено (уже как элемент R^2). Время работы такого «комбинированного алгоритмы поиска» зачастую оказывается меньше времени работы алгоритма поиска на множестве R^2.

Другой особенностью при выборе эволюционного алгоритма является то обстоятельство, что выбранный алгоритм, часто приводит популяцию в направлении одних экстремумов с большей вероятностью, чем в направлении других. Например, в пространстве поиска существует два экстремума x и y, а для решения задачи применяются два эволюционных алгоритма X и Y. Первый из них с бо́льшей вероятностью приводит популяцию в окрестность экстремума x, а второй — в окрестность y (рис. 35). Однако алгоритму X значительно сложнее (уже находясь в окрестности экстремуму) достичь значения x, чем алгоритму Y, если в его популяцию добавить решения из окрестности x.

Таким образом, взаимодействие разных эволюционных алгоритмов состоит в том, что хорошие решения, полученные одним из алгоритмов, могут быть улучшены другим алгоритмом. Это может приводить к увеличению скорости улучшения решений. Кроме того, такое взаимодействие

может обеспечить получение решений, которые не могли быть найдены ни одним из алгоритмов в отдельности.

1.4.1. Идея метода композитного генетического программирования

Метод композитного генетического программирования направлен на использование указанных особенностей. Цель этого метода – увеличить скорость и улучшить качество решений оптимизационной задачи, позволив различным эволюционным алгоритмам взаимодействовать между собой. В качестве примера опишем предлагаемый метод для решения задачи классификации плотности для одномерных бинарных клеточных автоматов [45]. Кроме того, как ниже будет показано, можно применить предлагаемый метод для решения модифицированной задачи об “Умном муравье” (постановка классической задачи об “Умном муравье” приведена в разд. 1.4.1.1 отчета по первому этапу темы [51]).

1.4.1.1. Задача классификации плотности

Введем определения, необходимые для постановки задачи классификации плотности, а затем выполним постановку этой задачи.

Одномерный детерминированный клеточный автомат (однородная одномерная детерминированная среда) – это специального вида оператор $P: \mathrm{X}_n \to \mathrm{X}_n$. Этот оператор действует на пространстве $X_n = S_n^{\mathrm{Z}} = \{x\},\ x = (x_i),\ x_i \in S_n = \{0,\ldots,n\},\ i \in \mathrm{Z}$, которое состоит из всех бесконечных в обе стороны последовательностей, члены которых – целые числа от 0 до *n*: Автомат *P*, действующий на множестве X_n, задается радиусом *r*, $r \in \mathrm{Z}^+$ и функцией $f: S_n^{2r+1} \to S_n$ и имеет следующий вид. Для любого $x \in \mathrm{X}_n$ *i*-я компонента $(Px)_i = f(x_{i-r},\ldots,x_{i+r})$.

Рассмотрим некоторый детерминированный одномерный бинарный клеточный автомат P' радиуса r', заданный функцией f'. Согласно определению, данному выше, такой автомат действует на множестве 2^Z. Для некоторого натурального числа *L,* где $L > r'$, существует сужение *P* оператора P' с множества 2^Z на множество 2^L, задаваемое радиусом $r = r'$, функцией $f \equiv f'$, такое что $(Px)_i = f(x_{(i-r+L)\div L}, x_{(i-r+1+L)\div L}, \ldots, x_{(i-r+k+L)\div L}, \ldots, x_i, \ldots, x_{(i+r-k)\div L}, \ldots, x_{(i+r-1)\div L}, x_{(i+r)\div L})$.

Менее формально: возьмем отрезок длины *L*, «свернем» его в окружность и зададим на этой окружности оператор *P*.

Зададим вещественное число $\rho^* \in [0,1]$. Определим вещественную функцию $\rho(x): 2^L \to [0,1]$ как «плотность конфигурации» $x \in 2^L$ — отношение числа единиц в ней к *L*. Будем говорить, что клеточный автомат *P распознает (классифицирует)* конфигурацию *x,* если $\exists n: P^{(n)}x = s$, где *s* — стационарное состояние:

- «все нули» (0^L), если $\rho(x) < \rho^*$;
- «все единицы» (1^L), в противном случае: $\rho(x) \geq \rho^*$.

Рис. 36 иллюстрирует данное определение.

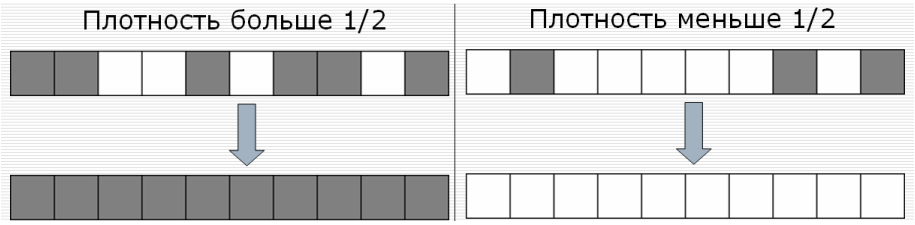


Рис. 36. Клеточный автомат распознает конфигурацию, где:
$\rho > \frac{1}{2}$ (слева) и $\rho < \frac{1}{2}$ (справа)

Зададим индикаторную функцию $I_{\rho^*}(P,x): 2^{2^{2r+1}} \times 2^L \to \{0,1\}$, принимающую единичное значение, если ли оператор P (заданный радиусом r и функцией $f: 2^{2r+1} \to \{0,1\}$) распознает конфигурацию x, и значение ноль в противном случае. Зафиксировав L и r, определим функцию $g_{L,r,\rho^*}: 2^{2r+1} \to [0,1] \subset \Re$; $g_{L,r,\rho^*}(f \in 2^{2r+1}) = \dfrac{\sum_{x \in 2^L} I_{\rho^*}(f,x)}{2^L}$.

Постановка задачи классификации плотности: при заданных r, L и ρ^* требуется найти $f_{\max} \in 2^{2^{2r+1}}$, на которой достигается максимум функции g_{L,r,ρ^*}, то есть $\max\limits_{f \in 2^{2^{2r+1}}} g_{L,r,\rho^*}(f) = g_{L,r,\rho^*}(f_{\max})$.

Другими словами, при заданном радиусе автомата и длине окружности требуется найти правило, задающее клеточный автомат, распознающий как можно большее число конфигураций. Будем говорить, что функция (правило) $f' \in 2^{2^{2r+1}}$ *полностью (совершенно) решает задачу классификации плотности (Density Classification Task – DCT*), если $g_{L,r,\rho^*}(f') = 1$.

В неформальной постановке задача может быть сформулирована так: *требуется найти правило заданного радиуса, классифицирующее наибольшее количество начальных конфигураций заданной длины.* На рис. 37 показан процесс распознавания автоматом начальной конфигурации, содержащей больше нулей, чем единиц – плотности меньшей 0.5.

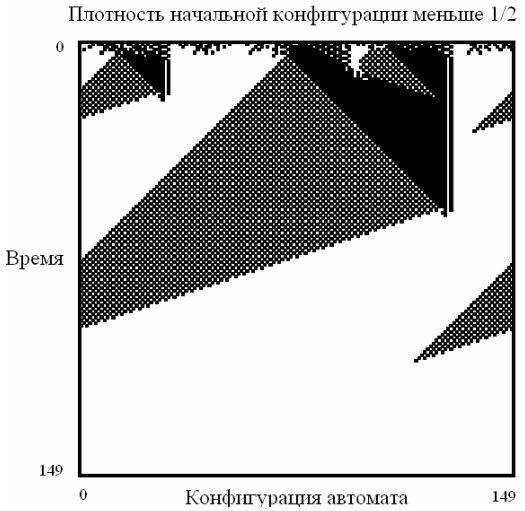


Рис. 37. Распознавания автоматом начальной конфигурации, $\rho < 1/2$

1.4.1.2. Постановка проблемы, возникающей при решении задачи классификации плотности с помощью эволюционных алгоритмов

При анализе работы программ, реализующих эволюционные алгоритмы (стандартные генетические алгоритмы (*Conventional Genetic Algorithms – GA*) [34], генетическое программирование (*Genetic Programming – GP*) [15] и программирование с экспрессией генов (*Gene Expression Programming – GEP*) [16]), была выявлена **следующая проблема:**

- при “слишком сложной” структуре хромосом, реализованной в алгоритме *GA* (в виде столбца значений таблицы истинности, представленного битовой строкой длины 128), а также в алгоритме *GP* (в виде деревьев с 26 функциями и 26 терминалами) бо́льшая часть вычислений проходит впустую, и крайне медленно появляются хорошие решения. В то же время наличие хороших начальных приближений крайне важно для эффективной работы эволюционных алгоритмов;
- при “слишком простой” структуре хромосом, реализованной в алгоритме *GEP*, очень быстро находятся хорошие начальные приближения, однако они не могут быть улучшены в силу недостаточной выразительности хромосом. Так, по-видимому, не существует простой булевой формулы (решения задачи классификации плотности), которая классифицирует более *83%* конфигураций.

Для решения поставленной проблемы предлагается простой, но весьма *эффективный* метод – “Композитное генетическое программирование”. Помимо решения указанной проблемы, композитное генетическое программирование обладает рядом других достоинств.

1.4.2. Описание метода

Основная идея метода композитного генетического программирования – позволить различным эволюционным алгоритмам, решающим одну и ту же оптимизационную задачу на множестве автоматов, взаимодействовать между собой – одни алгоритмы быстро генерируют хорошие решения, а другие в дальнейшем их улучшают. Для реализации данной идеи предлагается создать многокомпонентную систему с *репозиторием*, находящимся на сервере, в котором хранятся решения задачи в *универсальной форме,* оцениваемые *универсальной функцией приспособленности.* Использование универсальной формы обеспечивает взаимодействие алгоритмов между собой, а универсальная функция предназначена для того, чтобы существовала общая для всех алгоритмов оценка решений. Применительно к *DCT*:

- универсальная форма решений – столбец значений таблицы истинности правил (битовая строка фиксированной длины);
- универсальная функция – вероятность распознавания данным правилом случайной тестовой конфигурации, вычисляемая статистически на большо́м наборе тестовых конфигураций.

В общем случае (для других задач) необходимо выбрать некоторую универсальную форму представления автоматов.

Клиентские программы, реализующие различные методы решения задачи, представлены отдельными *модулями* (*GEP*-модуль, *GA*-модуль, *GP*-модуль и т.д.) Они периодически отдают серверу наилучшие из своих решений и запрашивают решения из репозитория по определенному *критерию* (например, чуть лучшие, чем их собственные). Форма представления решений на клиентских модулях, как правило, существенно отличается от *универсальной формы*, но должна быть к ней приводима – существует некоторая *функция преобразования решений*, отображающая решения различных *модулей* в *универсальную форму*. Пример такой функции для модуля *GEP* будет приведен ниже. Желательно, чтобы эта функция была обратима и “быстро” вычислима (например, линейно по длине представления решения). *Универсальная функция приспособленности* – целевая функция

решаемой оптимизационной задачи. Она также может существенно отличаться от функций приспособленности модулей.

На рис. 38 приведена схема композитного генетического программирования для задачи *DCT*.

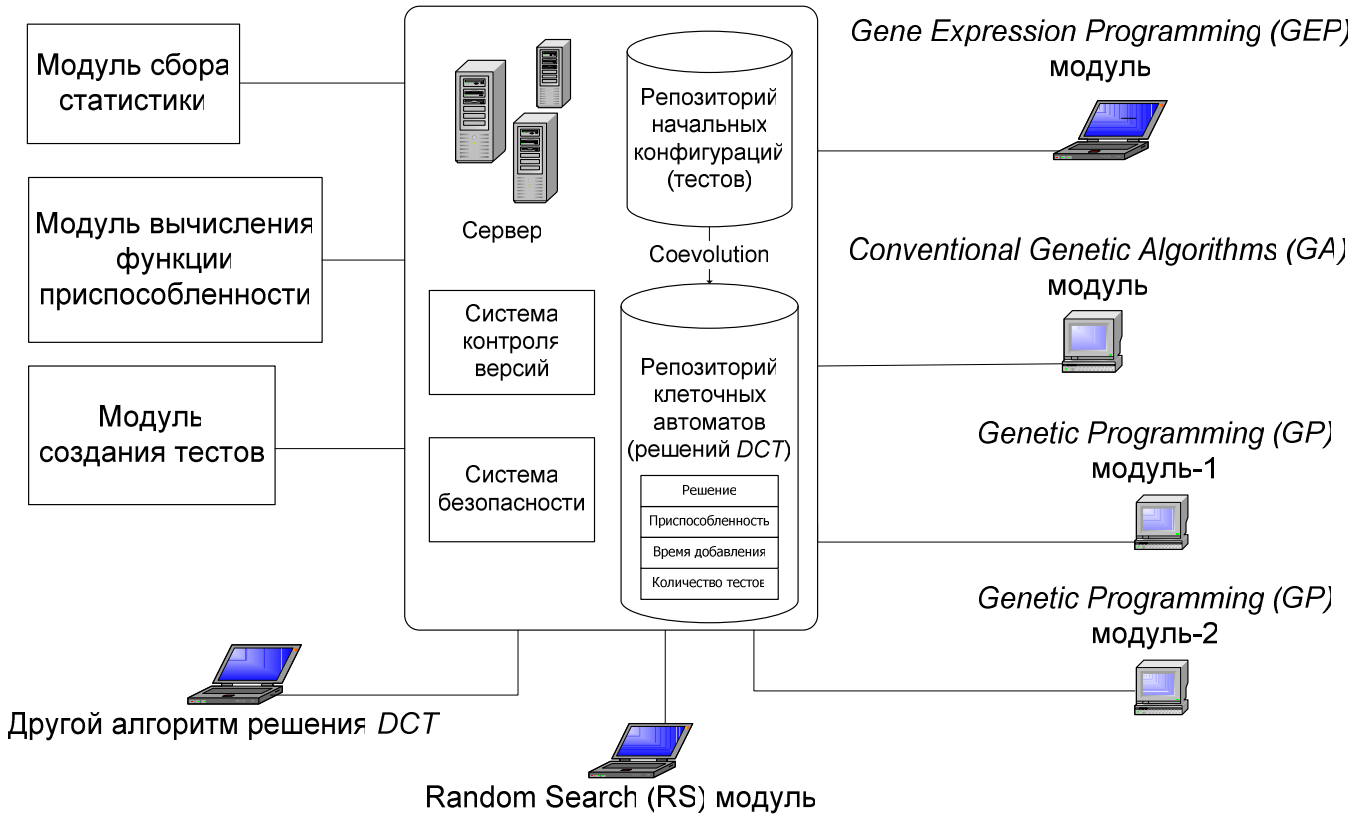


Рис. 38. Схема композитного генетического программирования

Модуль с простым представлением решений через *репозиторий* передает быстро найденные хорошие решения другим *модулям*, обладающим более выразительным представлением решений, которые способны улучшить любые хорошие решения, но не могут быстро их найти. Так, в *DCT* метод *GEP* с простым представлением хромосом позволяет быстро находить хорошие решения и передавать их методам *GP* или *GA* для дальнейшего улучшения. Самостоятельно методы *GP* и *GA* не способны быстро находить хорошие решения.

Репозиторий выступает и как надежное централизованное хранилище найденных решений для большой и сложной распределенной задачи, и как основа системы контроля версий. К нему подключаются *модули* и “сохраняют” найденные ими решения (по своему идентификатору), которые позже ими (по этому же идентификатору) полностью или частично загружаются.

Предлагаемый метод позволяет “объединить” для решения одной задачи различные, как по способу представления хромосом, так и по способу задания функции приспособленности алгоритмы – *модули*. Применительно к *DCT*:

- разный способ задания хромосом имеют все три алгоритма: *GA* – строки фиксированной длины, *GP* – деревья, *GEP* – *Karva*-деревья (линеаризованные деревья);

- функции приспособленности также различны. Например, при использовании коэволюции (подхода, описанного в работах [14, 23, 35]) функция приспособленности изменяется вместе с популяцией начальных конфигураций в процессе эволюции. Без коэволюции также существует несколько способов ее задания: процент распознанных тестовых конфигураций, сгенерированных один раз в начале работы алгоритма, или процент распознанных тестовых конфигураций, генерируемых на каждом шаге эволюции. Кроме того, тестовые конфигурации можно генерировать различными “случайными” способами – равномерно над каждым битом или равномерно над плотностью конфигурации.

Как отмечалось выше, поскольку в общем случае решения в репозитории представляются в ином виде, нежели в модуле, необходимо существование функции преобразования решений. Эта функция вычисляется на сервере. Так решения *DCT*, полученные методами *GP* (деревья) и *GEP* (*Karva*-деревья), преобразуются в столбец значений таблицы истинности правил (битовые строки фиксированной длины). Преобразование здесь выполняется вычислением значения булевой функции, представленной деревом на всевозможных значениях переменных $x_{-r},...,x_r; x_i \in \{0, 1\}$. Например, так называемое *GKL*-правило $IF(x_0, M(x_0, x_{-1}, x_{-3}), M(x_0, x_1, x_3)$ [48], полученное алгоритмом *GEP*, преобразуется в битовую строку:

```
00000000010111110000000001011111000000000101111100000000001011111
00000000010111111111111101011111000000000101111111111111101011111
```

1.5. Выводы

Предложенные методы позволяют существенно сократить размерность пространства поиска оптимальных решений для задач генерации автоматов. Таким образом, они должны позволить решать такие задачи более эффективно. Отметим, что методы сокращенных таблиц переходов и представления автоматов деревьями решений также позволяют более тонко выбирать методы мутации, что должно ускорить получение оптимальных особей.

Для разработанных методов требуется провести анализ функциональных особенностей и характеристик, а также условий и ограничений функционирования. Это позволит определить их области применимости. Для исследования эффективности предложенных методов на практике необходимо создать прототипы программных средств, основанные на них, и провести соответствующие эксперименты.

2. ФУНКЦИОНАЛЬНЫЕ ОСОБЕННОСТИ И ХАРАКТЕРИСТИКИ МЕТОДОВ ГЕНЕРАЦИИ АВТОМАТОВ

В данной главе рассматриваются вопросы применимости разработанных методов генерации автоматов управления системами со сложным поведением, в частности, определяются их функциональные особенности (разд. 2.1), характеристики (разд. 2.2), а также условия и ограничения функционирования (разд. 2.3).

2.1. Функциональные особенности методов генерации автоматов

В данном разделе рассматриваются функциональные особенности методов генерации автоматов, разработанных в главе 1.

2.1.1. Метод сокращенных таблиц переходов

Метод основывается на предположении возможности выбора перехода из состояния автомата на основе анализа ограниченного числа предикатов. При этом набор таких предикатов для каждого состояния задается подмножеством всех предикатов. Таким образом, все выбранные для анализа предикаты считаются равно значимыми, и рассматриваются все возможные их комбинации. Число вариантов переходов зависит только от числа значимых предикатов. Например, два перехода из состояния 1 на рис. 39, как и в общем случае анализа трех предикатов будут, представлены восемью переходами, как показано на рис. 40.

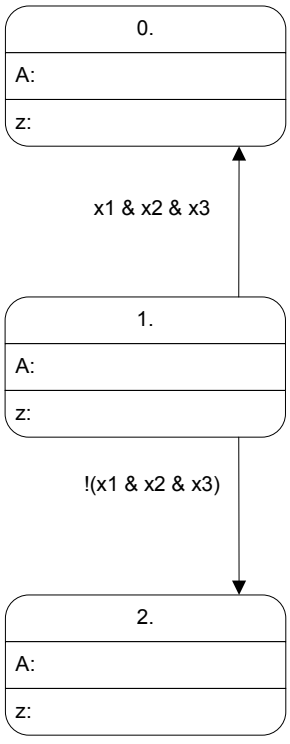


Рис. 39. Два перехода из состояния 1 при описании условия перехода формулой

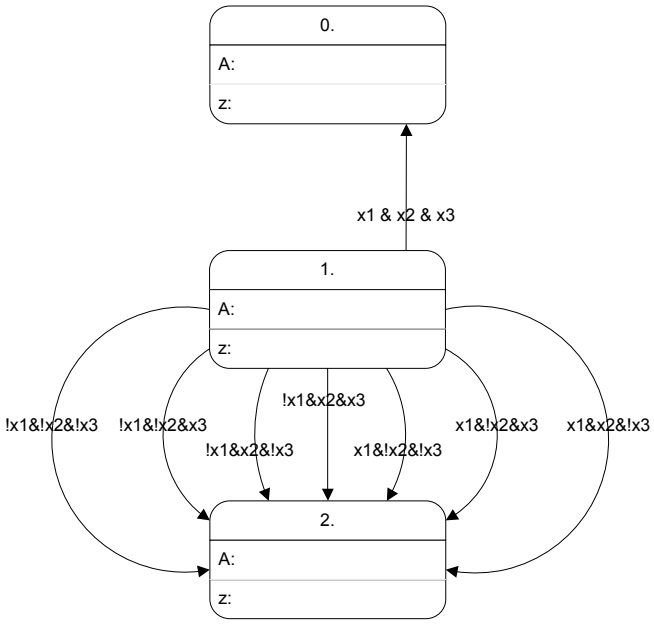


Рис. 40. Восемь переходов в представлении сокращенных таблиц, соответствующие двум описанным формулами переходам

Такой, избыточный для задач с большим числом значимых в состоянии предикатовЮ, и простой логикой подход хорошо подходит для реализации сложной логики, стимулируя учет специальных случаев. Так, если в приведенном примере окажется, что, скажем, случай !x1&!x2&!x3 требует особой обработки, то при описании таблицей это, скорее всего, будет обнаружено. Фрагмент автомата, изображенный на рис. 41, не будет анализироваться наравне с фрагментом из рис. 40, а при описании формулой вероятность вывода фрагмента автомата, изображенного на рис. 42, представляется значительно меньшей, чем для рис. 39.

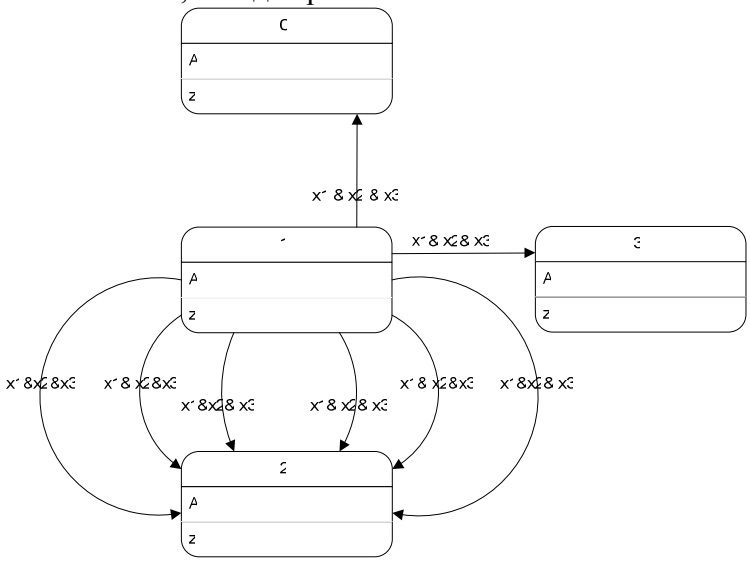


Рис. 41. Обработка специального случая при табличном представлении

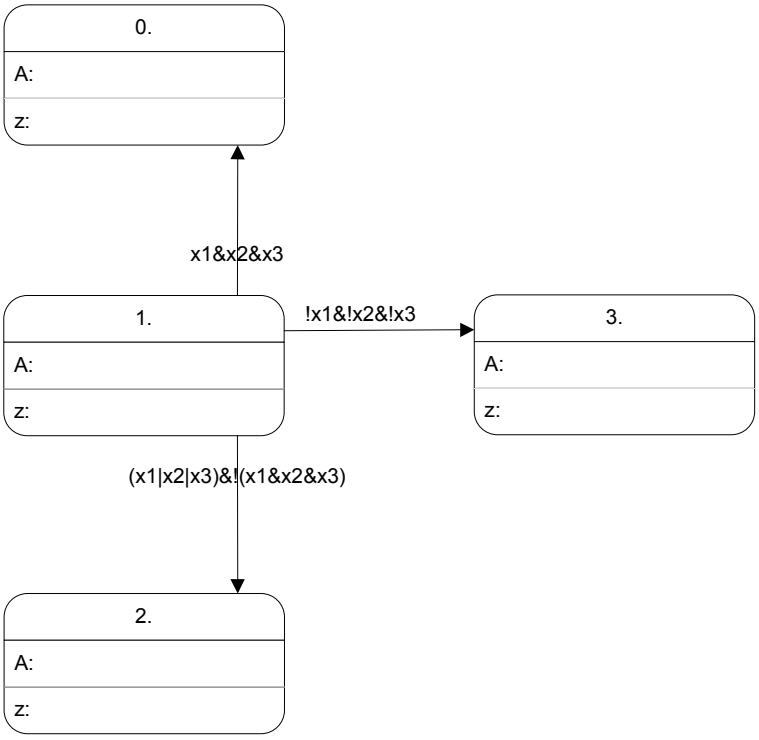


Рис. 42. Обработка специального случая при описании условий формулами

2.1.2. Метод представления автоматов деревьями решений

Разработанный метод обладает следующими особенностями:

1. Метод предназначен для генерации одного автомата. Генерация системы связанных или вложенных автоматов не поддерживается.
2. Метод предназначен для генерации автоматов с дискретными входными переменными. Недискретные входные переменные не поддерживаются.
3. Метод может быть применен как для генерации автоматов Мили, так и для генерации автоматов Мура.
4. Метод использует представление автоматов высокого уровня абстракции.

К достоинствам изложенного метода относятся:

1. Уменьшение длины хромосомы.
2. Естественность представления автомата для человека.
3. Возможность наложения ограничений на генерируемый автомат.

Уменьшение длины хромосомы осуществляется за счет большей выразительности деревьев решений по сравнению с таблицами переходов. Это отражается в уменьшении необходимого объема памяти для хранения хромосомы, что позволяет увеличить популяцию, а также ускоряет вывод. Отметим, что для функций, плохо представимых с помощью деревьев решений, необходимый объем памяти наоборот возрастает.

Основной выигрыш в производительности происходит за счет уменьшения длины хромосомы. Результаты экспериментов (разд. 3.2.2) демонстрируют, что в случаях, когда выводились деревья решений малой глубины, предлагаемый метод позволил найти лучшие решения, чем генетические

алгоритмы, оперирующие с полной таблицей переходов. Когда же выводились деревья решений большой глубины, тогда выигрыш получить не удавалось.

Отметим, что уменьшение длины хромосомы ведет также и к ускорению генетических операций. Несмотря на то, что измерение фитнесс-функции, как правило, является наиболее затратным шагом генетических алгоритмов, с увеличением числа предикатов время совершения генетических операций может также стать существенным.

Естественность представления автомата для человека вытекает из упрощения представления функций с помощью деревьев решений по сравнению с полной таблицей переходов.

Возможность наложения ограничений на генерируемый автомат представляется довольно значительным преимуществом, так как позволяет сузить область поиска, тем самым ускоряя генерацию автоматов. Кроме того, часто требуется найти не оптимальное решение, а достаточно простое хорошее решение, что может быть осуществлено с помощью задания ограничений на целевой автомат.

Основными недостатками использования изложенного представления автоматов являются:

1. Возможность нескольких представлений одного и того же автомата.
2. Потеря части генов родителя при выполнении операции обрезки.

Ниже приводятся возможные способы нейтрализации указанных недостатков.

Отметим, что **возможность нескольких представлений** является общей особенностью представления хромосом на достаточно высоком уровне абстракции, например, она также имеется при представлении автоматов с помощью графов или сокращенных таблиц. При этом неоднозначность представления обуславливается следующими причинами:

- неоднозначностью нумерации состояний;
- неоднозначностью представления функции переходов с помощью деревьев решений.

При использовании разработанного подхода в качестве модуля *композитного генетического программирования* указанная особенность может вести к тому, что представление автомата будет утрачено. Отметим, что для этого представления автоматов важно не только поведение автомата и его полная таблица переходов, но и конкретная структура деревьев решений, так как скрещиваются деревья решений, а не таблицы переходов.

Классическим способом устранения неоднозначности нумерации состояний является эвристика *MTF (Move To Front)* [10]. Она заключается в перенумерации состояний в порядке обхода графа переходов автомата в ширину. Особенностью применения этой эвристики к рассматриваемому методу является то, что необходимо определить порядок переходов из вершины. Это можно сделать, например, следующим образом — найти для каждого перехода лексикографически минимальный набор значений входных переменных, соответствующий этому переходу. После этого рассматривать переходы в порядке лексикографического возрастания соответствующего набора. Рассмотрим путь от корня до листа дерева, соответствующего переходу. Значения всех переменных, встретившихся на этом пути, фиксированы для всех наборов, соответствующих переходу. Остальные переменные могут принимать любые значения. Лексикографически минимальным из подходящих наборов будет набор, в котором все переменные, значения которых не фиксированы, принимают минимальное значение.

Неоднозначность представления функции переходов может быть устранена любым алгоримом построения дерева решений по набору значений функции. Важно отметить, что при этом частое перестроение деревьев может сыграть отрицательную роль, так как при этом теряется структура деревьев, которая влияет на генетические операции.

Потеря части генов родителя при операции обрезки ведет к тому, что хорошие поддеревья при копировании в потомков постепенно вырождаются. Однако, это представляется меньшей проблемой, чем появление генетического мусора — излишней информации. Без операции обрезки

наоборот, плохие (недостижимые) поддеревья могут беспрепятственно копироваться в потомков, так как никак не влияют на фитнесс-функцию.

Одним из способов решения данной проблемы является применение модификации эвристик «сжатия» и «расширения» [2]. Наиболее естественным преобразованием «сжатия» является запрет на изменение одного из поддеревьев. После этого генетические операции не могут изменять «сжатое» поддерево. Преобразование «расширения» в таком случае логично сделать обратным операции «сжатия» — разрешить его применение только к «сжатым» поддеревьям. Модификация заключается в добавлении дополнительного требования — запрете обрезки «сжатых» поддеревьев, что приведет к сохранению хороших генов.

2.1.3. Методы оптимизации генетических алгоритмов для построения автоматов

Использование автоматов с флагами вместо обычных автоматов позволяет за счет увеличения числа переходов уменьшить число состояний. Благодаря этому в автоматах уменьшается число **недоступных состояний**. Поведение автоматов с флагами меньше изменяется при использовании оператора мутации, чем поведение автоматов без флагов.

Использование алгоритма восстановления связей между состояниями позволяет увеличить среднее число состояний в автоматах поколения. Чем больше состояний у автомата, тем меньшему числу изменений он подвергается и, как следствие, тем меньше изменяется его поведение. В итоге генетический алгоритм начинает отдавать предпочтение автоматам с большим числом состояний и соответственно с более сложным поведением. Это полезно если приближенным решением задачи является автомат с небольшим числом состояний, а любое лучшее решение должно иметь значительно большее число состояний. В этом случае генетический алгоритм легко может застрять в локальном оптимуме. Эта проблема возникает из-за того, что стандартный оператор мутации (изменяет значение выходной переменной на переходе или состояние, в которое осуществляется переход) при однократном применении крайне редко может значительно увеличить число состояний в автомате. Алгоритм восстановления связей между состояниями решает эту проблему.

Алгоритм сортировки состояний в порядке использования приводит к одному виду автоматы с одинаковым поведением, но разным порядком нумерации состояний. Например, поведение автомата, граф переходов для которого приведен на рис. 43, аналогично поведению автомата приведенного на рис. 24. Однако генетический алгоритм будет считать эти автоматы разными решениями. Применение алгоритма сортировки состояний в порядке использования приведет оба автомата к одному виду.

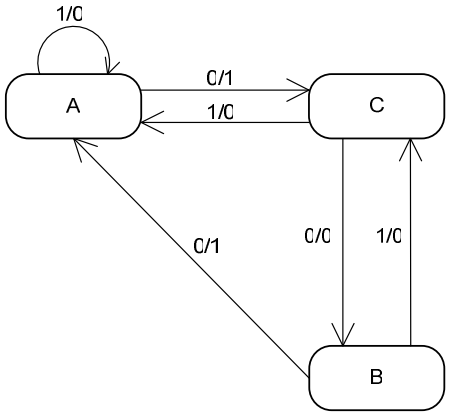


Рис. 43. Граф переходов автомата с тремя состояниями

Таким образом, использование алгоритма сортировки состояний автомата в порядке использования позволяет сократить пространства поиска, в котором генетический алгоритм перебирает решения.

2.1.4. Метод композитного генетического программирования

Метод поддерживает *коэволюции*, преимущества которых описаны в работах [14, 23, 35]. В этом случае на сервере находятся сразу несколько *репозиториев*: решений, тестов для решений, тестов для тестов и т. д. Так в *DCT* имеются два *репозитория* (рис. 38): решений (правил клеточных автоматов) и тестов (начальных конфигураций, на которых вычисляется приспособленность правил). При этом существует разделение ролей по *модулям* – одни *модули* специализируются на поиске решений, другие – на составлении тестов к решениям, а третьи – на вычислении *универсальной функции приспособленности* на основе решений и тестов, найденных другими *модулями*.

В подходе, предложенном в работе [23] и реализованном в работе [45], важную роль в вычислении функции приспособленности тестовых конфигураций играет значение статистически вычисляемой формулы $E_{i,j} = (R_i, \rho(IC_j)) = \ln(2) + p\ln(p) + q\ln(q);\ 0 < p, q \le 1$, которая при $p = 0,\ q = 0$ приобретает вид: $E_{i,j} = \ln(2)$. В исходной формуле R_i — правило, IC_j — начальная конфигурация, q – вероятность (вычисляемая статистически) того, что правило R_i распознает конфигурацию с плотностью $\rho(IC_j)$, $p = 1 - q$. Статистика отражает “покрытие решений тестами” – показывает для каких решений существуют хорошие тесты, а для каких их не существует, тем самым помогая оценивать “качество тестов”. Напомним, что на множестве тестовых конфигураций задана *функция приспособленности*, статистика E – участвует в задании этой функции. Такого типа статистики собираются отдельными *модулями* (рис. 38) на основе репозиториев решений и тестов, и сохраняются в репозитории. Затем модули, запрашивая из репозитория решения и тесты, получают также и относящиеся к ним статистики, и не тратят время на самостоятельное их вычисление. При наличии таких статистик отдельный *модуль*, предоставляя серверу информацию о своих решениях, имеет возможность запросить у него “хорошие” тесты для этих решений.

К решениям, добавляемым в *репозиторий,* приписываются метаданные (атрибуты): каким *модулем* данное решение добавлено в репозиторий, дата и время добавления, значение функции приспособленности для этих решений, число тестов, использованных в вычислении функции (рис. 38). За счет этого реализуются сразу несколько возможностей:

- сбор статистики по таким показателям как *показатель эволюции*, определенный как скорость роста функции приспособленности на лучших решениях *репозитория*, разнородность решений *репозитория*, средняя плотность правил (хорошие правила обладают плотностью, близкой к 0.5), средняя плотность тестовых конфигурации (чем больше этот показатель, тем сложнее набор тестовых конфигураций), число беспорядков правила (чем оно меньше, тем “монотоннее” правило и проще формула, его задающая). На основе статистики делаются заключения о том, происходит ли постепенная максимизация универсальной функции приспособленности, как быстро происходит максимизация, какие *модули* вносят наибольший вклад, какие *модули* не предлагают хороших решений и могут быть отключены с оповещением о нерезультативности. Статистика собирается как непосредственно на сервере, так и отдельными клиентскими *модулями,* специализирующимися именно на ее сборе, а не на реализации эволюционных алгоритмов*;*
- в общем случае, основываясь на дате добавления решения, организуется *система контроля версий*, позволяющая сравнивать репозитории на различные даты, “*откаты-*

вать" *(rollback)* на конкретную дату и просматривать историю *"добавления решений" (commit)* различными модулями. Операция "отката" требуется, так как если в начале *универсальная функция приспособленности* была определена не точно, то можно потерять хорошие решения. Еще один пример, при котором необходим откат – это сбой в работе системы, который произошел в результате нарушения одним из модулей системы безопасности;

- в общем случае, на основе метаданных, приписанных к решениям, организуется соревнование (турнир) между различными *модулями* (как алгоритмами, представляемыми различными участниками) в решении сложной распределенной задачи.

В рассматриваемой системе реализуется обеспечение безопасности *(security)* и разделение доступа. Каждый *модуль* имеет либо *анонимный* доступ к системе, либо *авторизованный* (по уникальному идентификатору и паролю). Первый подход применим в случае, если модуль предоставляет *репозиторию* решения, оцениваемые *универсальной функцией приспособленности,* вычисленной на сервере, как "очень хорошие". В этом случае неважно, кто предоставил решения, главное, что *репозиторий* самостоятельно убедился в их "качестве". Другое дело – если модуль просит *репозиторий* добавить решения, на которых значение *универсальной функции приспособленности* уже вычислены самим *модулем*. С одной стороны перепроверка этих значений на сервере – затратная по времени операция. Однако, в то же время, если *модулем* была допущена ошибка при вычислении функции приспособленности, то в репозиторий попадут плохие решения, замещая хорошие. Это приведет к замедлению процесса эволюции. Еще более наглядные примеры – *модули*, специализирующиеся на вычислении *универсальной функции приспособленности* и на сборе различных статистик. На основе работы таких *модулей репозиторий* отдает решения, запрашиваемые другими *модулям*, и принимает решения от них. В системе определено *несколько ролей*, с каждой из которых связаны определенные *права*. Каждому *модулю* ставится в соответствие несколько ролей, таких как "администратор системы", "вычислитель статистики по решениям (тестам)", "вычислитель *универсальной функции приспособленности* для решений (тестов)" и т.д.

2.2. Характеристики методов генерации автоматов

2.2.1. Метод сокращенных таблиц переходов

Для оценки характеристик метода сокращенных таблиц был проведен ряд экспериментов, в которых эффективность данного метода сравнивалась с эффективностью метода полных таблиц. Сравнение проводилось по следующим критериям: объем занимаемой памяти, время, затрачиваемое на обработку каждого поколения, скорость роста функции пригодности (от номера поколения и от времени).

Как упоминалось выше, основное достоинство метода сокращенных таблиц состоит в том, что он решает проблему экспоненциального роста размера описания автомата с увеличением числа входных переменных (предикатов объекта управления). Это свойство метода подтвердилось при экспериментальной проверке. Во время эксперимента описания автоматов хранились в памяти компьютера. Поэтому объем занимаемой памяти в этом случае прямо пропорционален размеру описания автомата. Результаты эксперимента приведены на рис. 44.

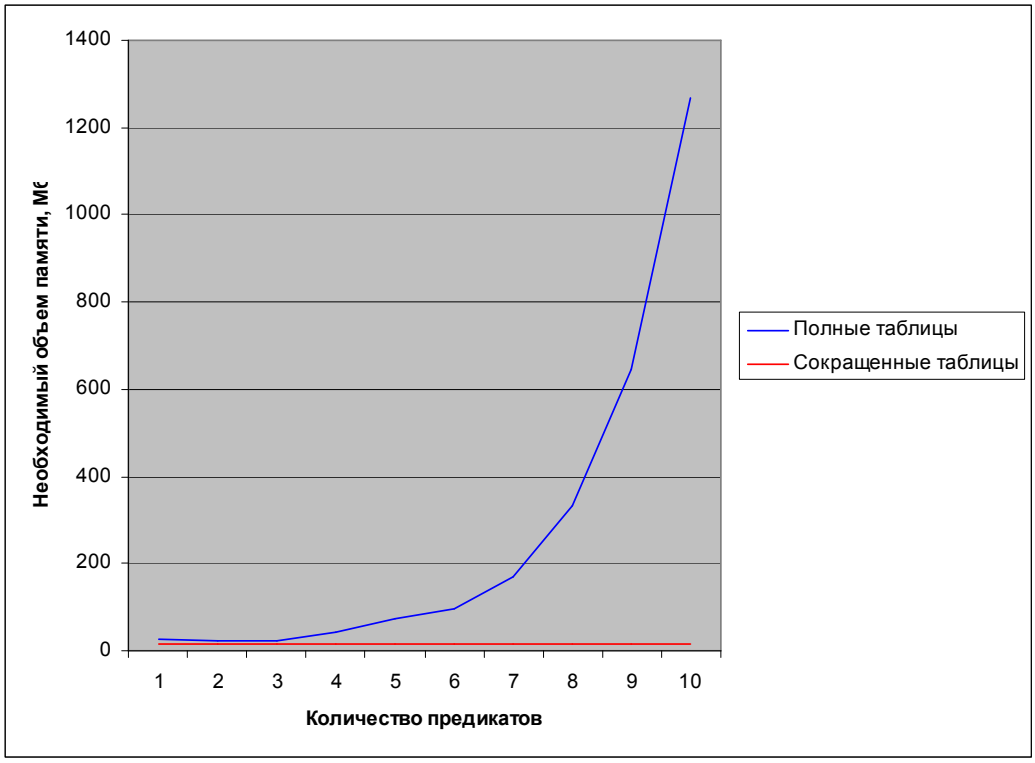


Рис. 44. Зависимость объема занимаемой памяти от числа предикатов

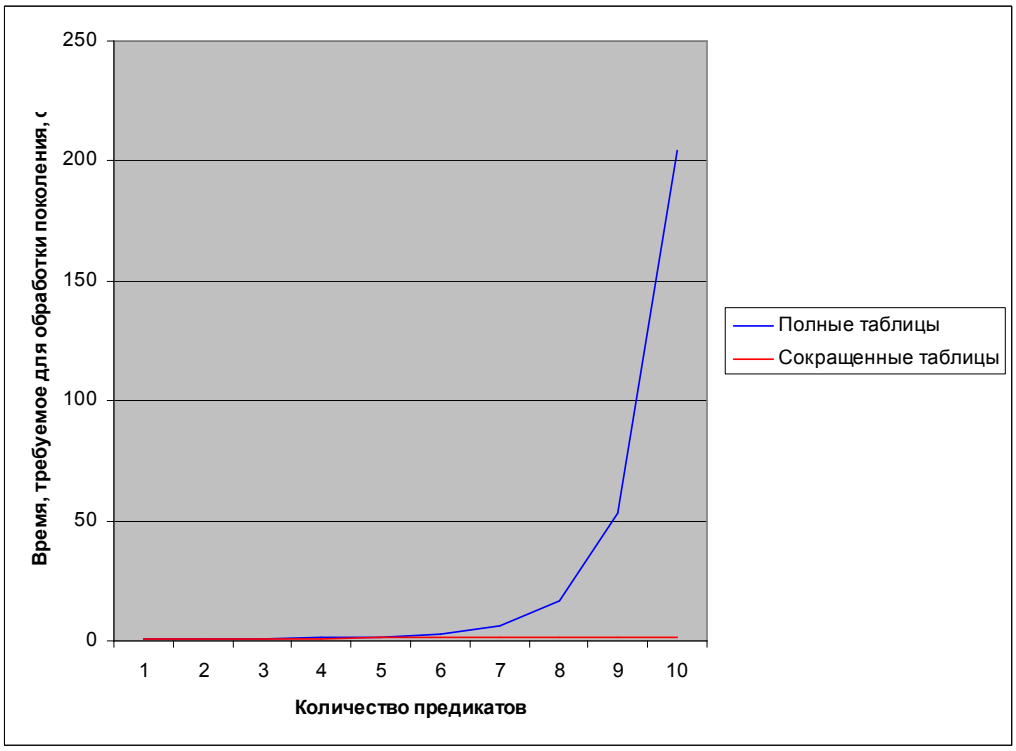


Рис. 45. Зависимость времени обработки поколения от числа предикатов

На графике видно, что объем занимаемой памяти при использовании метода полных таблиц растет экспоненциально, в то время как при использовании метода сокращенных таблиц объем занимаемой памяти с ростом числа предикатов практически не изменяется (в действительности он зависит от числа предикатов линейно, однако коэффициент линейной зависимости настолько мал, что в экспериментах она не отражается).

Аналогичный характер имеет зависимость времени, требуемого на обработку каждого поколения, от числа предикатов (рис. 45).

Теперь перейдем к оценке скорости роста функции пригодности. На рис. 46 приведены зависимости значения оценочной функции от номера поколения при использовании метода полных таблиц и метода сокращенных таблиц (измерения приводились при небольшом числе предикатов).

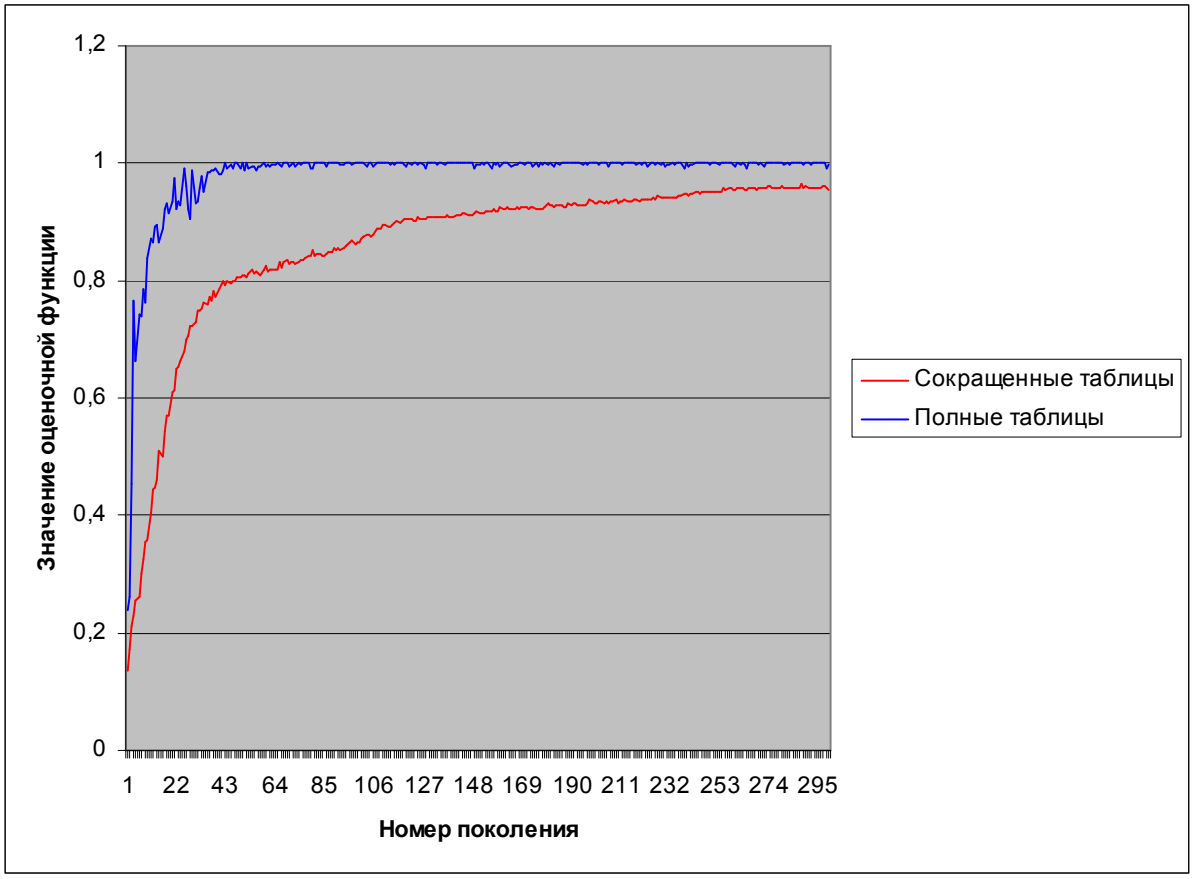


Рис. 46. Зависимость значения оценочной функции от номера поколения

Из графика следует, что оптимизация методом полных таблиц требует вычисления меньшего числа поколений. Это означает, что при небольшом числе предикатов метод полных таблиц обладает более высоким быстродействием. Однако с ростом числа предикатов стремительно растет время обработки одного поколения методом полных таблиц. Кроме того, из-за экспоненциального роста объема требуемой памяти применение метода полных таблиц, начиная с некоторого числа предикатов, становится не просто неэффективным, а практически невозможным. В экспериментах авторам не удалось построить методом полных таблиц автомат с более чем с 14 входными переменными.

Отметим также, что автоматы, которые построены методом полных таблиц, практически невозможно изобразить и понять, так как в них присутствует большое число избыточных переходов, а

условия на переходах громоздки. Напротив, автоматы, построенные, методом сокращенных таблиц, могут быть сравнительно легко поняты человеком.

Для того, чтобы адекватно оценить быстродействие обоих методов, необходимо установить зависимость значений оценочной функции, достигаемых с помощью каждого метода за определенное время, от числа предикатов. Такая зависимость для промежутка времени, равного пяти минутам, приведена на рис. 47.

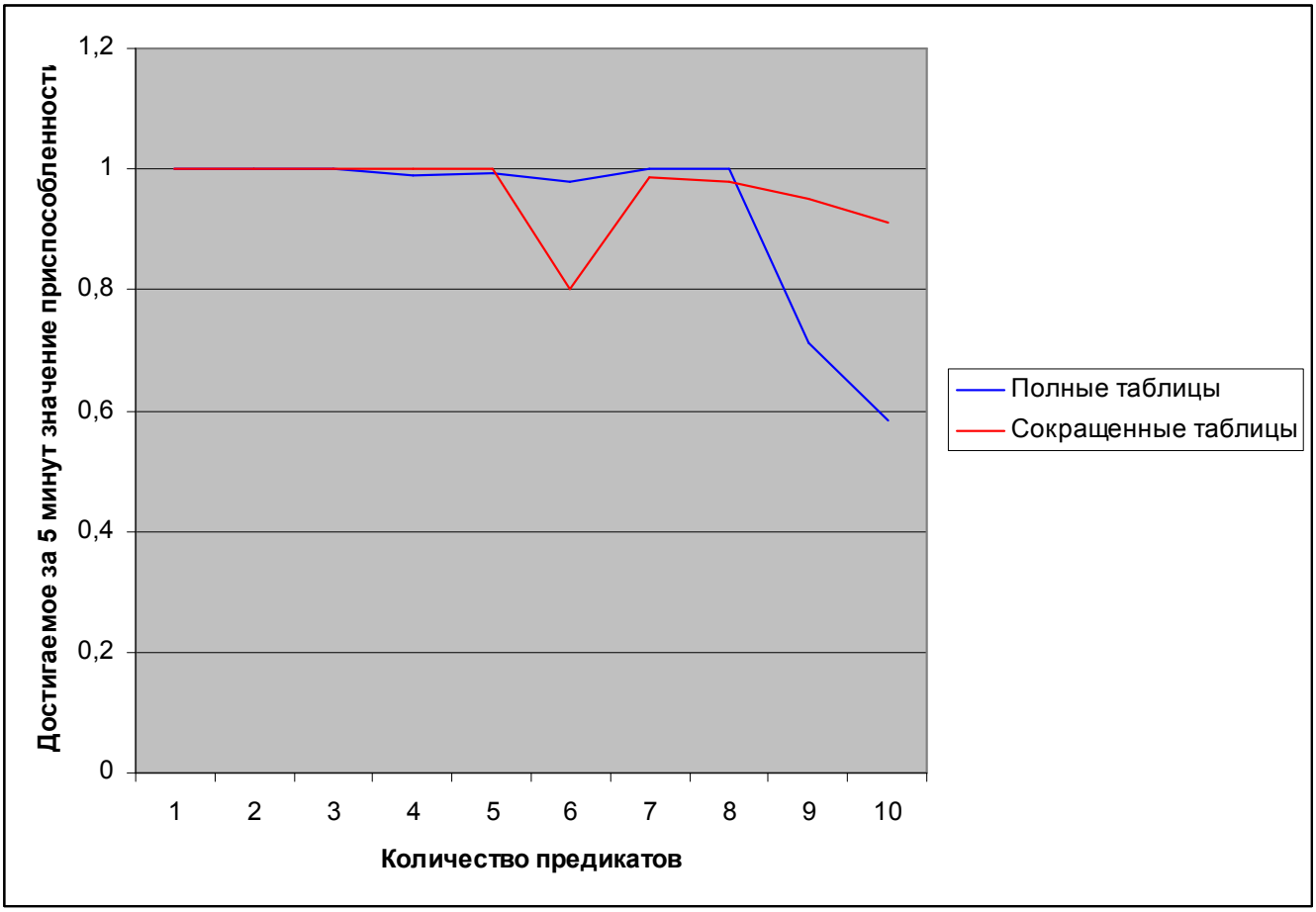


Рис. 47. Зависимость значения оценочной функции для заданной продолжительности работы метода от числа предикатов

Как и ожидалось, приведенные зависимости показывают, что при небольшом числе предикатов метод полных таблиц имеет более высокое быстродействие, однако с ростом числа предикатов его быстродействие резко падает. В то же время, быстродействие метода сокращенных таблиц с ростом числа предикатов уменьшается незначительно.

Из приведенного исследования характеристик методов полных и сокращенных таблиц можно сделать следующие выводы:

- при небольшом числе предикатов метод полных таблиц является более эффективным по времени и достаточно эффективным по памяти, однако построенные этим методом автоматы не понятны человеку;

- начиная с некоторого числа предикатов применение метода полных таблиц практически невозможно, в то время как метод сокращенных таблиц остается достаточно эффективным и по времени и по памяти.

2.2.2. Метод представления автоматов деревьями решений

Генетические операции над деревьями решений обладают следующими характеристиками:

1. Скрещивание деревьев происходит за $O(n)$, где n — число узлов в дереве.
2. Мутация деревьев происходит за время $O(n)$.
3. Обрезка недостижимых ветвей может быть выполнена за время $O(n)$.
4. Порождение случайного дерева происходит за время $O(n)$.

При этом, если ограничить максимальную высоту дерева величиной *m*, то все операции будут совершаться за время $O(2^m)$.

Генетических операции над автоматами, представленными в виде набора деревьев решений, совершаются за время $O(t)$, где t — суммарное число переходов автомата, представленного таким образом. Эта оценка следует из того факта, что суммарное число листьев в деревьях решений соответствует числу переходов автомата:

$$T_{automata} = \sum_{state \in S} T_{decree}(state) = \sum_{state \in S} O(n_{state}) = O(\sum_{state \in S} n_{state}) = O(t).$$

В случае, когда на генерируемый автомат наложено ограничение на максимальную высоту деревьев решений, нетрудно видеть, что генетические операции могут быть совершены за время $O(2^m \times S)$, где S — число состояний автомата.

Для хранения дерева решений из *n* узлов необходимо $O(n)$ памяти, а для хранения автомата требуется $O(t)$ памяти. Последняя оценка может быть получена аналогично оценке на время совершения генетических операций.

Число переходов автомата в рассматриваемом представлении в худшем случае экспоненциально от числа входных переменных. Это связано с тем, что существуют функции, плохо выражаемые деревьями решений. Однако, в большинстве практических задач число переходов значительно меньше.

2.2.3. Методы оптимизации генетических алгоритмов для построения автоматов

При n флаговых переменных число переходов из каждого состояния автомата будет равно m^{1+n}, где m – число допустимых комбинации значений входных переменных. Объем памяти, необходимой для хранения графа переходов автомата, пропорционален числу переходов из каждого состояния автомата. Создание новой особи из двух родительских также выполняется за время, пропорциональное числу переходов из каждого состояния автомата. Для того, чтобы генетический алгоритм перебирал решения для автоматов с флагами со скоростью не меньшей, чем для автоматов без флагов, число состояний в автоматах с флагами должно не больше $\frac{s}{m^{1+n}}$, где s – максимально число состояний для автомата без флагов.

2.2.4. Метод композитного генетического программирования

Если при работе эволюционного алгоритма в популяции появляется особь со значением приспособленности много превосходящим приспособленность остальных особей популяции, то *разнородность* (*diversity*) такой популяции (определяемая, например, как среднеарифметическое

расстояний Хэмминга между всеми парами хромосом популяции) быстро убывает. Это приводит к резкому уменьшению производительности эволюционного алгоритма. Обоснование этого факта приводится в работе [19]. Поэтому *модуль*, реализующий алгоритм решения задачи (*GEP, GA* или *GP*), периодически запрашивает решения, которые “чуть лучше” лучших из его собственных решений, а дальше пытаться самостоятельно улучшить полученный результат. Кроме того, *модуль* с целью повышения разнородности, в некоторые моменты времени предоставляет *репозиторию* свою популяцию, запрашивая решение, наиболее сильно отличающееся от его собственных решений, которое обладает примерно такой же приспособленностью. Возможен и другой вариант, когда *модуль* функционирует самостоятельно (без общения с *репозиторием*) до тех пор, пока идет оптимизационный процесс – значение функции приспособленности постепенно увеличивается. Как только модуль не способен улучшить решения самостоятельно – он обращается за помощью к *репозиторию*, заодно предоставляя ему наилучшие из своих решений.

Рассматриваемая система является *распределенной системой вычислений*, к которой в любое время подключаются как разные *модули*, так и по нескольку экземпляров одного модуля (на рис. 38 изображено два экземпляра модуля *GP*). Как показано в работе [44], распределенные эволюционные алгоритмы (поиск решения задачи выполняется сразу в нескольких независимых небольших популяциях, между которыми иногда осуществляется миграция хороших решений), которые обычно имеют сверхлинейную производительность по той причине, что особи в разных популяциях в процессе эволюции развиваются в направлении различных экстремумов, а миграция между популяциями позволяет улучшать независимо найденные решения. При этом общая разнородность, вычисляемая на объединении всех популяции, остается на более высоком уровне, чем в случае нераспределенного подхода – когда поиск решения выполняется в одной популяции большого размера.

2.3. Условия и ограничения функционирования методов генерации автоматов

2.3.1. Метод сокращенных таблиц переходов

Из постановки задачи следует, что описываемый метод применим только в том случае, если имеется готовый объект управления (не только описание множества вычислительных состояний, но и реализация предикатов и действий). В связи с этим, одним из перспективных направлений развития метода является решение задачи оптимизации предикатов и действий объекта управления совместно с оптимизацией управляющего автомата.

Кроме того, отметим, что предложенный метод для простоты использует частный случай модели автоматизированного объекта управления. Во-первых, отсутствуют внешние события, а на автомат влияют только значения предикатов объекта управления. Во-вторых, действия возможны только на переходах (используется автомат Мили). Наконец, предикаты и действия не имеют параметров.

Для эффективного использования метода сокращенных таблиц необходимо, чтобы используемые в задаче предикаты имели локальную природу. Если для решения задачи требуется контроль всей совокупности входных данных, и учет подмножеств предикатов ограниченной мощности не позволяет принимать управляющие решения, применение данного метода будет неэффективным. Примером может служить задача отслеживания изменения входных воздействий.

Пусть имеется три бинарных входных воздействия, и требуется произвести некоторое действие в случае любого изменения входных параметров. В силу того, что требуется постоянный контроль всех воздействий, применение метода сокращенных таблиц эквивалентно применению полных таблиц. На рис. 48 показан пример автомата, который может быть получен в результате применения метода полных таблиц или эквивалентного ему метода сокращенных таблиц без

ограничения числа значащих предикатов в состоянии (точнее, максимальное число значащих предикатов принимается равным общему числу предикатов, в данном случае трем).

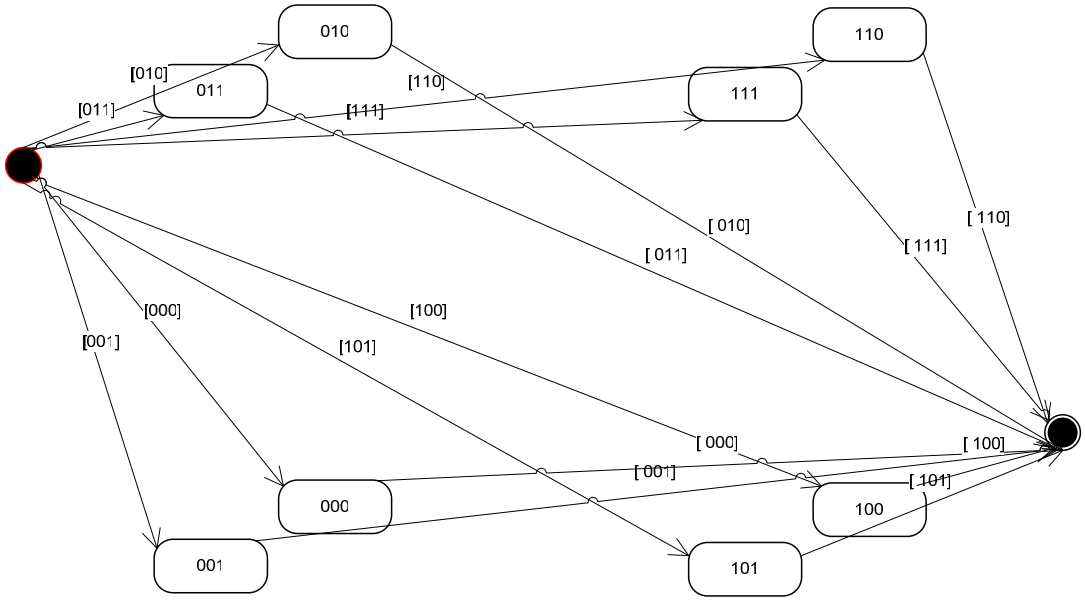


Рис. 48. Результат работы метода полных таблиц

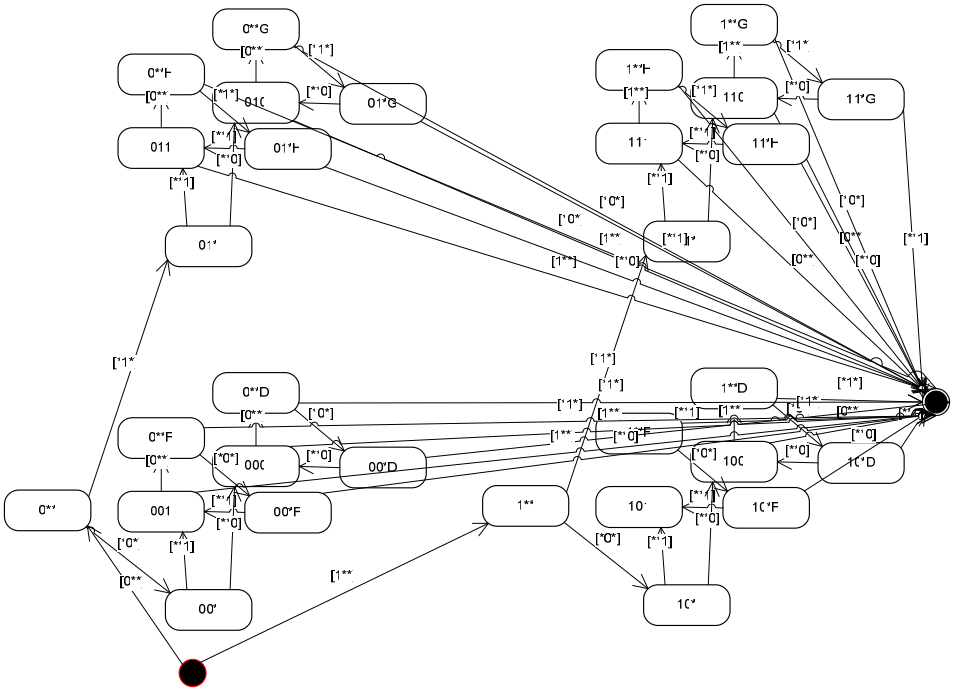


Рис. 49. Работа метода сокращенных таблиц с единственным значимым предикатом

Ограничение же числа значимых предикатов приводит к существенному росту числа состояний и переходов в автомате. Пример результата работы метода сокращенных таблиц с одним значимым в каждом состоянии предикатом приведен на рис. 49.

На рис. 50 приведено изображение правой нижней четверти рис. 49. Остальные части аналогичны.

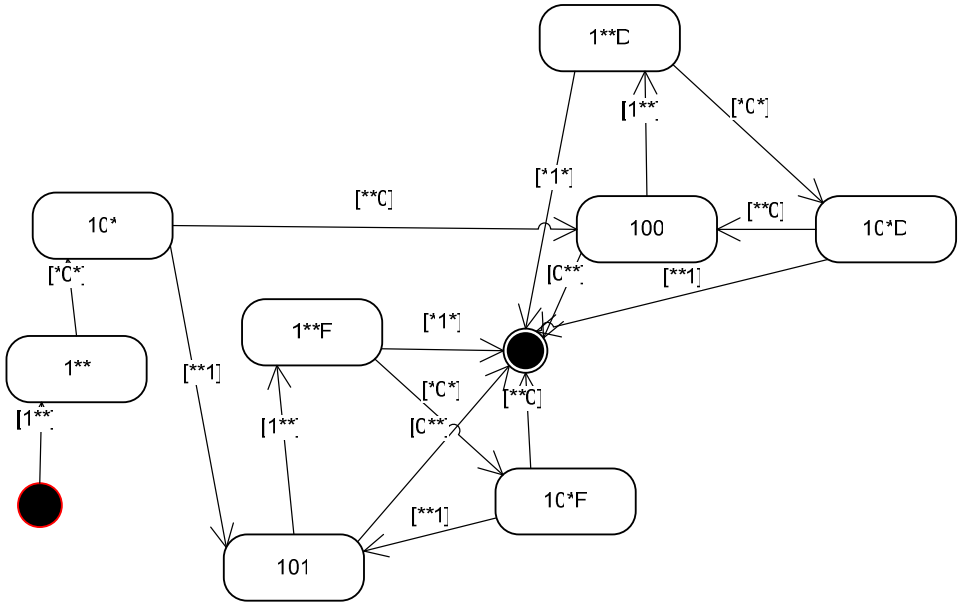


Рис. 50. Элемент результата работы метода сокращенных таблиц с единственным значимым предикатом.

В рассмотренном случае ограничение числа значимых предикатов привело к следующим последствиям:

- увеличение размерности пространства поиска, пропорциональное отношению чисел состояний автоматов, в 3,44 раза;
- увеличение размерности для выбора значимых предикатов на 17%;
- сокращение размерности пространства поиска за счет учета единственного предиката в состоянии: в четыре раза;
- потеря информации в результате скрещивания состояний с разными учитываемыми предикатами, а также мутации значимых предикатов.

В итоге применение метода сокращенных таблиц для решения рассмотренной задачи нецелесообразно. Однако, для большинства практических задач характерна локальная природа предикатов, и потому, метод сокращенных таблиц оказывается более эффективным.

2.3.2. Метод представления автоматов деревьями решений

Метод может быть применен для решения задач, допускающих представление в виде автомата Мура или Мили с дискретными входными переменными. Генерация систем взаимодействующих или вложенных автоматов не поддерживается.

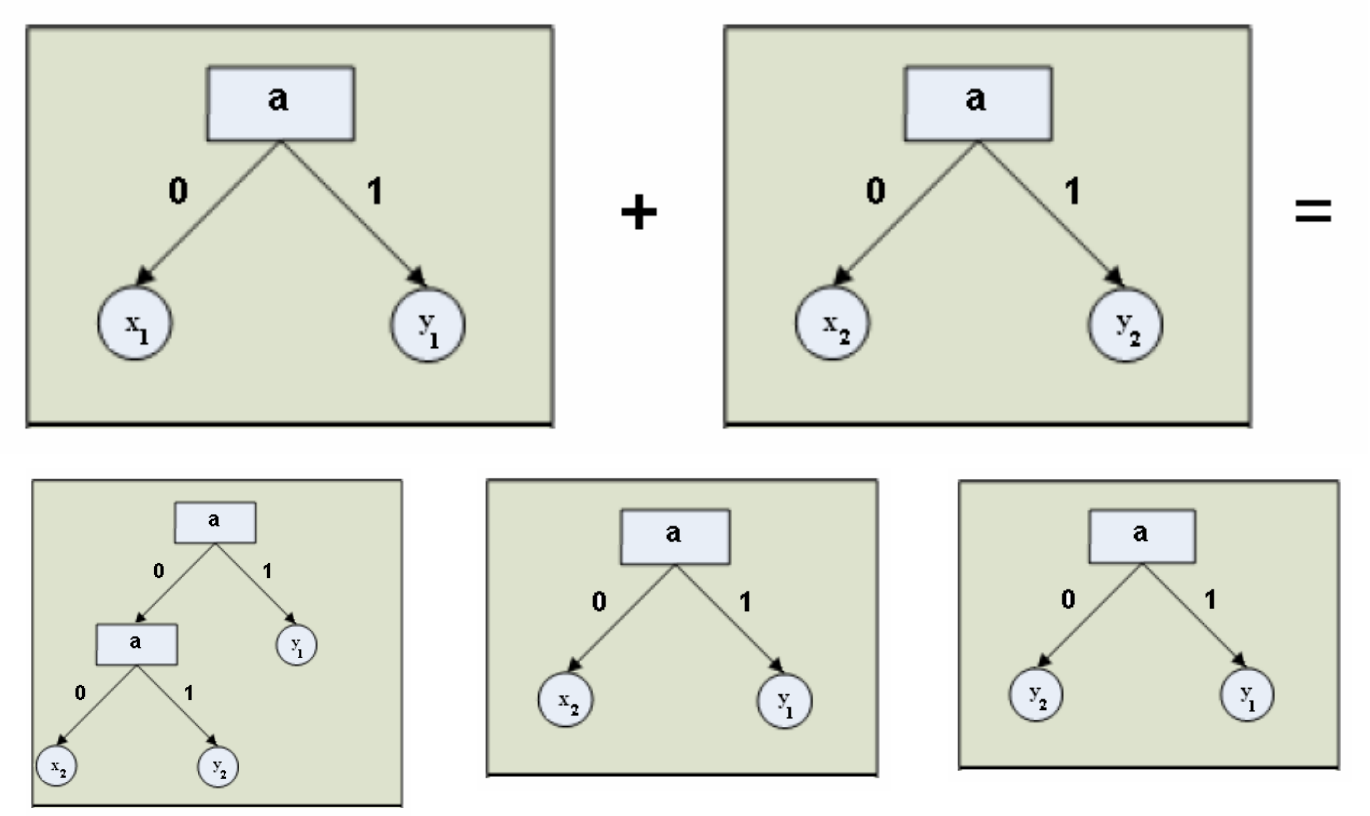
Метод показывает лучшие по сравнению с традиционными генетическими алгоритмами результаты при соблюдении следующих условий:

1. Переходы искомого автомата из зафиксированного состояния зависят от малого числа переменных по сравнению общим числом входных переменных автомата.
2. Общее число переменных автомата достаточно велико.

При нарушении первого условия деревья решений являются недостаточно выразительными для представления функции перехода. В таком случае хранение полной таблицы переходов является более предпочтительным.

Покажем, чем плохо нарушение второго условия. Для этого рассмотрим скрещивание двух деревьев решений, высота которых равна единице, помеченных одним и той же переменной *a*. Пометим для наглядности хранимые в листьях деревьев значения как x_1, y_1, x_2, y_2. При этом *x* — значение функции при *a* равном нулю, а *y* — при *a* равном 1.

Пусть для примера в первом дереве выбран узел, соответствующий x_1. После этого, в зависимости от выбора узла во втором дереве, равновероятно может произойти одна из трех ситуаций: замена узла x_1 вторым деревом либо одним из листьев x_2 и y_2 (рис. 51).



а) б) в)

Рис. 51. Скрещивание деревьев малой высоты

Можно видеть, что в одном случае из трех (случай в) значение, соответствующее *y*, скопировалось в *x*. При использовании же представления автоматов в виде таблицы переходов, либо битовой строки, такое бы не произошло. Рассмотренное копирование ведет к появлению плохого потомства и существенно замедляет генерацию.

При увеличении числа предикатов рассматриваемый эффект нейтрализуется.

2.3.3. Методы оптимизации генетических алгоритмов для построения автоматов

Для описанных в разд. 1.3 методов оптимизации генетических алгоритмов предполагается, что решения, с которыми ведется работа, хранятся в виде графов переходов. Эти методы также можно использовать, если структура хромосомы особи позволяет преобразовать ее в граф переходов, а затем выполнить обратную операцию. Например, преобразуем битовую строку в граф переходов, выполняем восстановление связей между состояниями и кодируем получившийся граф переходов с помощью битовой строки. Однако такой подход связан с увеличением вычислительных ресурсов.

Использовать алгоритм сортировки состояний в порядке использования не целесообразно, если при вычислении значений оценочной функции на вход автоматам одного поколения подаются различные последовательности значений входных переменных. Это происходит, например, при использовании совместной эволюции [26].

2.3.4. Метод композитного генетического программирования

Метод композитного генетического программирования является эффективным при решении задач, для которых выполняются следующие условия:

- существуют различные эволюционные алгоритмы решения задачи;
- решения задачи представимы различными способами в виде особи популяции и скорость эволюционного поиска существенно зависит от выбранного представления;
- различные эволюционные алгоритмы решения задачи приводят популяцию в направлении одних экстремумов с большей вероятностью, чем в направлении других;
- при решении задачи распределенными эволюционными алгоритмами имеет место сверхлинейная производительность.

Соображения в поддержку указанных условий приведены в разд. 1.4 настоящего отчета.

2.4. Выводы

Метод сокращенных таблиц переходов позволяет эффективно сокращать размерность пространства поиска оптимальных решений. В частности, пространство поиска увеличивается пропорционально числу состояний автоматов. При этом потеря информации в результате скрещивания состояний несущественно влияет на эффективность.

Метод представления автоматов деревьями решений также демонстрирует высокую эффективность при большом числе входных переменных, особенно если переходы зависят от малой их доли. В случае малого числа входных переменных или большого числа входных переменных на каждом ребре, метод сокращенных таблиц переходов является предпочтительным.

Методы оптимизации генетических алгоритмов, разработанные в разд. 1.3, могут применяться совместно как с методом сокращенных таблиц, так и с представлением автоматов деревьями решений. При этом достигается большая эффективность этих методов за счет дальнейшего сокращения пространства возможных решений.

Метод композитного генетического программирования также позволяет повысить скорость получения оптимальных решений за счет комбинирования нескольких генетических алгоритмов. При этом получаемый алгоритм заимствует сильные стороны у комбинируемых алгоритмов. Однако, данный метод не применим, если у комбинируемых алгоритмов, скорости приближений к оптимуму совпадают.

3. ТЕХНОЛОГИЯ ГЕНЕТИЧЕСКОГО ПРОРАММИРОВАНИЯ ДЛЯ ГЕНЕРАЦИИ АВТОМАТОВ УПРАВЛЕНИЯ СИСТЕМАМИ СО СЛОЖНЫМ ПОВЕДЕНИЕМ

В данной главе рассматриваются прототипы программных средств, созданные для оценки эффективности предложенных методов.

В разд. 3.1 рассматривается программное средство *AutoGen*, созданное на основе метода сокращенных таблиц переходов, а в разд. 3.2 — средство на основе представления автоматов деревьями решений. Эффективность оптимизации алгоритмов генетического программирования для автоматов и композитного генетического программирования рассматриваются в разд. 3.3 и 3.4 соответственно. Эксперименты, проводимые с созданными прототипами программных средств, позволят более точно оценить их эффективность на практике и определить области применимости.

3.1. ПРОГРАММНОЕ СРЕДСТВО *AUTOGEN*

В целях экспериментальной проверки предложенных методов авторами было реализовано инструментальное средство *AutoGen*, позволяющее производить построение логики системы со сложным поведением, задавая в качестве входных данных только реализацию объекта управления и оценочной функции.

3.1.1. Модель программного средства

Прототип позволяет использовать как упрощенную, так и усовершенствованную версии метода, позволяя сравнивать их эффективность, а также выражать общность и выделять различия в форме наследования классов. Описание структуры наследования классов на *Unified Modeling Language* (*UML)* приведено на рис. 52, а аналогичная диаграмма в *Business Object Notation (BON)* – на рис. 53.

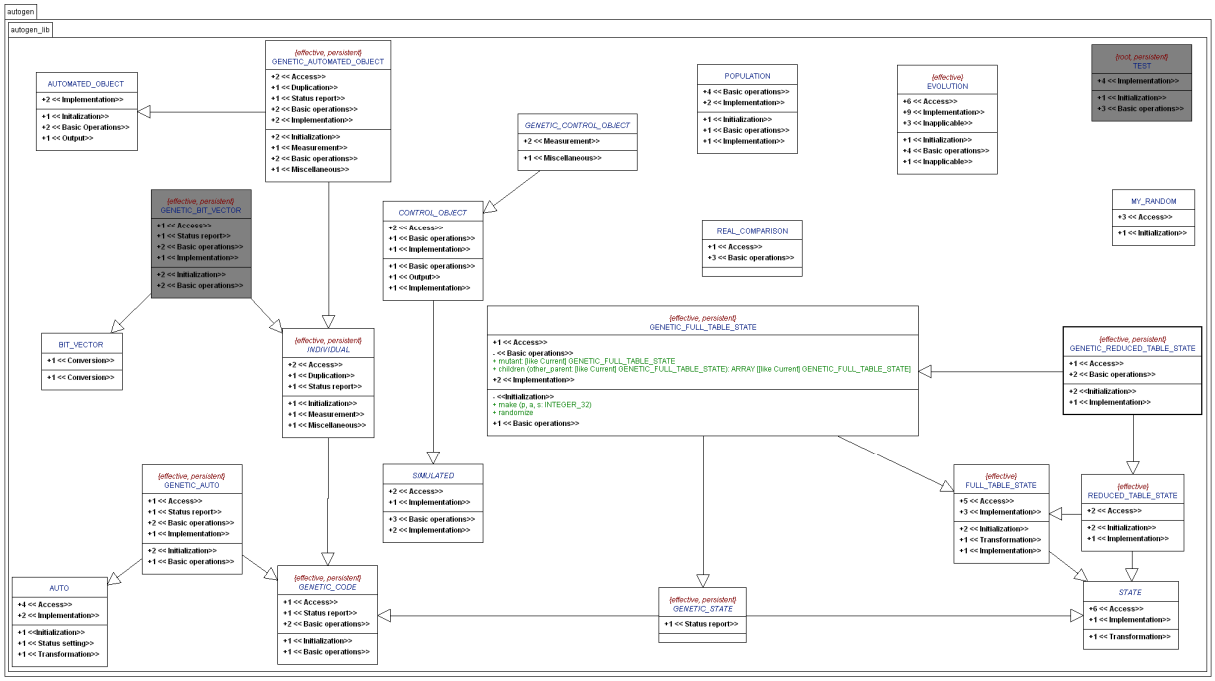


Рис. 52. Диаграмма классов: наследование (язык *UML*)

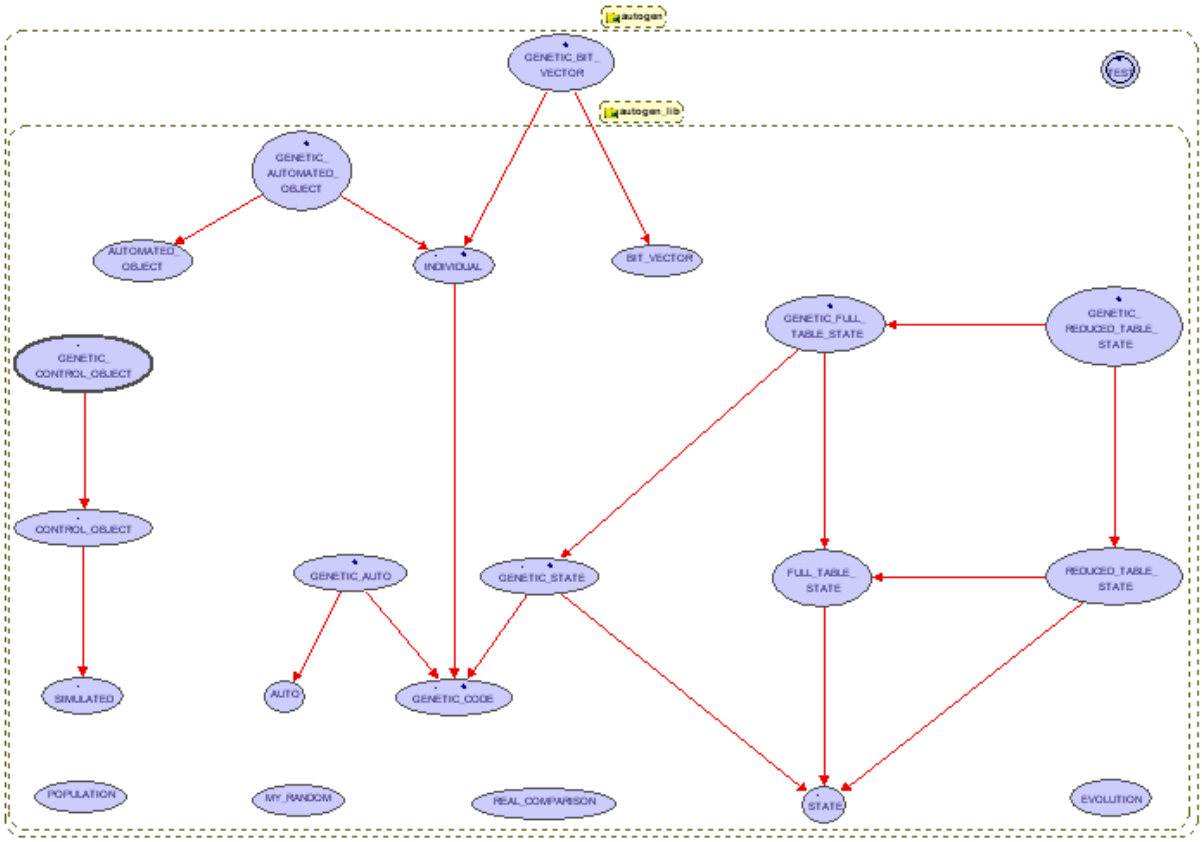


Рис. 53. Диаграмма классов: наследование (нотация *BON*)

Сущности автоматного программирования [54] безотносительно к генетическим алгоритмам описываются классами STATE (состояние конечного автомата), AUTO (конечный автомат), CONTROL_OBJECT (объект управления) и объединяющий их AUTOMATED_OBJECT (автоматизированный объект).

Аспект генетического программирования базируется на классе GENETIC_CODE (генетический код). Соединение класса GENETIC_CODE с классами STATE, AUTO и AUTOMATED_OBJECT порождает, соответственно, классы GENETIC_STATE, GENETIC_AUTO и GENETIC_AUTOMATED_OBJECT, которые объединяют автоматное и генетическое программирование. Взаимодействие между классами показано на рис. 54, .55.

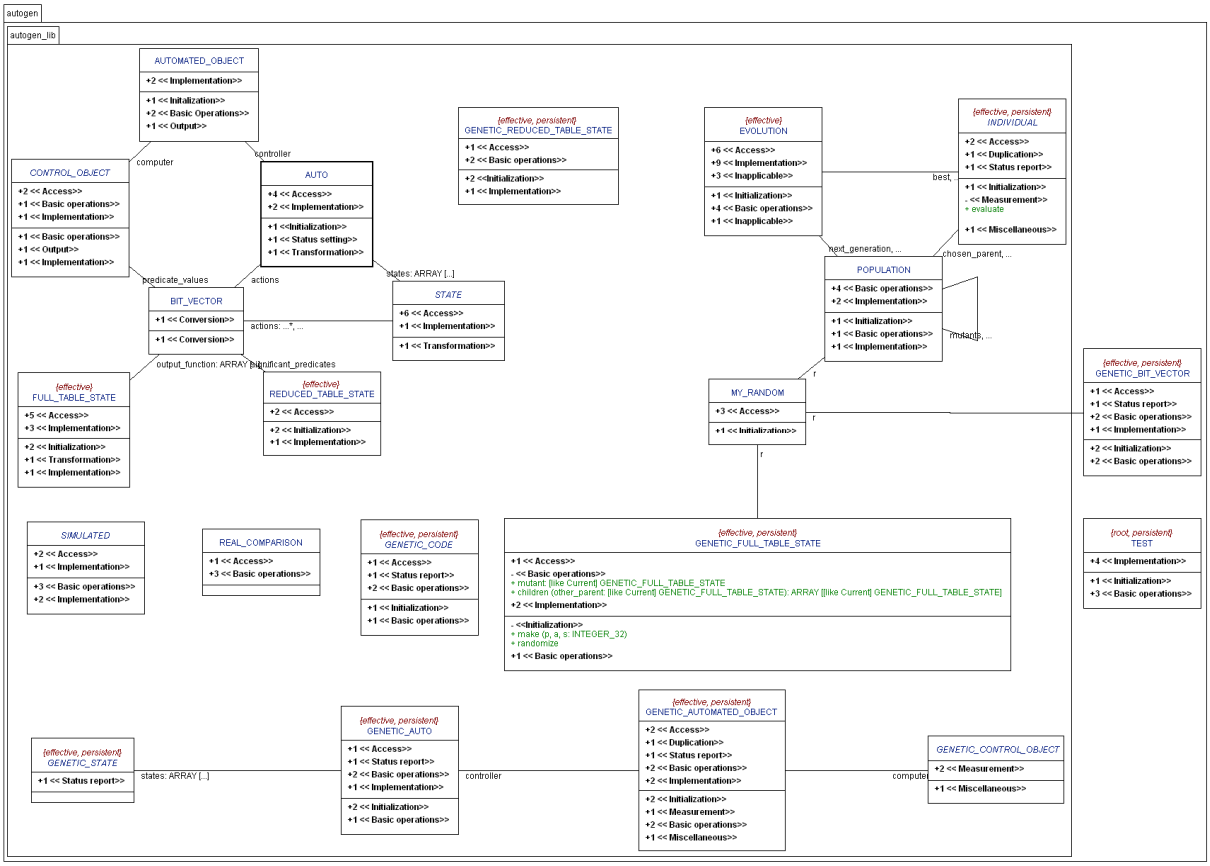


Рис. 54. Клиентские связи (язык *UML*)

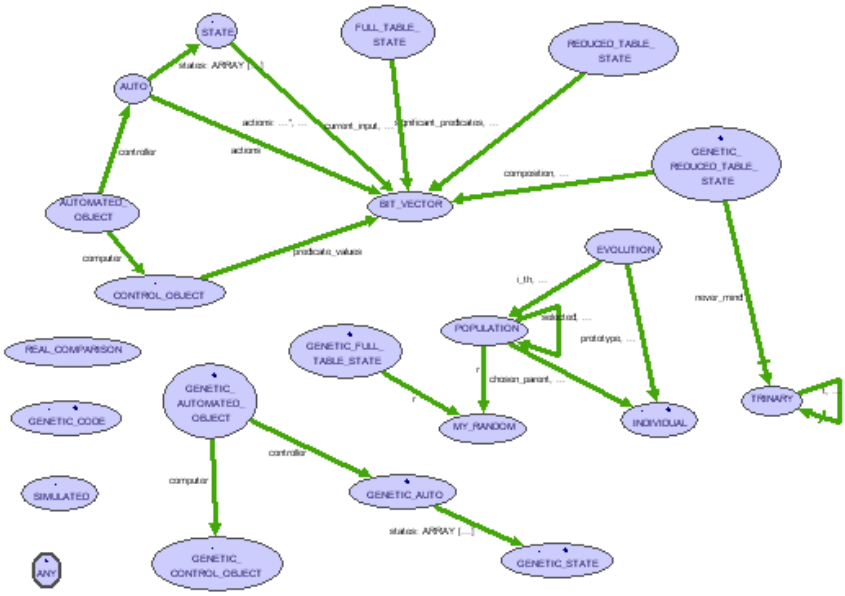


Рис. 55. Клиентские связи (нотация *BON*)

3.1.2. Прототип программного средства

Инструментальное средство *AutoGen* было реализовано авторами на языке программирования *Eiffel*. Экспериментальная проверка реализации была осуществлена на примере задачи построения управляющего автомата разливочной линии. Для этого авторами были разработаны эмулятор разливочной линии, реализующий действия и предикаты объекта управления и оценочная функция. Также был построен (вручную) один из автоматов, который решает поставленную задачу (рис. 58). Автомат содержит пять состояний. Здесь и далее стартовым является первое состояние.

Объект управления разливочной линии состоит из транспортера, на котором установлены бутылки, и дозирующей емкости (бака) с верхним (впускным) и нижним (выпускным) клапанами. Объем бака соответствует объему каждой бутылки.

Верхний клапан соединяет дозирующую емкость с трубой, подающей жидкость. Нижний клапан позволяет жидкости выливаться из бака и заполнять находящуюся под ним бутылку либо проливаться на транспортер, если бутылка не установлена. При наличии полной бутылки под нижним клапаном жидкость через клапан не проходит. Перед запуском линии все бутылки пусты, бак заполнен и под нижним клапаном установлена первая бутылка. Разливочная линия схематично изображена на рис. 56.

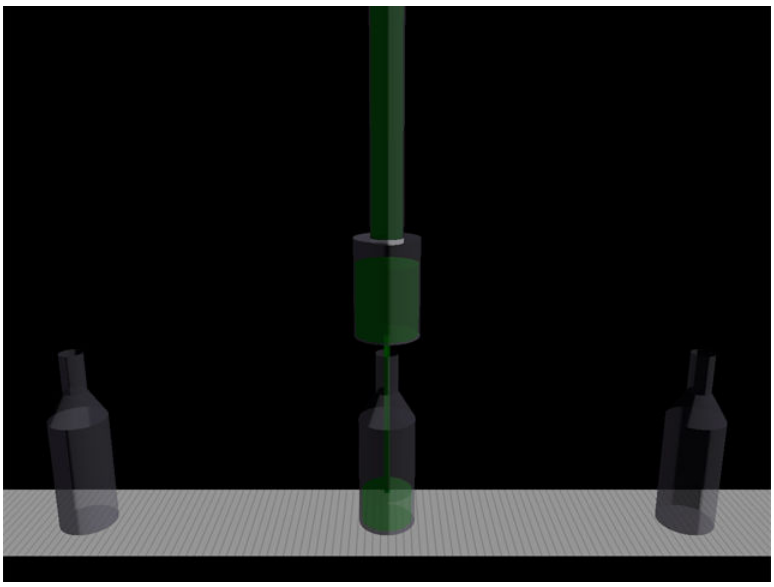


Рис. 56. Разливочная линия

Работой разливочной линии можно управлять, открывая и закрывая клапаны, а также запуская и останавливая транспортер. Управляющий автомат имеет пять двоичных входов от сигнализаторов, показывающих, правильно ли стоит бутылка под нижним клапаном, движение транспортера, опустошение и заполнение дозирующей емкости. Пятый вход автомата не несет информации и добавлен в экспериментальных целях. Схема связей управляющего автомата разливочной линии показана на рис. 57.

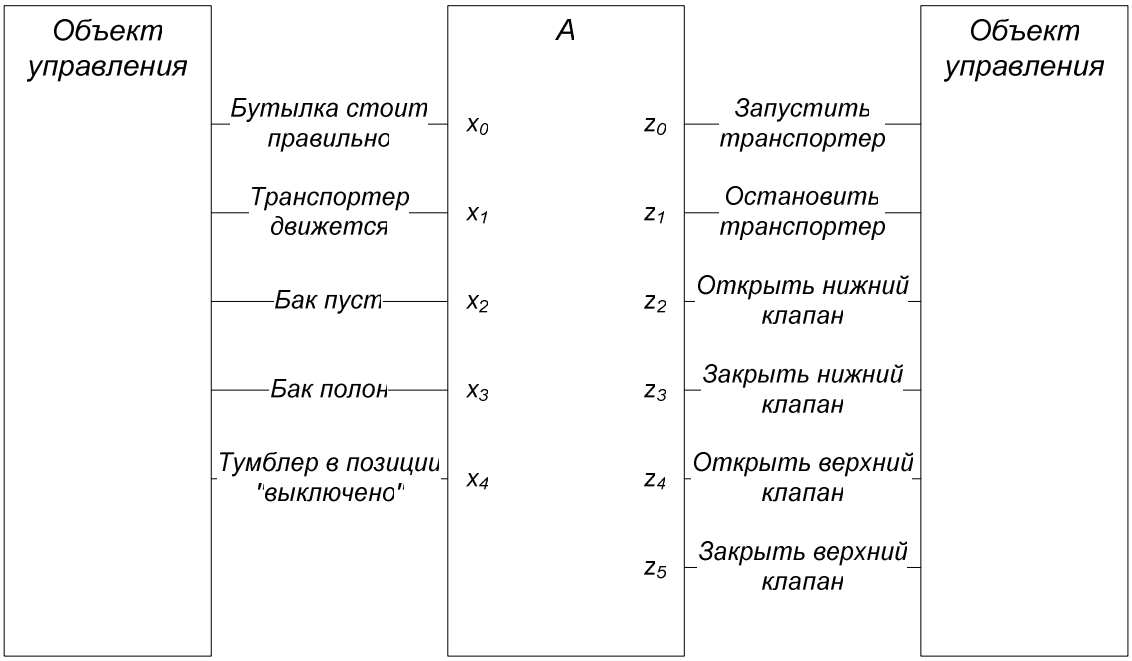


Рис. 57. Схема связей управляющего автомата разливочной линии

Задача состоит в заполнении максимального числа бутылок за определенный промежуток времени.

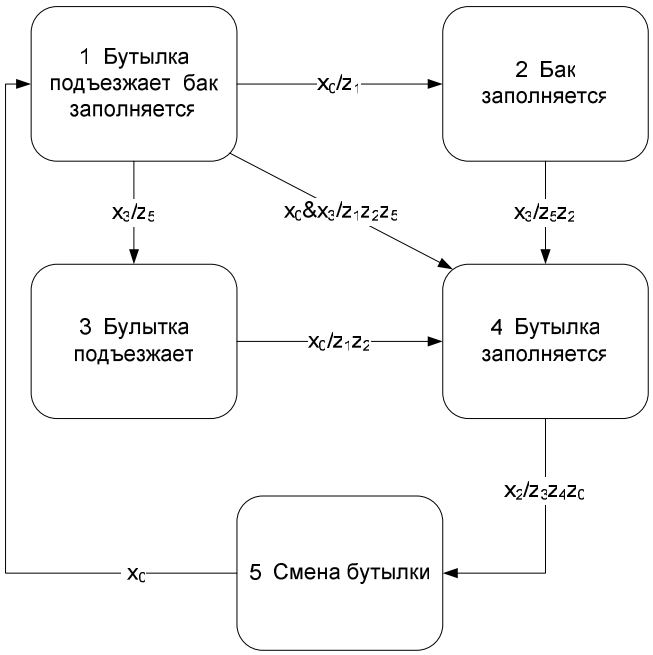


Рис. 58. Управляющий автомат разливочной линии, построенный вручную (|*S*| = 5, max(*r*) = 2)

После этого с помощью инструментального средства *AutoGen* были автоматически построены автоматы с представлением состояний в виде полных и сокращенных таблиц. Автомат,

представленный полными таблицами, имеет по 32 перехода из каждого состояния. Это делает бессмысленным его изображение и анализ.

Автомат, представленный сокращенными таблицами, оказался значительно проще. Один из результатов оптимизации изображен на рис. 59.

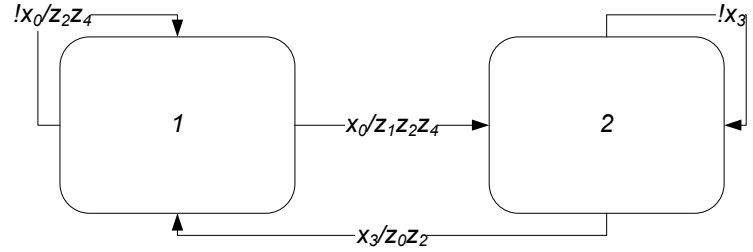


Рис. 59. Управляющий автомат разливочной линии, полученный в результате генетической оптимизации (|*S*| = 2, *r* = 1)

Анализ полученного автомата показал, что его работоспособность вызвана неявными зависимостями в эмуляторе разливочной линии. В частности, при реализации эмулятора пропускные способности верхнего и нижнего клапанов были заданы как постоянные, причем первая выбрана вдвое больше второй. Автомат в процессе работы использует этот факт. Таким образом, было установлено, что первоначальные реализации объекта управления и оценочной функции не полностью соответствует словесной формулировке задачи. Этот недостаток был впоследствии устранен путем внесения в параметры эмулятора (скорость движения транспортера, пропускная способность клапанов) псевдослучайных отклонений. Результатом повторного применения генетической оптимизации стал автомат, граф переходов которого изображен на рис. 60.

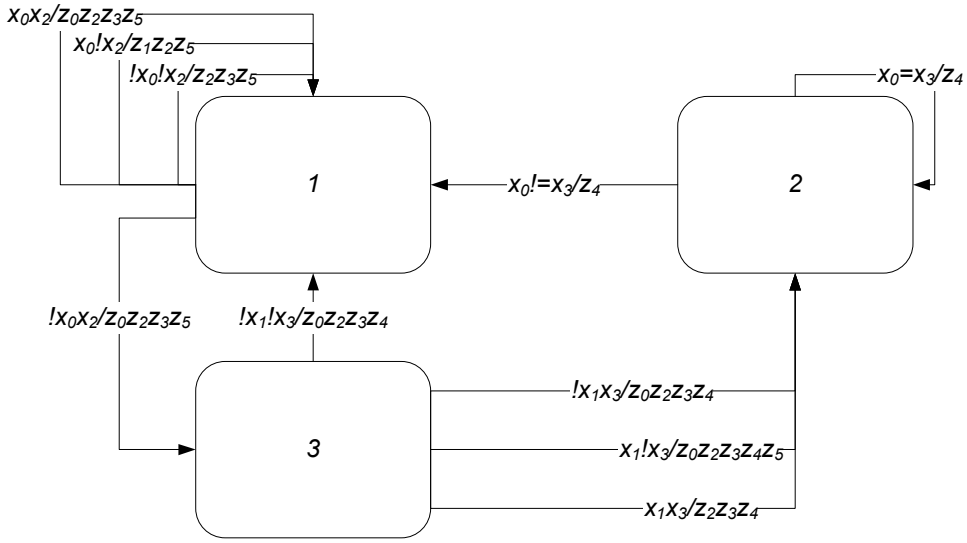


Рис. 60. Управляющий автомат "случайной" разливочной линии, полученный в результате генетической оптимизации (|*S*| = 3, *r* = 2)

3.2. Программное средство, основанное на представление автоматов деревьями решений

3.2.1. Модель программного средства

Предложенный метод генетического программирования для автоматных моделей был реализован в рамках проекта *Autoant* — разработка фреймворка для написания генетических алгоритмов и исследования задачи «Умный муравей» и ее модификаций, рассмотренных в работе [47]. Фреймворк обладает следующей функциональностью:

- архитектура проекта позволяет проводить исследования задачи «Умный муравей» и ее модификаций;
- программные интерфейсы просты для использования;
- допускается использование фреймворка из приложений, написанных на языках *C++* и *Java*;
- поведение муравья и соответствующего ему автомата, визуализируются;
- в рамках фреймворка реализованы различные эволюционные алгоритмы для решения исследуемых задач;
- модуль для реализации генетических алгоритмов.

Фреймворк разработан на языке *Java* и находится по адресу http://neerc.ifmo.ru/svn/automata/autoant. Его важным достоинством является то, что он применим для широкого круга задач, так как не ориентирован на специфику задачи «Умный муравей» и автоматного программирования.

Далее описывается только модуль фреймворка для реализации генетических алгоритмов.

На рис. 61 приведена диаграмма классов этого модуля.

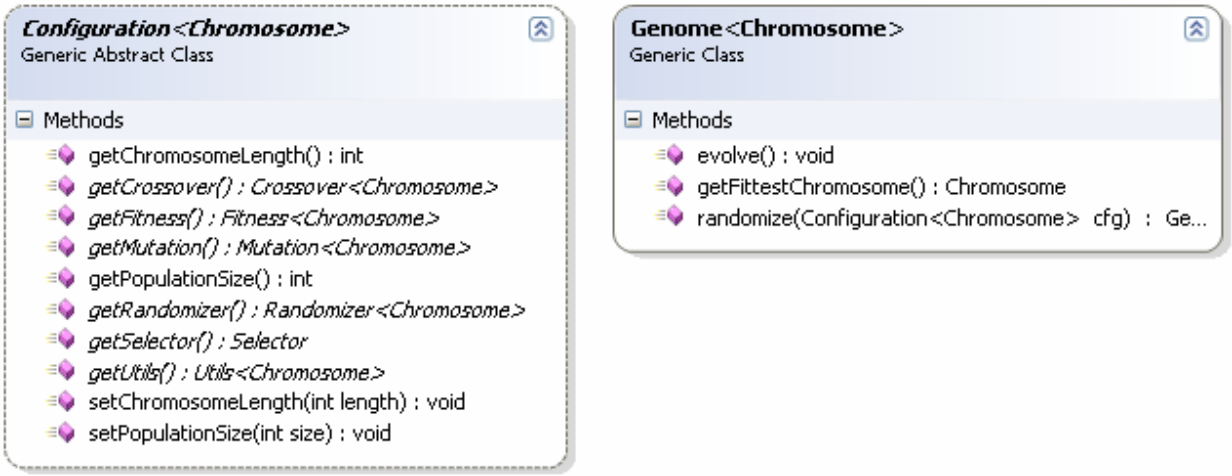


Рис. 61. Диаграмма классов модуля *autoant-ga*

Эволюционный алгоритм задается реализацией двух классов:

- *Configuration* — класс конфигурации эволюционного алгоритма. Параметром является класс, представляющий хромосому, который и будет генерироваться.
- *Genome* — класс, представляющий популяцию особей.

Эти классы являются *Generic*-классами, что позволяет варьировать способ представления и тем самым реализовывать различные варианты эволюционных алгоритмов.

Класс *Configuration* предоставляет доступ к реализациям генетических операций. Диаграмма интерфейсов генетических операций приведена на рис. 62.

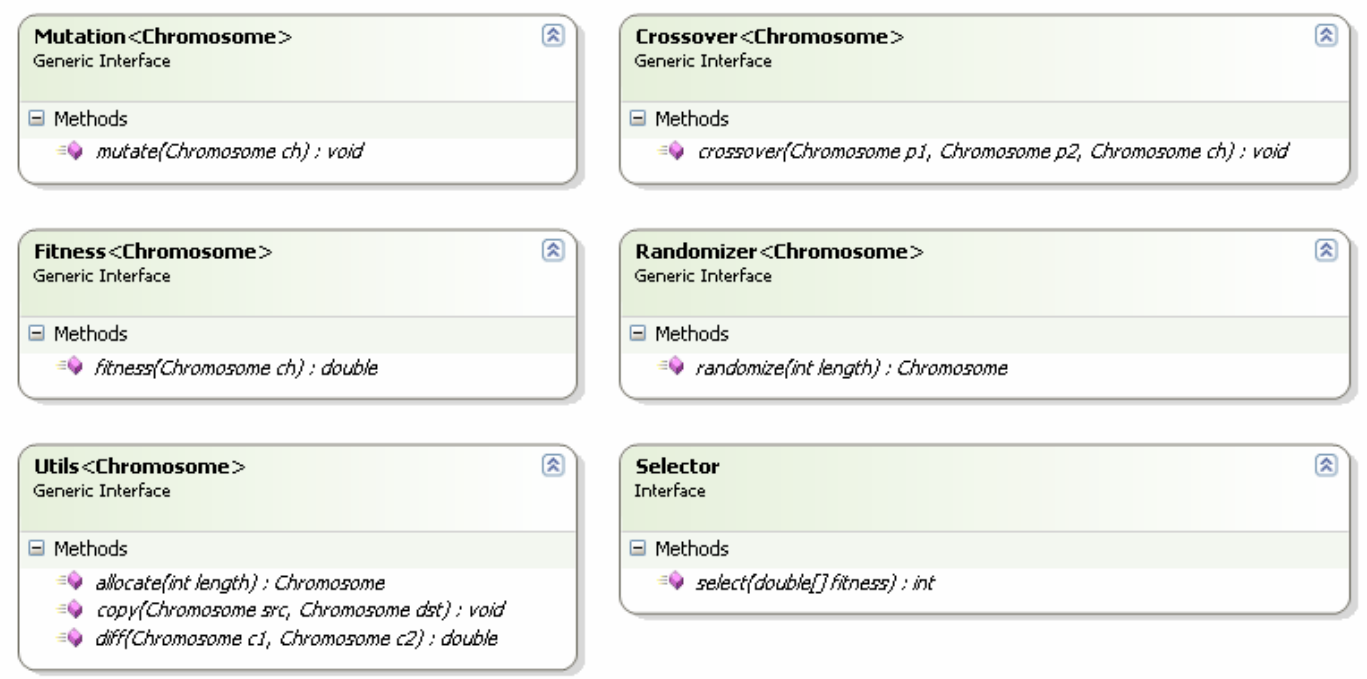


Рис. 62. Интерфейсы генетических операций

Приведем краткую характеристику этих интерфейсов:

- *Mutation* — интерфейс мутации;
- *Crossover* — интерфейс скрещивания, принимающего две особи и записывающего результат в третью;
- *Fitness* — интерфейс фитнесс-функции;
- *Randomizer* — интерфейс создания случайной особи;
- *Utils* — интерфейс утилит, позволяющих копирование особей, их сравнение и создание пустых хромосом.
- *Selector* — интерфейс отбора, выбирающего особь для размножения по набору значений фитнесс-функции;

Для реализации конкретного эволюционного алгоритма необходимо:

1. Реализовать класс-хромосому — представление сущности в виде особи эволюционного алгоритма.
2. Реализовать классы, наследующие интерфейсы генетических операций над классом-хромосомой.
3. Реализовать класс, наследующий класс *Configuration* и предоставляющий доступ к настройкам и генетическим операциям.

После этого класс конфигурации может быть использован для создания генома особей, с которым могут выполняться генетические алгоритмы.

В рамках работы над фреймворком были реализованы следующие алгоритмы генерации автоматов:

- генетические алгоритмы над битовыми строками;
- генетические алгоритмы над таблицами переходов;
- метод генетического программирования, использующий представление автоматов с использованием деревьев решений.

3.2.2. Прототип программного средства

Разработанный подход был протестирован на задаче «*Умный муравей-2*». Модифицируем задачу «*Умный муравей*» следующим образом: пусть муравей видит не одну клетку перед собой, а некоторую область (рис. 63).

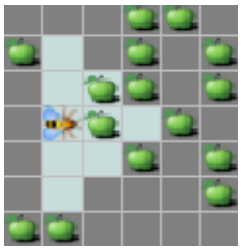


Рис. 63. Видимая область

В случае, когда карта зафиксирована, это преобразование упрощает задачу. Возможно дальнейшее обобщение задачи на случайные карты — пусть еда в каждой клетке располагается с некоторой вероятностью μ. Значение μ является параметром задачи. При этом необходимо найти такую стратегию муравья, которая максимизирует математическое ожидание съеденной еды.

Оптимальная стратегия в данной задаче сильно зависит от значения μ. Например, при $\mu = 1$ оптимальной является стратегия, при которой муравей посещает наибольшее число клеток, а в случае $\mu \approx 0$ — стратегия, когда муравей увидит наибольшее число клеток и обязательно посетит клетку, в которой увидит еду.

Автомат управления муравьем в этой задаче имеет восемь (число видимых клеток) входных переменных — каждая из которых определяет, есть ли еда в клетке, соответствующей переменной. Все входные переменные имеют логический тип. Применение автоматов в этой задаче оправдано, так как информация о клетках, увиденных на предыдущих шагах, может быть сохранена в состоянии автомата.

Предложенный подход сравнивался с генетическими алгоритмами, оперирующими над битовыми строками, и генетическими алгоритмами, оперирующими над таблицей переходов.

Первый эксперимент заключался в сравнении полученных значений оценочной функции (объема съеденной еды) за фиксированное число шагов с одинаковыми настройками. Запуск алгоритмов производился со следующими настройками:

- стратегия отбора — элитизм, для размножения отбираются 25% популяции, имеющих наибольшее значение фитнесс-функции;
- частота мутации — 2%;
- размер популяции — 200 особей;
- число популяций — 100;
- фитнесс-функция — среднее значение съеденной еды на 200 случайных картах (карты внутри одной популяции совпадают, карты различных популяций различны);

- последнее измерение фитнесс-функции осуществлялось на случайном наборе из 2000 карт.

Результаты эксперимента приведены в табл. 3.

Таблица 3. Результаты эксперимента

μ	0.01			
	Фитнесс-функция			
Число состояний	2	4	8	16
Битовые строки	2.74	2.93	2.83	2.89
Таблица переходов	2.68	3.49	3.74	3.78
Предложенный метод	2.71	2.92	2.88	3.69
μ	0.02			
	Фитнесс-функция			
Число состояний	2	4	8	16
Битовые строки	7.66	8.38	7.95	6.98
Таблица переходов	6.12	7.32	7.24	7.28
Предложенный метод	7.68	8.04	7.32	8.25
μ	0.03			
	Фитнесс-функция			
Число состояний	2	4	8	16
Битовые строки	14.46	13.81	13.23	11.93
Таблица переходов	12.48	12.17	11.72	11.15
Предложенный метод	14.14	13.86	13.77	14.18
μ	0.04			
	Фитнесс-функция			
Число состояний	2	4	8	16
Битовые строки	19.11	18.68	17.47	15.10
Таблица переходов	17.18	15.94	15.03	13.68
Предложенный метод	18.28	20.28	18.60	20.18

Второй эксперимент проверял ход эволюции в известных методах и в предложенном. Для этого осуществлялся запуск при тех же настройках, но не на 100, а на 400 популяций. Сравнить ход эволюции можно на приведенном графике (рис. 64).

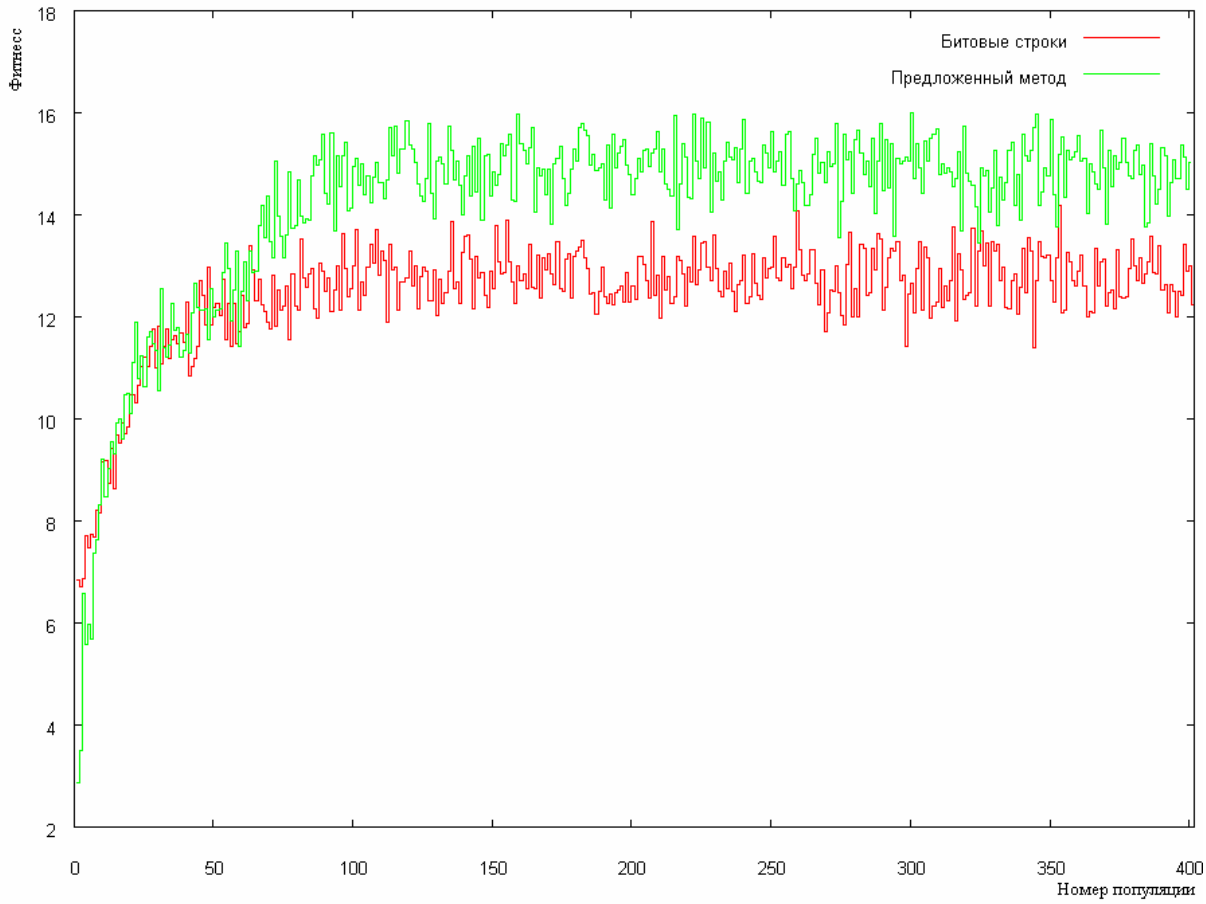


Рис. 64. Ход эволюции при $\mu = 0.03$ для автоматов с 16 состояниями

3.2.3. Результаты экспериментов

Анализ показывает, что в случаях, когда важны значения почти всех предикатов (при $\mu = 0.01$), предлагаемый метод работает хуже известных. Это можно обьяснить тем, что искомые автоматы плохо описываются деревьями решений. Однако, при больших значениях μ предложенный метод работает лучше, особенно при большом числе состояний. Таким образом, можно сделать вывод — метод работает в большинстве случаев, за исключением ситуаций, когда функция переходов не может быть эффективно выражена деревом решений.

Интересна зависимость высоты дерева от параметров, приведенная в табл. 4. Можно заметить, что средняя высота деревьев падает с увеличением числа состояний. Это можно объяснить тем, что многие клетки видимые муравью, были видны ему и раньше — например после шага в области видимости муравья остается три клетки, а после поворота — пять клеток. Информация о клетках, увиденных раньше, при возрастании числа состояний может храниться в номере состояния. Учитывая то, что в автоматном программировании состояния играют роль неких характеристик уже совершенных действий, можно считать, что предложенный метод хорошо подходит для генерации автоматов.

Таблица 4. Средняя высота деревьев в выведенных автоматах

μ	0.01			
Число состояний	2	4	8	16
Средняя высота	5.5	3.25	2.38	3.63
μ	0.02			
Число состояний	2	4	8	16
Средняя высота	5.0	7.0	2.88	2.0
μ	0.03			
Число состояний	2	4	8	16
Средняя высота	4.5	2.5	1.88	1.56
μ	0.04			
Число состояний	2	4	8	16
Средняя высота	4.0	3.0	1.75	2.63

3.3. Программное средство для оценки эффективности методов оптимизации генетических алгоритмов для построения автоматов

3.3.1. Модель программного средства

Для проведения экспериментов с методами оптимизации генетических алгоритмов для задачах о флибах и умном муравье было разработано программное средство на языке C#.

Реализация автоматов осуществляется с помощью трех классов – `StateMachine`, `State` и `Branch` (рис. 65).

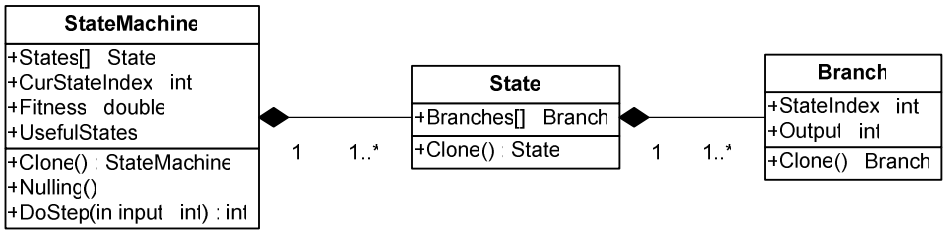


Рис. 65. Диаграмма классов для реализации автомата

В качестве главного класса при реализации флиба применяется класс `StateMachine`. Классы `State` и `Branch` реализуют его состояния и переходы соответственно. В каждом из этих классов имеется метод `Clone`, предназначенный для создания копий объектов.

Массив `States` в классе `StateMachine` содержит состояния автомата. Свойство `CurStateIndex` используется для хранения номера текущего состояния автомата в массиве `States`, а доступ к значению оценочной функции можно получить с помощью свойства `Fitness`. Словарь пар номеров [«старый номер состояния» – «новый номер состояния»] (далее **словарь используемых состояний**) доступен через свойство `UsefulStates`.

Метод `DoStep` переводит автомата в новое состояние, изменяет **словарь используемых состояний** и возвращает значение выходной переменной, генерируемой автоматом. Метод `Nulling` возвращает автомат в начальное состояние.

В массиве `Branches` класса `State` содержатся дуги переходов из данного состояния. Номер элемента в массиве соответствует значению входной переменной.

Переменные `StateIndex` и `Output` класса `Branch` – это номер состояния, в которое переходит флиб по этой дуге, и значение выходной переменной соответственно.

Для реализации стратегий отбора используются классы, изображенные на рис. 66. Эти классы реализуют интерфейс `ISelection`.

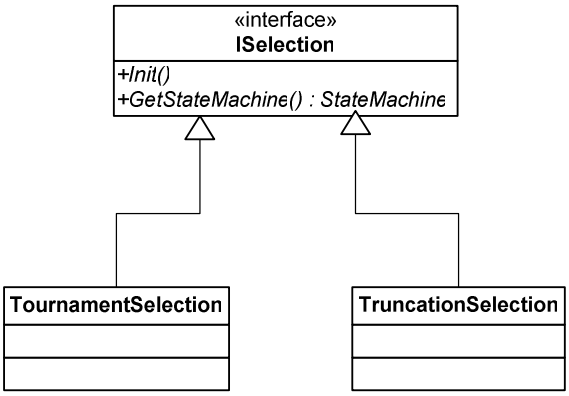


Рис. 66. Диаграмма классов для реализации стратегий отбора

В метод `Init` передаются автоматы текущего поколения, и происходит инициализация стратегии отбора. Метод `GetStateMachine` возвращает родительскую особь для скрещивания. Класс `TournamentSelection` реализует турнирный отбор [32], а класс `TruncationSelection` – стратегию отбора отсечением [22].

Для реализации операторов скрещивания используются классы, изображенные на рис. 67. Эти классы реализуют интерфейс `ICrossover`.

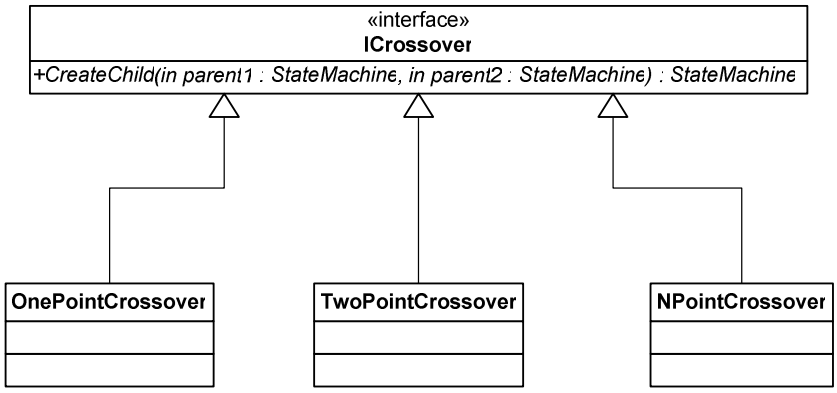


Рис. 67. Диаграмма классов для реализации операторов скрещивания

Метод `CreateChild` создает новый автомат из двух родительских. Классы `OnePointCrossover` и `TwoPointCrossover` реализуют одноточечный и двухточечный операторы скрещивания. Класс `NPointCrossover` реализует *n*–точечный оператор скрещивания.

Для реализации операторов мутации используются классы, изображенные на рис. 68. Эти классы реализуют интерфейс `IMutation`.

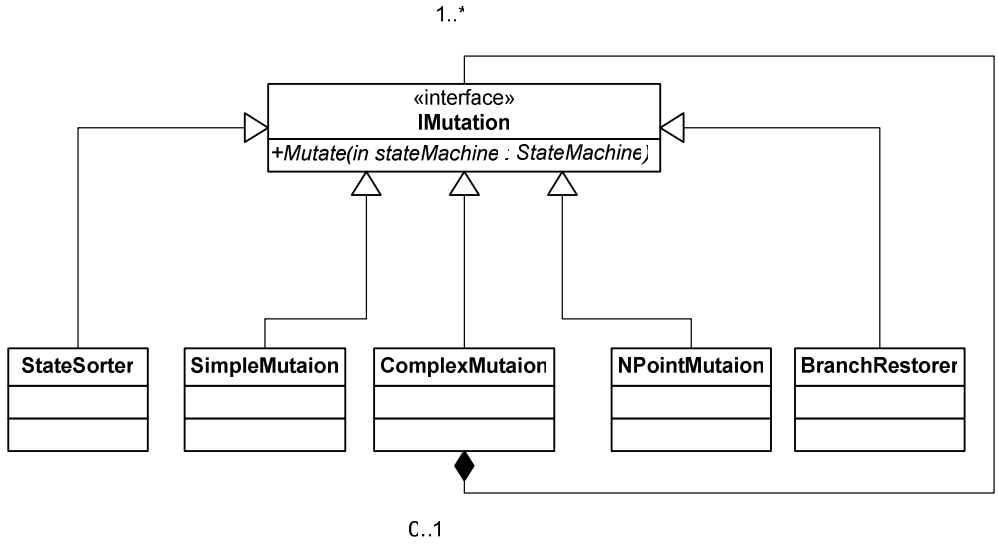


Рис. 68. Диаграмма классов для реализации операторов мутации

Метод `Mutate` выполняет операцию мутации над переданным в него автоматом. Классы `SimpleMutation` и `NPointMutation` реализуют стандартный и *n*–точечный операторы мутации [22] соответственно.

Для реализации алгоритма восстановления связей между состояниями используется класс `BranchRestorer`. Класс `StaterSorter` реализует алгоритм сортировки состояний в порядке использования.

Для выполнения нескольких операций мутации над одним автоматом применяется класс `ComplexMutation`.

Классы, реализующие вычисление значения оценочной функции, изображены на рис. 69. Эти классы реализуют интерфейс `IFitness`.

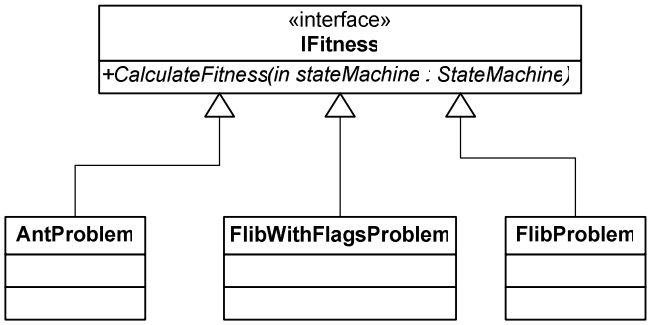


Рис. 69. Диаграмма классов для реализации вычисления значения оценочной функции

Метод `CalculateFitness` вычисляет значение оценочной функции для переданного в него автомата. Класс `AntProblem` моделирует поведение умного муравья и определяет значение оценочной функции для него. Класс `FlibProblem` выполняет туже функцию для флибов. Класс `FlibWithFlagsProblem` отвечает за работу с флибами, реализованными с помощью автоматов с флагами.

Основной класс `GAEngine`, реализующий работу генетического алгоритма, изображен на рис. 70.

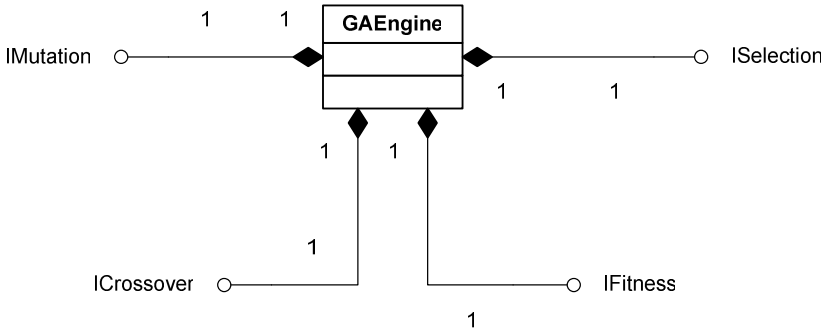


Рис. 70. Диаграмма классов для реализации генетического алгоритма

Класс `GAEngine` зависит от того, какие именно операторы скрещивания, мутации, стратегии отбора и оценочные функции используются. Такой подход позволяет легко модифицировать генетический алгоритм.

3.3.2. Прототип программного средства

Для проверки эффективности методов оптимизации генетических алгоритмов был проведен ряд экспериментов.

3.3.2.1. Эксперименты по использованию автоматов с флагами

Эксперименты, описываемые в этом разделе, проводились с помощью генетического алгоритма, решающего задачу о флибах. При формировании нового поколения применялся турнирный отбор [32] и принцип элитизма [14] (в новое поколение добавляется одна или несколько лучших особей из предыдущего поколения). Для формирования новой особи использовался одноточечный оператор скрещивания. Все эксперименты проводились при размере поколения 100 и вероятности применения одноточечного оператора мутации 0,03. Число воздействий среды на флиб выбрано равным 100. Благодаря этому число правильно предсказанных символов по величине равно точности предсказания символов в процентах. В таблицах с результатами экспериментов приводится минимальные, максимальные и средние точности предсказания автоматически построенных флибов. В качестве битовой маски применялась строка `1111010010111101001`.

На графиках по оси абсцисс указаны номера поколений, по оси ординат – число правильно предсказанных символов лучшим предсказателем в каждом поколении. При построении графиков использовались **усредненные данные**, полученные при проведении 50 экспериментов с одинаковыми начальными параметрами.

Чем больше состояний у автомата, тем более сложным может быть его поведение. Исходя из этого, можно попытаться улучшить точность предсказания с помощью увеличения числа состояний флиба. В табл. 5 приведены результаты экспериментов для флибов, заданных автоматом без флагов и имеющих число состояний от 10 до 80.

Таблица 5. Результаты экспериментов над флибами с разным числом состояний

Число состояний	Худший результат	Усредненный результат	Лучший результат
10	89	93,76	95
20	90	94,2	95
40	89	94,58	100
80	89	96	100

Как видно из таблицы, увеличение числа состояний действительно несколько улучшает точность построенного предсказателя.

Графики для экспериментов над флибами с разным числом состояний приведены на рис. 71. Точками изображен график для результатов эксперимента над флибами с 10 состояниями. Штрихпунктирная линия используется для флибов с 20 состояниями, а пунктирная линия для флибов с 40 состояниями. График для результатов эксперимента над флибами, имеющими 80 состояний, изображен сплошной линией.

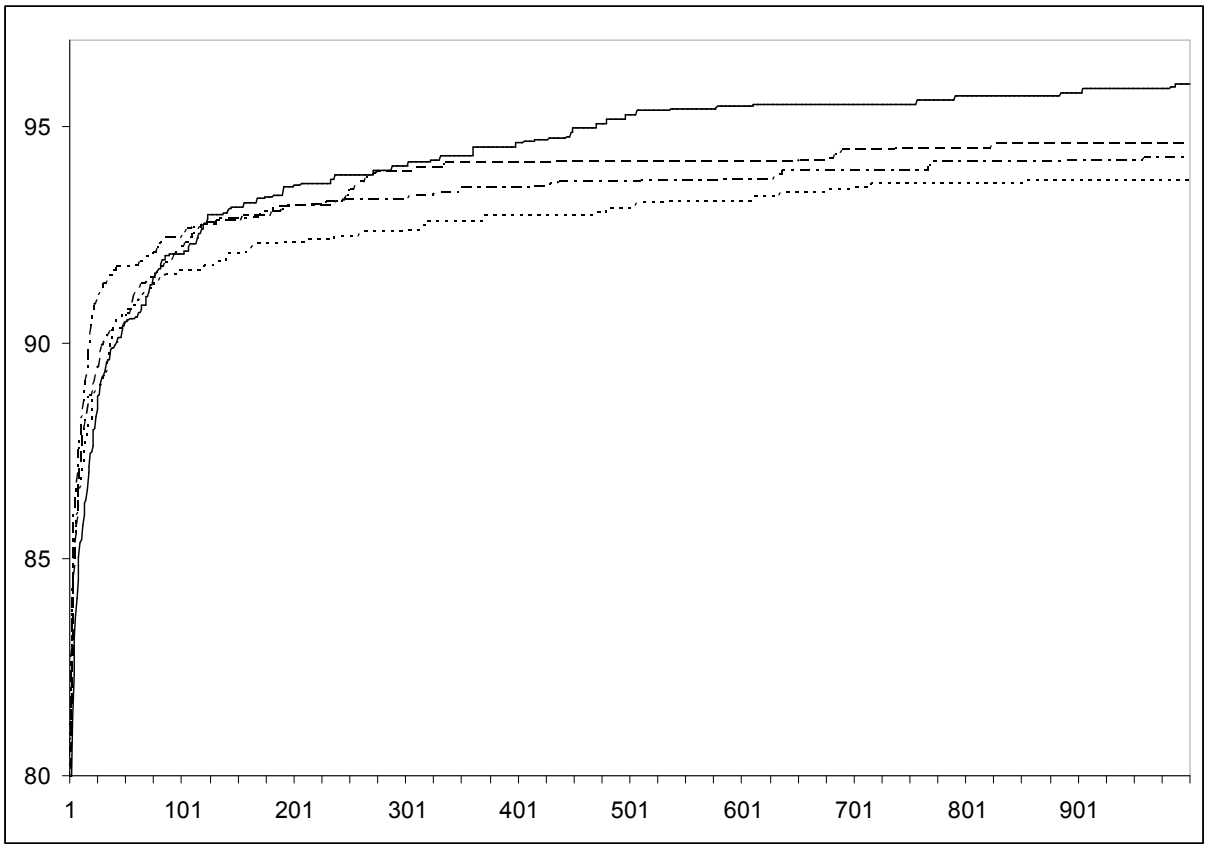


Рис. 71. Графики усредненных результатов для экспериментов над флибами с разным числом состояний

Основным недостатком увеличения числа состояний является рост объема вычислений и, как следствие, увеличение времени, необходимого для построения предсказателя.

В табл. 6 приведены результаты экспериментов над флибами с разным числом состояний и флагов.

Таблица 6. Результаты экспериментов над флибами с разным числом состояний и флагов

Число состояний	Число флагов	Худший результат	Усредненный результат	Лучший результат
80	0	89	96	100
40	1	90	96,82	100
20	2	95	98,54	100
10	3	95	99,74	100

Графики для экспериментов над флибами с разным числом состояний и флагов приведены на рис. 72. Точками изображен график для результатов эксперимента над флибами с 80 состояниями без флагов. Штрихпунктирная линия используется для флибов с 40 состояниями и одним флагом, а пунктирная линия используется для флибов с 20 состояниями и двумя флагами. График результатов эксперимента над флибами с 10 состояниями и тремя флагами изображен сплошной линией.

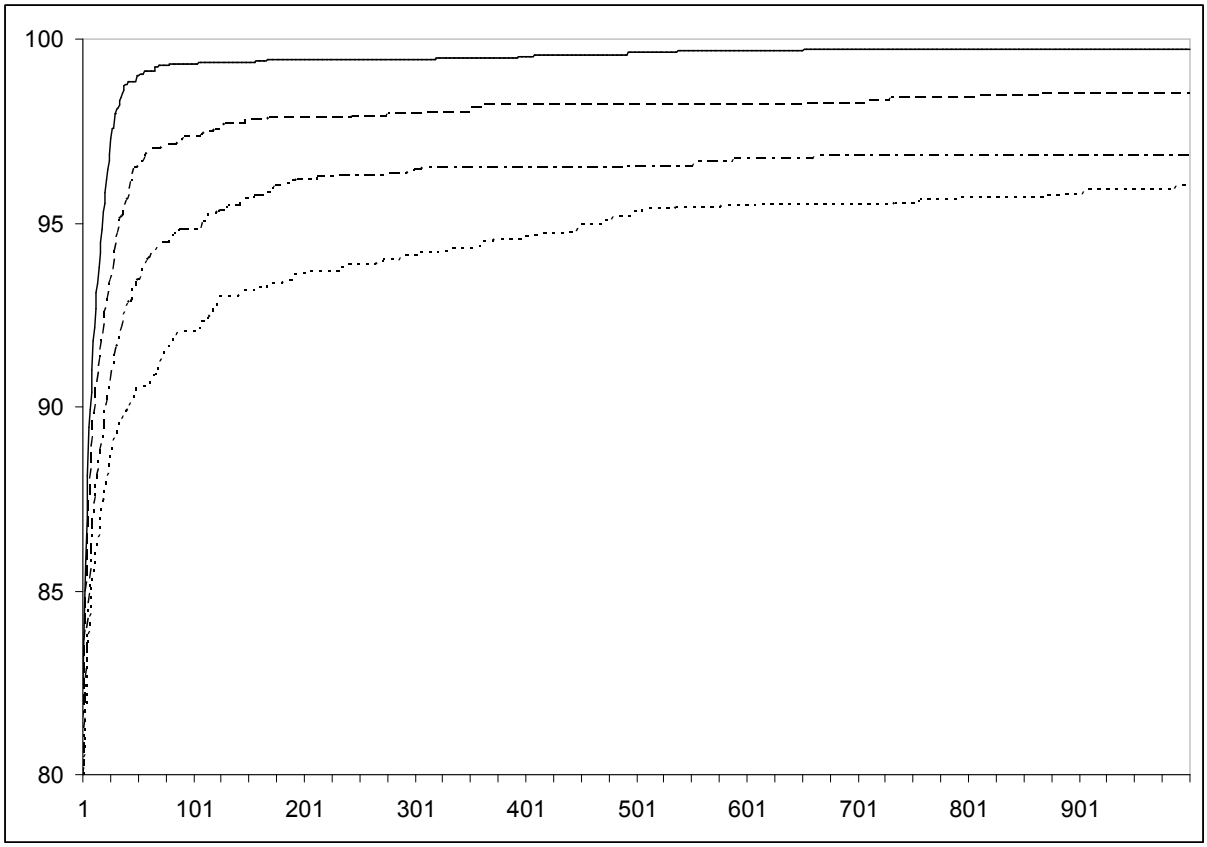


Рис. 72. Графики усредненных результатов для экспериментов над флибами с разным числом состояний и флагов

Необходимо отметить, что число состояний и флагов в данном эксперименте подобрано таким образом, что общее число переходов во всех флибах остается одинаковым.

Из приведенных таблиц и графиков следует, что применение автоматов с флагами для моделирования флибов позволяет строить более точные предсказатели.

Эксперименты показывают, что применение автоматов с флагами является более эффективным методом для улучшения точности работы генетического алгоритма по сравнению с простым увеличением числа состояний автоматов.

3.3.2.2. Эксперименты по применению алгоритма восстановления связей между состояниями

Эксперименты, описываемые в этом разделе, проводились с помощью генетического алгоритма, решающего задачу о флибах. При формировании нового поколения применялся турнирный отбор [32] и принцип элитизма [14] (в новое поколение добавляется одна или несколько лучших особей из предыдущего поколения). Для формирования новой особи использовался одноточечный оператор скрещивания. Все эксперименты проводились при размере поколения 100 и вероятности применения одноточечного оператора мутации 0,03. Число воздействий среды на флиб выбрано равным 100. Поэтому число правильно предсказанных символов по величине равно точности предсказания символов в процентах. В таблицах с результатами экспериментов приводится минимальные, максимальные и средние точности предсказания автоматически построенных флибов.

На графиках по оси абсцисс указаны номера поколений, по оси ординат – число правильно предсказанных символов лучшим предсказателем в каждом поколении. При построении графиков использовались **усредненные данные**, полученные при проведении 50 экспериментов с одинаковыми начальными параметрами. Пунктир использован для графиков, соответствующих стандартному генетическому алгоритму. Для случая, когда применяется алгоритм восстановления связей между состояниями, графики построены непрерывными линиями.

Ниже приводятся результаты трех экспериментов, которые отличаются между собой выбранным числом поколений, битовой маской и числом состояний флиба.

Первый эксперимент. Эксперимент производился для 400 поколений. Битовая маска, задающая среду, имеет вид – `11110100101111101001`. Число состояний флиба – 20.

Графики этого эксперимента приведены на рис. 73.

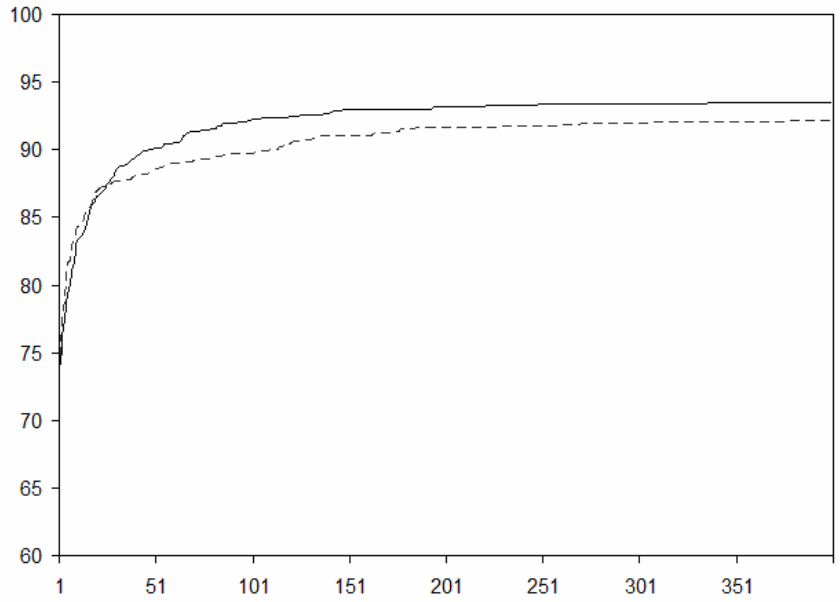


Рис. 73. Графики усредненных результатов для первого эксперимента по применению алгоритма восстановления связей между состояниями

Из приведенных графиков видно, что использование алгоритма восстановления связей между состояниями позволяет повысить среднюю эффективность генетического алгоритма.

Результаты первого эксперимента приведены в табл. 7.

Таблица 7. Результаты первого эксперимента по применению алгоритма восстановления связей между состояниями

Восстановление связей между состояниями	Худший результат	Усредненный результат	Лучший результат
Нет	83	92,26	100
Да	84	93,48	100

Второй эксперимент. Эксперимент производился для 400 поколений. Битовая маска, задающая среду, имеет вид – `111101001011110`. Число состояний флиба – 10. Отметим, что битовая маска в этом эксперименте короче, чем в предыдущем.

Графики для второго эксперимента приведены на рис. 74.

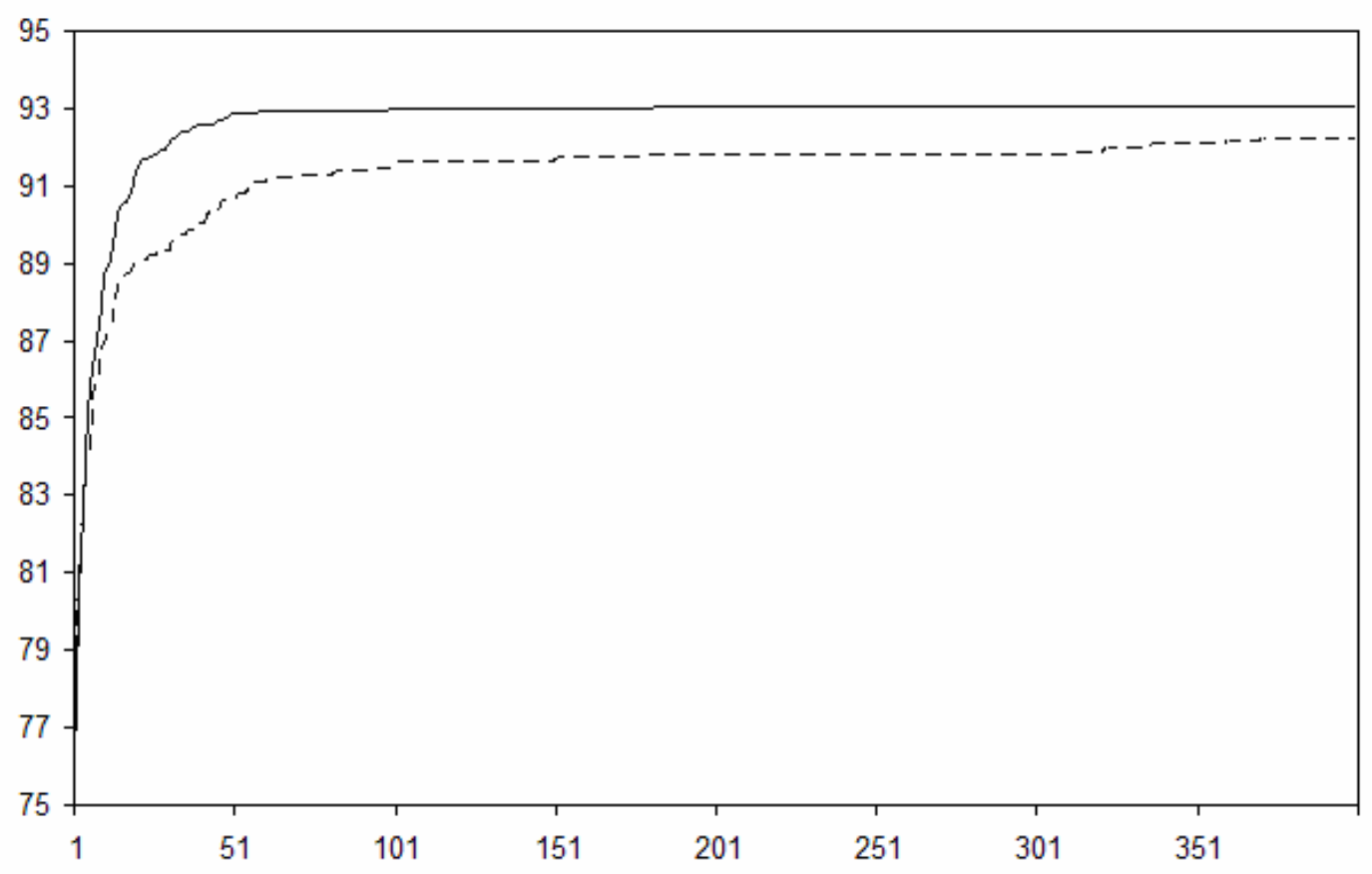


Рис. 74. Графики усредненных результатов для второго эксперимента по применению алгоритма восстановления связей между состояниями

Результаты второго эксперимента приведены в табл. 8.

Таблица 8. Результаты второго эксперимента по применению алгоритма восстановления связей между состояниями

Восстановление связей между состояниями	Худший результат	Усредненный результат	Лучший результат
Нет	86	92,2	94
Да	87	93,06	94

Значения, приведенные в столбце "Усредненный результат", соответствуют крайним правым точкам графиков на рис. 74.

Третий эксперимент. Эксперимент производился для 1600 поколений. Битовая маска, задающая среду, имеет вид – `10101111011000111101111110011001`. Число состояний флиба – 30. Отметим, что битовая маска в этом эксперименте содержит большее число разрядов, чем в обоих предыдущих.

Графики этого эксперимента приведены на рис. 75.

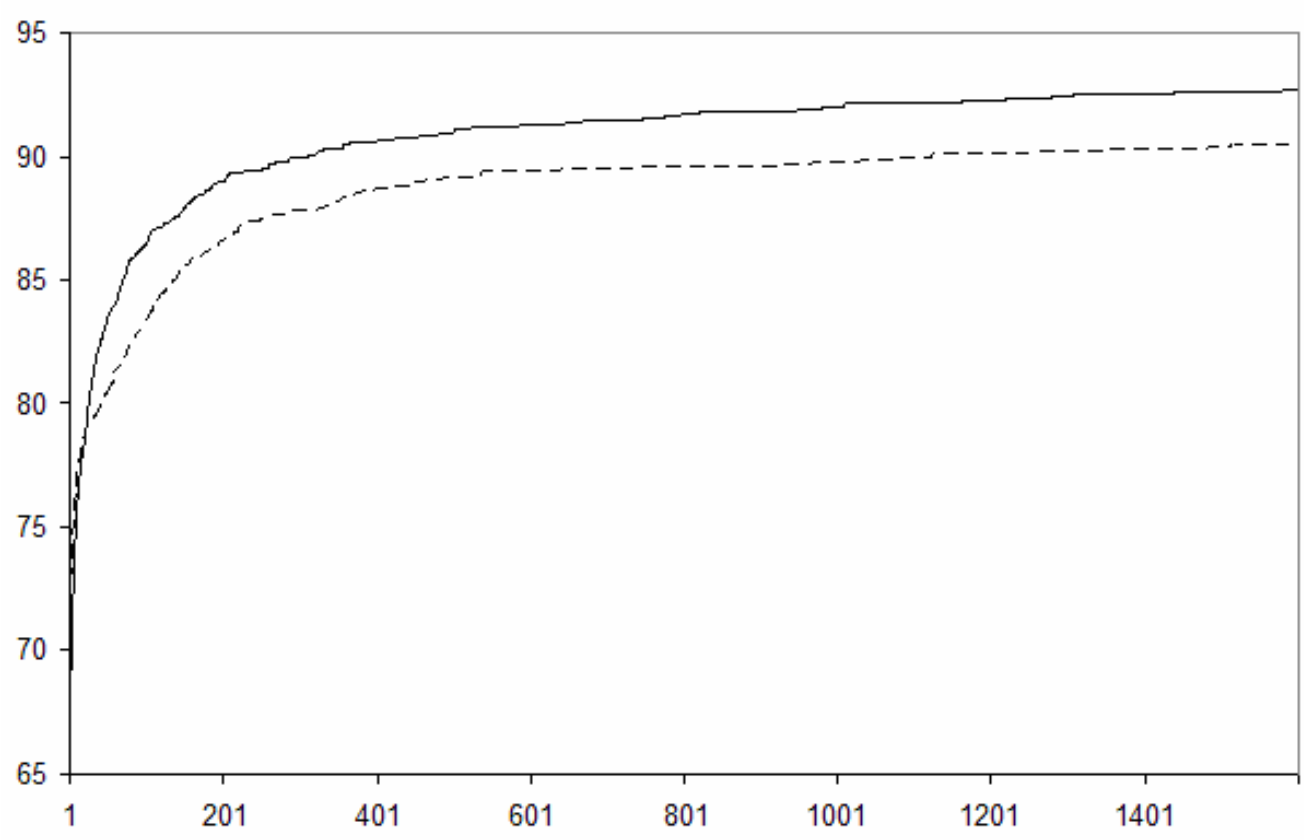


Рис. 75. Графики усредненных результатов для третьего эксперимента по применению алгоритма восстановления связей между состояниями

Результаты третьего эксперимента приведены в табл. 9. Выигрыш от применения алгоритма восстановления связей между состояниями в данном эксперименте больше, чем в двух предыдущих, благодаря увеличению числа поколений и усложнению задачи.

Таблица 9. Результаты второго эксперимента по применению алгоритма восстановления связей между состояниями

Восстановление связей между состояниями	Худший результат	Усредненный результат	Лучший результат
Нет	83	90,44	97
Да	85	92,72	97

Проведенные эксперименты показали, что алгоритм восстановления связей между состояниями повышает эффективность работы генетического алгоритма как при различных битовых масках, задающих среду, так и различном числе состояний флиба.

3.3.2.3. Эксперименты по применению алгоритма сортировки состояний в порядке использования

Эксперименты, описываемые в этом разделе, проводились с помощью генетического алгоритма, решающего задачу об умном муравье. При формировании нового поколения применялся отбор отсечением [22]. Для формирования новой особи, как и в работе [22], использовался *n*–точечный оператор скрещивания и *n*–точечный оператор мутации.

Все эксперименты, о которых идет речь ниже, проводились при размере поколения 10000 и пороге отсечения 2000. Вероятность применения оператора скрещивания к элементу автомата выбрана равной 0.04, а вероятность применения оператора мутации – 0.02. Число состояний автомата было ограничено двадцатью. Эксперименты проводились на персональном компьютере с процессором *AMD Athlon* 3800+.

В табл. 10 приведены результаты экспериментов, в которых использовалась целевая функция из работы [22]. Число поколений в экспериментах, в которых использовались предложенные в настоящей работе операторы мутации, было выбрано равным 100. Число поколений в экспериментах только с *n*–точечным оператором мутации не превышало двухсот. Каждый эксперимент без дополнительных операторов мутации длился примерно 310 с (время построения двухсот поколений). Продолжительность каждого эксперимента с дополнительными операторами мутации составила около 270 с (время построения ста поколений).

В столбце «Номер поколения» таблицы приведен номер поколения, в котором был найден лучший автомат в эксперименте. В столбце «Результат» приведено число ячеек с едой, съеденных лучшим муравьем в эксперименте. Число **используемых состояний** лучшего муравья, полученного в эксперименте, приведено в столбце «Число используемых состояний». В столбце «Время поиска» приводится время в секундах, прошедшее с начала эксперимента, до построения поколения, в котором был найден лучший автомат в эксперименте.

Таблица 10. Результаты экспериментов по применению алгоритма сортировки состояний в порядке использования

	Номер эксперимента	Номер поколения	Результат	Число используемых состояний	Время поиска
Без применения алгоритма сортировки состояний в порядке использования	1	137	87	15	225
	2	180	88	15	218
	3	73	88	12	137
	4	72	86	12	114
	5	68	89	12	114
С применением алгоритма сортировки состояний в порядке использования	1	47	89	12	154
	2	49	89	11	160
	3	56	89	13	182
	4	51	89	12	145
	5	67	89	13	189

Как следует из нижней части табл. 10, использование предложенных операторов мутации позволило при всех запусках найти автоматы, решающие поставленную задачу. Известный алгоритм (верхняя часть табл. 10) с этим не справился в четырех экспериментах из пяти, даже породив вдвое большее число поколений. Это, по всей видимости, связано с тем, что в данных экспериментах размер поколения был значительно меньше размера поколения, использованного в работе [22].

3.4. Программное средство для поддержки композитного генетического программирования

В данном разделе описывается предлагаемое программное средство для поддержки композитного генетического программирования.

Приведем назначение основных классов и интерфейсов программного средства. В скобках после названия интерфейса указана его конкретизация для *DCT*:

`Server` (`DCTServer`) – сервер, который хранит репозиторий. Наследуется от интерфейса `java.rmi.Remote`, что позволяет организовать архитектуру клиент-сервер.

`Client` (`CAClient`: `GEP`, `GA`, `GP`) – модуль, содержащий соответствующий алгоритм решения задачи.

`Fitness` (`CAFitness`) – функция приспособленности, вычисляемая для решений задачи;

`Point`, `AbstractPoint` (`CA`) – решение задачи (элемент популяции);

`Repository` (`CARepository`) – репозиторий решений (наследуется от интерфейса `java.rmi.Remote`);

`PointMetadata` – метаданные решений (значение функции приспособленности решения, дата добавления в репозиторий, имя модуля, добавившего данное решение и т. д.);

`SimplePointMetadata` – стандартная реализация интерфейса `PointMetadata`;

`DownloadDesire` – запрос (“пожелание”) модуля при обращении к репозиторию для получения решений;

`SlightlyBetterPointDesire` – запрос о предоставлении модулю решений, чуть лучших, чем его собственные.

3.4.1. Модель программного средства

На рис. 76, .77 приведена упрощенная *UML*-диаграмма классов. На ней отображены только основные интерфейсы и классы, а также их ключевые методы. На этих диаграммах не показан ряд классов, например, отвечающих за поддержку коэволюций.

Перечислим отдельно интерфейсы, имплементации интерфейсов, которые не связаны с решением *DCT*, и классы, реализующие интерфейсы для *DCT*.

Интерфейсы: `Remote, Fitness, Server, Client, Repository, Point, PointMetadata, DownloadDesire`.

Имплементации интерфейсов, которые не привязаны к *DCT* и могут быть использованы при имплементации решения любой другой задачи методом *композитного генетического программирования*: `AbstractPoint, SimplePointMetadata, SlightlyBetterPointDesire`.

Классы, которые реализуют интерфейсы для *DCT*: `CAFitness, DCTServer, CAClient, GEP, GA, GP, CARepository, CA.`

3.4.2. Прототип программного средства

Для реализации предложенного средства выбран язык *Java* и технология *Remote Method Invocation (RMI)* [39]. После реализации была произведена проверке эффективности метода при решении задачи классификации плотности.

Характеристики компьютера, на котором выполнялись эксперименты: процессор *Intel Pentium M* 1.86 *GHz*, RAM – 512M.

Результаты экспериментов и их анализ. Популяция каждого из эволюционных алгоритмов состоит из 100 особей (правил). Вычисление функции приспособленности выполняется на 10^5 начальных конфигураций. За несколько часов работы разработанного средства, реализующего композитное генетическое программирование, для рассматриваемой задачи было сгенерировано большое число хороших правил, для лучших из которых значения функции приспособленности приведены в табл. 11.

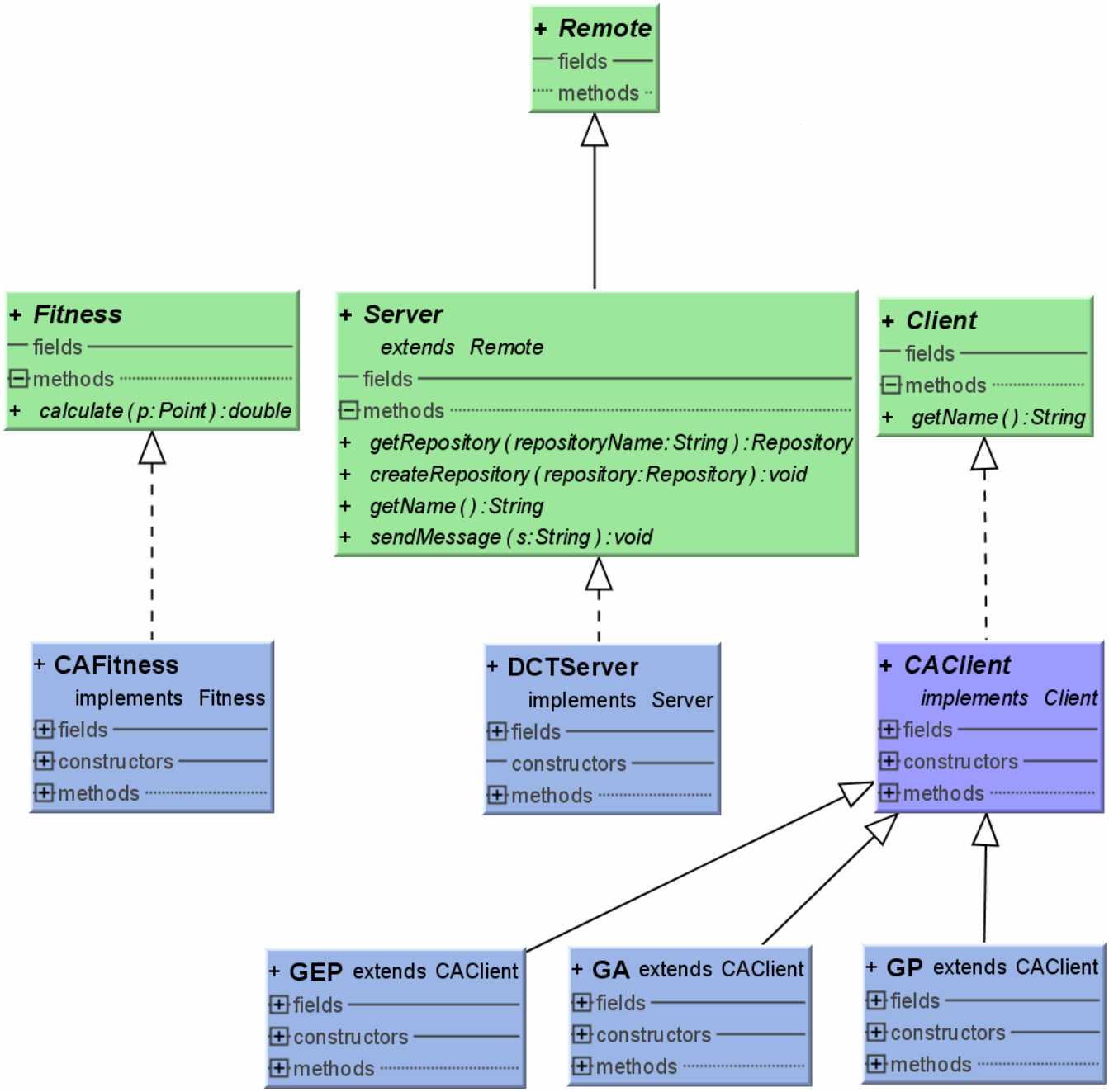


Рис. 76. Диаграмма классов

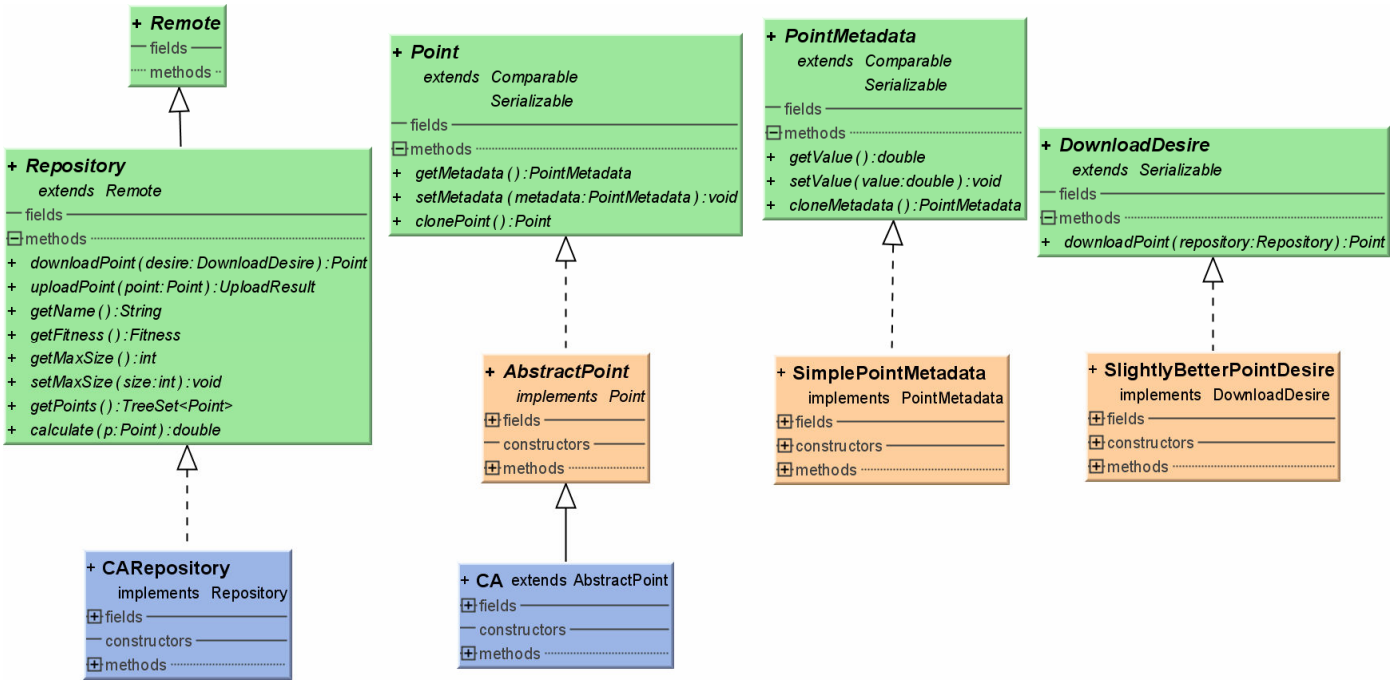


Рис. 77. Диаграмма классов (продолжение)

Таблица 11. Столбцы значений таблиц истинности найденных правил

Столбец значений	Результат
00000001000101000001000011010111000000010000111100111001010101110000110101110101111111110001011111010001001111011111100101010111	0.837
00000001000101000011000011011111000000010000111100111001000101110000101101101011111111100010111101110001001111011111100101010111	0.833
00000001000101000011000011010111000000010000111100111001010101110000111101110100111111110001011110111000100111011111100101010111	0.831
00000000000001010011001100100111000000000101010100110011011101110000111100001010000011110010011111111111101010101111111111011101111	0.826
00000001000101000000000001101011100000001000011110011100101010110000110110110100111111110001011111110001001111111111100101010111	0.826

Ни в работе [15], ни в работе [16] не было получено столь высоких результатов, при том, что в первой из этих работ популяция состояла из 51200 правил, а вычисления осуществлял 64-процессорный компьютер, который решал эту задачу более месяца. В работе [23] поиск лучшего из известных правил с результатом 0.86 выполнялся в течение недели, а популяция решений состояла из 1000 особей. Из изложенного следует, что **метод композитного генетического программирования обеспечивает весьма высокий результат решения задачи классификации плотности при сравнительно небольших вычислительных затратах**.

В следующем разделе приводится описание того, каким образом метод композитного генетического программирования применить для решения модифицированной задачи об “Умном муравье”.

3.4.3. Результаты экспериментов

На основе созданного прототипа программного средства композитного генетического программирования были проведены эксперименты, показывающие эффективность метода композитного генетического программирования. В качестве задачи, на которой проводились эксперименты была выбрана задача об “Умном муравье”ю

3.4.3.1. Постановка задачи

Постановка задачи об “Умном муравье” приведена в разд. 1.1.4.1 отчета по первому этапу темы [51]. В указанном разделе приводятся несколько возможных решений данной задачи – алгоритмов управления муравьем, съедающим все 89 яблок за 200 ходов. Таким образом, задача об “Умном муравье” в классической постановке является решенной. Сравнительная простота задачи связана с тем, что расположение яблок на поле фиксировано, а размеры поля и число яблок достаточно малы, что позволяет найти решение задачи за приемлемое время (не превышающее нескольких часов). Число входных воздействий, на которые реагирует муравей также мало: яблоко расположено перед муравьем или перед муравьем яблока нет.

Поэтому интересным представляется обобщение задачи об “Умном муравье”, при котором расположение яблок на поле не является заранее фиксированным, а число входных воздействий увеличено. В отличие от классической задачи об “Умном муравье”:

- распределение яблок на поле является вероятностным. При этом вероятность яблока оказаться в некоторой клетке одинакова для всех клеток поля и равна μ;
- обзор муравья расширен – он одновременно видит восемь клеток поля (не только непосредственно перед собой). На рис. 78 клетка, в которой находится муравей, закрашена. Муравей ориентирован в направлении стрелки. Таким образом, муравей реагирует на $2^8 = 256$ входных воздействий.

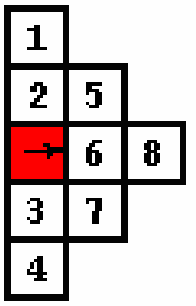


Рис. 78. Обзор муравья

В такой постановке задачи число яблок, съеденных муравьем за 200 ходов, является случайной величиной ξ, заданной на дискретном множестве элементарных исходов Ω – множестве расположений яблок (битовых матриц размера 32 на 32). Каждому исходу ω_i, содержащему k единиц (яблок), поставим в соответствие вероятность данного исхода $p(\omega_i) = \mu^k(1-\mu)^{n-k}$, где $n = 32 \times 32$.

Задача: создать муравья, для которого математическое ожидание числа съеденных яблок $E[\xi]$ принимает максимальное значение.

Задача в этой постановке не является тривиальной. Поэтому рассмотрим один из возможных алгоритмов поведения муравья (рис. 79).

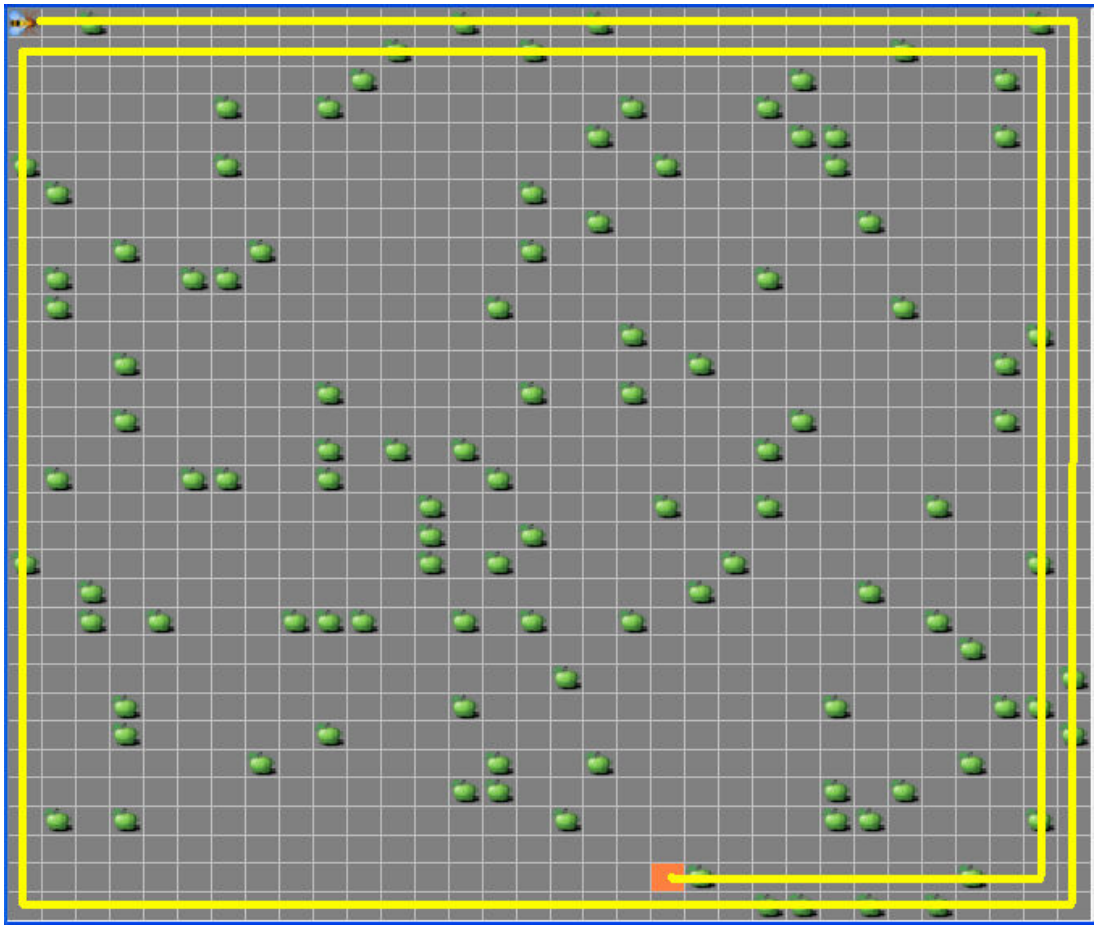


Рис. 79. Возможное решение задачи

Траектория его движения– ломаная. Муравей совершает 200 ходов. При этом математическое ожидание числа съеденных яблок

$$E[\xi] = (200 - \text{число поворотов}) * \mu = (200 - 200 / 32) * \mu = 194\mu.$$

Данный алгоритм не приводит к оптимальному решению, так как известны решения, получаемое с помощью одного из эволюционных алгоритмов, с более высоким статистически вычисленным математическим ожиданием числа съеденных яблок [47].

3.4.3.2. Методы решения задачи эволюционными алгоритмами

Для решения поставленной задачи методом композитного генетического программирования программное средство должно содержать четыре модуля, каждый из которых используют один из эволюционных алгоритмов. В этих алгоритмах автоматы задаются в виде:

- деревьев решений;
- битовых строк;
- графов переходов;
- *Karva*-деревьев.

Приведем описание методов решения задачи, которые основаны на указанных представлениях автоматов. Метод, основанный на представление автоматов с помощью деревьев решений, описан в

разд. 1.2 настоящего отчета, а метод, использующий битовые строки – в разд. 1.4.1.1 отчета по первому этапу темы [51]. Ниже приведено описание двух оставшихся подходов.

3.4.3.3. Представление автоматов в виде графа переходов

Естественным представлением автомата является его представление в виде графа переходов. Это подход обладает и тем преимуществом, что генетические операции определяются для высокоуровневого представления, и поэтому просты и понятны. В общем виде описание генетических операций над графами приведено в разд. 2.1.2.3 отчета по первому этапу темы [51].

В рассматриваемом методе будем представлять условия на переходе в виде битовых строк длины восемь. Каждая позиция в строке соответствует требованию к наличию яблока в соответствующей клетке наблюдаемой области. Действия определим естественным образом – идти вперед, повернуть налево, повернуть направо, стоять на месте. Из каждого состояния будет существовать переход (с определенным действием) по каждому из 256 входных воздействий. На рис. 80 приведен фрагмент автомата в виде графа, на котором изображены лишь некоторые из 256 возможных переходов.

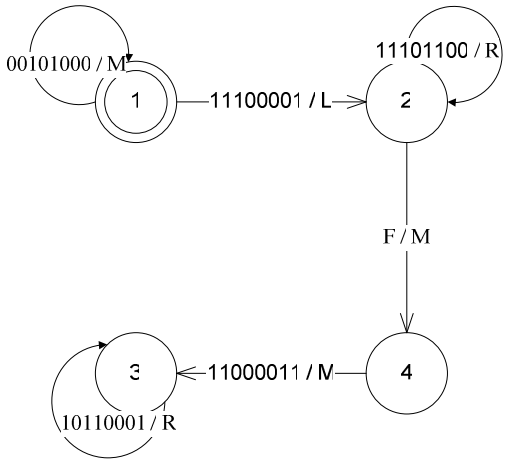


Рис. 80. Пример автомата в графовом представлении

3.4.3.4. Представление автоматов в виде *Karva*-деревьев

Этот метод основан на программировании с экспрессией генов [16] – разновидности эволюционных алгоритмов, обладающей высокой эффективностью. В отличие как от стандартных генетических алгоритмов, использующих строки фиксированной длины, так и от генетического программирования, использующего деревья, хромосомы в генетическом программировании с экспрессией генов представляются в виде строк фиксированной длины, которые преобразуются в *Karva*-деревья.

Предлагаемый метод основан на представления автомата с помощью *Karva*-дерева. Рассмотрим в качестве примера автомат на рис. 81, у которого входные воздействия {N, F}, а действия – {M, L, R}.

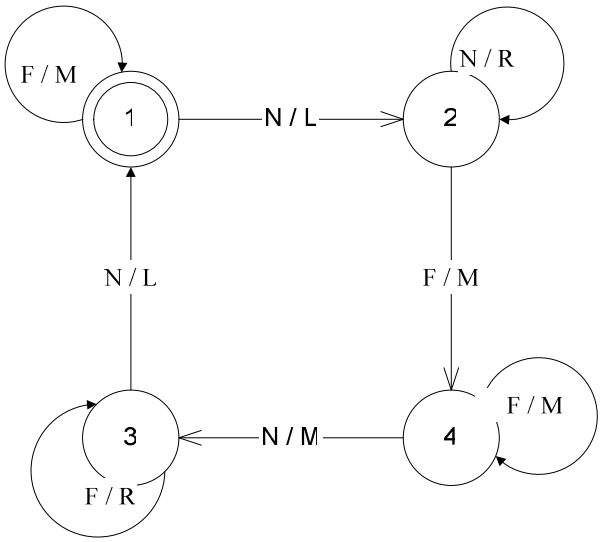


Рис. 81. Пример автомата

Представление этого автомата в виде *Karva*-дерева приведено на рис. 82.

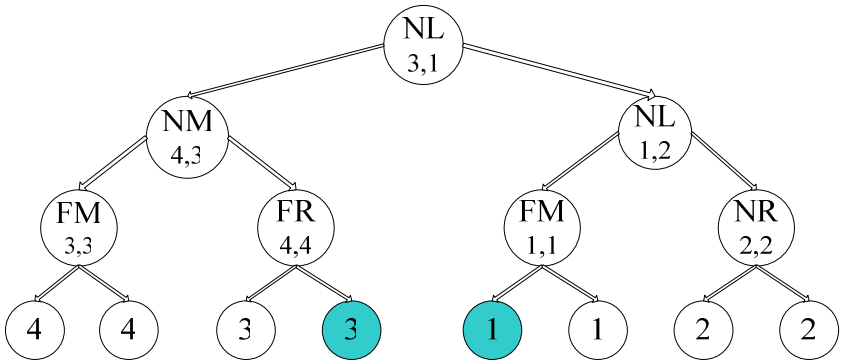


Рис. 82. Karva-дерево, задающее автомат

Вершиной этого дерева является либо функция, либо терминал. Множество терминалов – множество состояний автомата {1, 2, …, *n*}. В данном примере – {1, 2, 3, 4}. Множество функций определено следующим образом: каждой паре (входное воздействие v_i, действие z_j) сопоставим функцию $f_{ij}(x, y)$ двух аргументов. В данном примере множество функций – {NL, NR, NM, FL, FR, FM}. Значение функции – автомат, являющийся объединением автоматов *x* и *y*, в котором также задан переход из состояния *s* в состояние *t* по входному воздействию v_i с выполнением действия z_j. Здесь *s* – самый правый терминал в левом поддереве, *t* – самый левый терминал в правом поддереве. На рис. 82 в каждой вершине значения *s* и *t* приведены через запятую. Дерево линеаризуется в строку, стандартным для *Karva*-деревьев способом, который изложен в работе [16]. Строка, соответствующая дереву, изображенному на рис. 82, приведена в табл. 12.

Таблица 12. Линеаризованное Karva-дерево

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
NL	NM	NL	FM	FR	FM	NR	4	4	3	3	1	1	2	2

При указанном представлении решения в виде хромосомы, удается корректно и просто выполнять генетические операции над строками фиксированной длины, которые экспрессируются в сложные структуры – деревья.

В результате повышение быстродействия обеспечивается за счет применения программирования с экспрессией генов, а проблема большого числа входных воздействий решается за счет того, что из состояния автомата задаются переходы не для всех входных воздействий. В случае, когда реакция на входное воздействие не определена, осуществляется некоторое стандартное действие (например, пойти вперед) и автомат не изменяет состояния.

3.4.4. Универсальная функция приспособленности и универсальная форма решений

Универсальную функцию приспособленности определим как среднее число яблок, съеденных муравьем за определенное число игр. Каждая игра характеризуется некоторым случайным расположением яблок на поле. Из соображений малости дисперсии результата игры (измеренной статистически) число игр выбрано равным 1000.

В качестве универсальной формы решения предлагается использовать представление автомата в виде графа, но условия на переходах представлять не в виде битовых строк, а в виде строк длины восемь над алфавитом {0,1,?}. Здесь символ 1 в *i*-ой позиции строки, означает, что яблоко должно находиться в *i*-й клетке, символ 0 – что оно должно отсутствовать, а символ ? допускает оба варианта. Входное воздействие, изображенное на рис. 83, удовлетворяет условиям: 0000**11**00, 0000**?1**0?, ?000**??**0? и т.д. С другой стороны, одному условию могут удовлетворять несколько входных воздействий. Так условию ?????1?? удовлетворяют все входные воздействия, отвечающие ситуации, когда в клетке перед муравьем находится еда.

Рис. 83. Пример события

Поскольку входное воздействие может удовлетворять сразу нескольким условиям на переходах, необходимо задать на этих переходах порядок. При этом для каждого состояния определен переход по умолчанию. Он выполняется в случае, если не были выполнены условия ни на одном из других переходов, определенных для данного состояния. В соответствии с изложенным, автомат, описывающий поведение муравья, представляется в универсальной форме следующим образом: *n* – число состояний графа переходов. Далее для каждого из *n* состояний следует число *m* – число переходов из данного состояния. Далее приводится *m* описаний переходов в виде тройки:

<условие перехода, состояние, действие>. Кроме того, будем считать, что начальное состояние автомата имеет номер ноль.

Приведем пример записи автомата в универсальной форме. На рис. 84 изображен некоторый автомат.

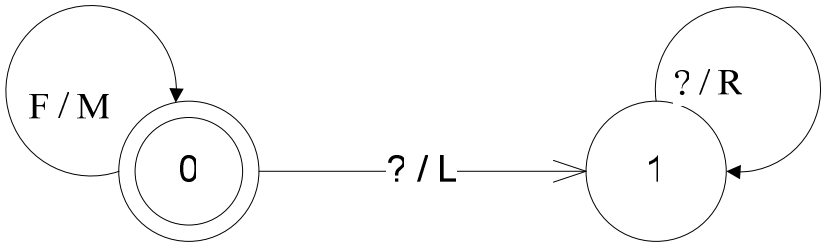


Рис. 84. Пример автомата

В соответствии с приведенным описание универсальной формы, запись автомата, изображенного на рис. 84, имеет в универсальной форме следующий вид:

```
2 2 ?????0?? 1 1 ???????? 0 0 1 ???????? 2 1
```

Для того чтобы выполнить перевод автомата из формы, характерной для каждого из четырех эволюционных алгоритмов, в универсальную форму, достаточно представить автомат в виде графа переходов, а затем воспользоваться указанным алгоритмом записи автомата.

3.5. Методические рекомендации и указания по разработке автоматов управления систем со сложным поведением

Для эффективной генерации автоматов управления системами со сложным поведением могут быть применены все методы, разработанные в данной работе. Схема их применения показана на рис. 85.

Рис. 85. Применение разработанных методов генерации автоматов управления системами со сложным поведением

При этом базовая версия алгоритма генетического программирования строится на основе сокращенных таблиц переходов или представления автоматов деревьями решений. Первый вариант выбирается в случае малого числа входных переменных или необходимости условий переходов, включающих большое число входных переменных. Второй вариант более эффективен в случае большого числа входных переменных, если при этом на каждом переходе используется малое их число. Оба эти подхода эффективнее стандартного представления автоматов битовыми строками, так как в них автоматные хромосомы имеют меньшую длину, что позволяет сократить пространство поиска.

Базовая версия алгоритма оптимизируется методами, описанными в разд. 1.3. При этом за счет отождествления автоматов с одинаковым внешнем поведением достигается дальнейшее сокращение пространства поиска.

При решении задач методами генетического программирования часто создается множество алгоритмом, отличающихся методом представления хромосом и/или функцией приспособленности. Если такие алгоритмы имеют разную зависимость скорости приближения к оптимуму от числа поколений или прошедшего времени, то для них может использоваться композитное генетическое программирование. При этом у алгоритмов будут заимствованы наиболее эффективные участки, что позволит снизить время получения требуемого решения.

Таким образом, технология генетического программирования для генерации автоматов управления системами со сложным поведением может быть сформулирована в виде последовательности этапов:

1. Постановка задачи.
2. Идентификация входных переменных.
3. Идентификация выходных воздействий.
4. Выбор функции приспособленности.
5. Выбор способа представления хромосом.
6. Применение методов оптимизации к выбранному способу представления.

7. Реализация полученного алгоритма и численная оценка его эффективности.
8. При недостаточной эффективности повторение шагов 4-7.
9. Комбинирование алгоритмов с помощью композитного генетического программирования.

На первом этапе производится анализ решаемой задачи, определяются критерии оптимальности решения.

На втором и третьем этапах рассматриваются кандидаты во входные переменные и выходные воздействия. Их число желательно минимизировать. Отметим, что увеличение числа входных переменных приводит к большему увеличению времени работы, чем для выходных воздействий.

На четвертом этапе выбирается функция приспособленности, которая должна соответствовать критериям оптимальности, сформулированным на первом этапе.

В зависимости от числа входных переменных и ожидаемой сложности условий на переходах выбирается один из способов кодирования автоматных хромосом. В случае одной и двух входных переменных может быть использовано непосредственное представление, как наиболее простое. При большом числе входных переменных, если условия на переходах просты следует выбрать метод представления автоматов деревьями решений, в противном случае — метод сокращенных таблиц переходов.

При необходимости, на шестом этапе применяются методы оптимизации. Наиболее простым в реализации является алгоритм сортировки состояний в пространстве использования. Большая эффективность может быть достигнута за счет комбинации его с алгоритмом восстановления связей между автоматами. Автоматы с флагами должны применяться, если специфика задачи предполагает похожее поведение в различных состояниях.

На седьмом этапе полученный алгоритм реализуется (возможно, с применением прототипов, описанных в главе 3) и запускается. При этом оценивается время, необходимое для достижения решения с требуемой степенью оптимальности. Если это время слишком велико, то повторяются шаги с четвертого по седьмой. При этом создается несколько версий алгоритма, которые обычно обладают разной скоростью приближения к оптимуму на различных участках. Это позволяет их комбинировать с помощью композитного генетического программирования.

Применение технологии генетического программирования для генерации автоматов управления системами со сложным поведением, основанной на разработанных методах, должно позволить получать автоматные системы управления, которые могут использоваться для практических задач.

ЗАКЛЮЧЕНИЕ

В результате исследований, выполненных на втором этапе работ по контракту, были разработаны методы генерации автоматов управления. Для предложенных методов были определены, функциональные особенности и характеристики. Также были разработаны прототипы программных средств, реализующих разработанные методы, и сформулирована технология генетического программирования для генерации автоматов управления системами со сложным поведением.

В первой главе были разработаны методы генерации автоматов управления системами со сложным поведением. Разработанные методы позволяют существенно сократить размерность пространства поиска оптимальных решений для задач генерации автоматов. Таким образом, они обеспечивают возможность более эффективного решения таких задач.

Исследования, выполненные во второй главе, позволили оценить теоретическую эффективность разработанных методов и области их применения. Разработаны рекомендации, основанные рекомендации по применению указанных методов.

Практическая реализация разработанных методов позволила подтвердить их эффективность при применении к различным задачам. На основе полученных данных были сформулированы более полные рекомендации по применению разработанных методов, была предложена технология генетического программирования для генерации автоматов управления системами со сложным поведением.

Таким образом, были получены решения всех задач, поставленных в техническом задании на проведение второго этапа работы.

Результаты выполненных работ, а также патентных исследований, позволяют утверждать, что научно-технический уровень исследований соответствует уровню исследований в рассматриваемой области, проводимых в лучших исследовательских центрах мира.

ИСТОЧНИКИ

1. *Andre D., Bennet F., Koza J.* Discovery by Genetic Programming of a Cellular Automata Rule that is Better than any Known Rule for the Majority Classification Problem. 1996. http://citeseer.ist.psu.edu/andre96discovery.html
2. *Angeline P., Pollack J.* Evolutionary Module Acquisition / Proceedings of the Second Annual Conference on Evolutionary Programming. Cambridge: MIT Press. 1993, pp.154–163. http://www.demo.cs.brandeis.edu/papers/ep93.pdf
3. *Ashlock D.* Evolutionary Computation for Modeling and Optimization. New York: Springer, 2006.
4. *Ashlock D. A., Emrich S. J., Bryden K. M. and others* A comparison of evolved finite state classifiers and interpolated markov models for improving PCR primer design / 2004 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB'04). 2004, p. 190–197.
5. *Ashlock D., Wittrock A., Wen T-J.* Training finite state machines to improve PCR primer design / Congress on Evolutionary Computation (CEC'02). 2002, pp. 13–18.
6. *Banzhaf W., Nordin P., Keller R. E., Francone F. D.* Genetic Programming – An Introduction. On the automatic Evolution of Computer Programs and its Application. San Francisco: Morgan Kaufmann Publishers, 1998.
7. *Belz A.* Computational Learning of Finite-State Models for Natural Language Processing. PhD thesis. University of Sussex. 2000.
8. *Belz A., Eskikaya B.* A genetic algorithm for finite state automata induction with an application to phonotactics / ESSLLI-98 Workshop on Automated Acquisition of Syntax and Parsing. Saarbruecken, 1998, pp. 9–17.
9. *Bryant K.* Genetic Algorithms and the Traveling Salesman Problem. Harvey Mudd College: Department of Mathematics, 2000. http://www.math.hmc.edu/math197/archives/2001/kbryant/kbryant-2001-thesis.pdf
10. *Chambers L.* Practical Handbook of Genetic Algorithms. Complex Coding Systems. Volumes I. II, III. CRC Press, 1999.
11. *Clelland C. H., Newlands D. A.* Pfsa modelling of behavioural sequences by evolutionary programming / Complex'94 – Second Australian Conference on Complex Systems. IOS Press. 1994, pp. 165-172.
12. *Coley D.*A. An Introduction to Genetic Algorithms for Scientists and Engineers. World Scientific Publishing Company, 1997. 227 p.
13. *Das S., Mozer M. C.* A Unified Gradient-Descent/Clustering Architecture for Finite State Machine Induction / Advances in Neural Information Processing Systems. 1994.
14. *De Jong K.* An analysis of the behavior of a class of genetic adaptive systems. PhD thesis. Univ. Michigan. Ann Arbor, 1975.
15. *Ferreira C.* Gene Expression Programming: A New Adaptive Algorithm for Solving Problems /Complex Systems. 2001. Vol. 13, issue 2, pp. 87–129. http://arxiv.org/pdf/cs.AI/0102027.pdf
16. *Fogel B., Fogel J., Atmar W.* Meta-Evolutionary Programming. In Chen, R. (ed.) / Proceedings of the 25th Asilomer Conference on Signals, Systems and Computers, CA: Maple Press. 1991. pp. 540-545. http://citeseer.ist.psu.edu/context/24073/0
17. *Frey C., Leugering G.* Evolving Strategies for Global Optimization. A Finite State Machine Approach / Genetic and Evolutionary Computation Conference (GECCO-2001). Morgan Kaufmann, 2001, pp. 27–33.
18. *Gold E. M.* Language Identification in the Limit // Information and Control. 1967. №10. pp. 447–474.

19. *Gustafson S.* An Analysis of Diversity in Genetic Programming. Ph.D. Dissertation. School of Computer Science and Information Technology. University of Nottingham. Nottingham. U.K., 2004. http://citeseer.ist.psu.edu/gustafson04analysis.html
20. *Huang D.* MS Thesis Preproposal: Adaptive Incremental Fitness Evaluation in Genetic Algorithms. 2005. NY: Rochester. http://www.cs.rit.edu/~dxh6185/downloads/MS_Thesis/Documents/Presentation.pdf
21. *Jefferson D., Collins R., Cooper C., Dyer M., Flowers M., Korf R., Taylor C., Wang A.* The Genesys System: Evolution as a Theme in Artificial Life /Proceedings of Second Conference on Artificial Life. MA: Addison-Wesley. 1992, pp.549–578. www.cs.ucla.edu/~dyer/Papers/AlifeTracker/Alife91Jefferson.html
22. *Jefferson. D., Collins. R., Cooper. C., Dyer. M., Flowers. M., Korf. R., Taylor. C., Wang. A.* Evolution as a theme in artificial life: The genesys/tracker system / Artificial Life II: Proceedings of the Workshop on Artificial Life. In Langton: Addison-Wesley. 1992, pp. 549-578.
23. *Juillie H., Pollack J.* Coevolving the "Ideal" Trainer: Application to the Discovery of Cellular Automata Rules. 1998. http://citeseer.ist.psu.edu/16712.html
24. *Kantschik W., Dittrich P., Brameier M.* Empirical Analysis of Different Levels of Meta-Evolution / Congress on Evolutionary Computation. 1999.
25. *Kantschik W., Dittrich P., Brameier M., Banzhaf W.* Meta-Evolution in Graph GP / Genetic Programming: Second European Workshop (EuroGP'99). 1999.
26. *Koza J.* Genetic Evolution and Co-Evolution of Computer Programs / Proceedings of Second Conference on Artificial Life. Redwood City, CA: Addison-Wesley. 1992. pp. 603-629. http://citeseer.ist.psu.edu/177879.html
27. *Koza J.* Genetic programming. On the Programming of Computers by Means of Natural Selection. MA: The MIT Press, 1998.
28. *Koza J. R.* Future Work and Practical Applications of Genetic Programming. Handbook of Evolutionary Computation. Bristol: IOP Publishing Ltd, 1997.
29. *Lankhorst M. M.* A Genetic Algorithm for the Induction of Nondeterministic Pushdown Automata. Computing Science Report. University of Groningen Department of Computing Science. 1995.
30. *Linton R.* Adapting binary fitness functions in genetic algorithms / Proceedings of the 42nd annual Southeast regional conference. NY: ACM Press. 2004, pp. 391–395.
31. *Lucas S. M.* Evolving Finite State Transducers: Some Initial Explorations / Genetic Programming: 6th European Conference (EuroGP'03). Berlin: Springer. 2003, pp. 130–141.
32. *Miller B., Goldberg M.* Genetic algorithms, tournament selection, and the effects of noise // Complex Systems. 1995. V. 9, I. 3, pp. 193–212.
33. *Miller J. H.* The Coevolution of Automata in the Repeated Prisoner's Dilemma. Working Paper. Santa Fe Institute. 1989.
34. *Mitchell M., Crutchfield P., Hraber T.* Evolving cellular automata to perform computations. Physica D. 75. 1993, pp. 361–391. http://web.cecs.pdx.edu/~mm/mech-imped.pdf
35. *Paredis, J.* Coevolving cellular automata: Be aware of the red queen! / Proceedings of the Seventh International Conference on Genetic Algorithms. 1997, pp. 393–400. http://citeseer.ist.psu.edu/paredis97coevolving.html
36. *Petrovic P.* Comparing Finite-State Automata Representation with GP-trees. Technical report. Norwegian University of Science and Technology. 2006.
37. *Petrovic P.* Evolving automatons for distributed behavior arbitration. Technical Report. Norwegian University of Science and Technology. 2005.
38. *Petrovic P.* Simulated evolution of distributed FSA behaviour-based arbitration / The Eighth Scandinavian Conference on Artificial Intelligence (SCAI'03). 2003.

39. *Remote Method Invocation.* http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp
40. *Spears W. M., Gordon D. F.* Evolving Finite-State Machine Strategies for Protecting Resources / International Syposium on Methodologies for Intelligent Systems. 2000.
41. *Teller A., Veloso M.* Internal Reinforcement in a Connectionist Genetic Programming Approach // Artificial Intelligence. 2000.
42. *Teller A., Veloso M.* PADO: A New Learning Architecture for Object Recognition / Symbolic Visual Learning. New York: Oxford University Press, 1996, pp. 81–116.
43. *Werfel J., Mitchell M., Crutchfield J.* Resource Sharing and Coevolution in Evolving Cellular Automata. 1998. http://citeseer.ist.psu.edu/werfel99resource.html
44. *Whitley D., Rana S., Heckendorn R.* The Island Model Genetic Algorithm: On separability, Population Size and Convergence. 1998. http://citeseer.ist.psu.edu/whitley98island.html
45. *Бедный Ю.Д.* Применение генетических алгоритмов для построения клеточных автоматов. Бакалаврская работа. СПбГУ ИТМО. 2006. http://is.ifmo.ru/papers/genalgcelaut.pdf
46. *Бедный Ю.Д., Корнеев Г.А., Шалыто А.А.* Применение генетических алгоритмов для построения клеточных автоматов. http://is.ifmo.ru/papers/genalgcelaut.pdf
47. *Бедный Ю.Д., Шалыто А.А.* Применение генетических алгоритмов для построения автоматов в задаче «Умный муравей». СПбГУ ИТМО. 2007. http://is.ifmo.ru/works/ant/
48. *Гач П., Курдюмов Г., Левин Л.* Одномерные однородные среды, размывающие конечные острова // Проблемы передачи информации. 1976. 14, 3.
49. *Гладков Л. А., Курейчик В. В., Курейчик В. М.* Генетические алгоритмы. М.: Физматлит. 2006.
50. *Лобанов П. Г., Шалыто А. А.* Использование генетических алгоритмов для автоматического построения конечных автоматов в задаче о “Флибах” / Сборник докладов 4-й Всероссийской научной конференции «Управление и информационные технологии» (УИТ-2006). СПбГЭТУ «ЛЭТИ». 2006. с.144–149. http://is.ifmo.ru/works/flib
51. *Промежуточный отчет по этапу I* "Выбор направления исследований и базовых компонентов. http://is.ifmo.ru/genalg/_2007_01_report-genetic.pdf
52. *Хопкрофт Дж., Мотвани Р., Ульман Дж.* Введение в теорию автоматов, языков и вычислений. М.: Вильямс, 2002.
53. *Царев Ф. Н., Шалыто А. А.* Применение генетического программирования для генерации автомата в задаче об «Умном муравье» / Сборник трудов IV-ой Международной научно-практической конференции «Интегрированные модели и мягкие вычисления в искусственном интеллекте». Том 2. М.: Физматлит. 2007, с. 590–597. http://is.ifmo.ru/genalg/_ant_ga.pdf
54. *Шалыто А. А*. SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
55. *Шеннон К.* Работы по теории информации и кибернетике. М.: Изд-во «иностр. литер.», 1963.