МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
(МИНОБРНАУКИ)

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

УДК 004.4’242
№ госрегистрации
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 I ЭТАПУ «ВЫБОР НАПРАВЛЕНИЯ ИССЛЕДОВАНИЙ И БАЗОВЫХ КОМПОНЕНТОВ»

ЛИСТОВ 115

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

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	Глава 2.
Ведущий научный сотрудник, канд. техн. наук	В. В. Киселев	Глава 1.
Ведущий научный сотрудник, канд. техн. наук	Р. Н. Котляр	Раздел 2.1.
Ведущий научный сотрудник, канд. техн. наук	Ю. П. Московцев	Раздел 2.3.
Ведущий научный сотрудник, канд. техн. наук, доцент	В. А. Третьяков	Раздел 2.2.
Ведущий научный сотрудник, канд. техн. наук	Г. М. Файкин	Раздел 1.1.
Руководитель временного трудового коллектива (ВТК), ассистент кафедры КТ	А. П. Мельничук	Раздел 2.2.
Член ВТК, профессор кафедры ИС	Е. Ю. Михайлова	Раздел 2.2.2.
Член ВТК, доцент кафедры КТ	В. Д. Наумчик	Раздел 2.2.1.
Член ВТК, доцент кафедры КТ	М. Ю. Осипов	Раздел 2.2.1.
Член ВТК, доцент кафедры КТ	А. Н. Воробьев	Раздел 2.2.2.
Член ВТК, доцент кафедры КТ	Ю. А. Щупак	Раздел 2.2.2.
Член ВТК, доцент кафедры КТ	С. В. Чириков	Раздел 2.2.1.
Член ВТК, доцент кафедры КТ	А. С. Сегаль	Раздел 2.2.2.
Член ВТК, доцент кафедры КТ	Д. Г. Шопырин	Раздел 2.2.1.
Член ВТК, доцент кафедры ИС	Д. А. Зубок	Раздел 2.2.2.
Член ВТК, ассистент кафедры ИС	В. В. Повышев	Раздел 2.2.1.
Член ВТК, ассистент кафедры ИС	В. В. Ильин	Раздел 2.2.1.
Член ВТК, ассистент кафедры ИС	М. Г. Холин	Раздел 2.2.2.
Аспирант	П.Г. Лобанов	Раздел 2.2.3.
Магистрант	Н.И. Поликарпова	Разделы 1.1.1, 1.2.
Магистрант	В. Н. Точилин	Разделы 1.1.1, 1.2.
Магистрант	Ф.Н. Царев	Раздел 2.2.
Магистрант	Ю.Д. Бедный	Глава 1.
Магистрант	В.Р. Данилов	Глава 1.
Студент	Е.А. Мандриков	Раздел 2.1.
Студент	В.А. Кулев	Раздел 2.1.

РЕФЕРАТ

Излагаются результаты анализа методов генерации автоматов управления системами со сложным поведением и базовых компонентов генераторов таких автоматов с целью выбора направления исследований при проведении 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).

Излагаются результаты анализа существующих методов методы генерации автоматов управления системами со сложным поведением. Предложен набор требований, которым должны удовлетворять эти методы. Описываются базовые компоненты генетического программирования, предназначенные для генерации автоматов управления системами со сложным поведением. Проанализированы методы описания структур хромосом и построения оценочных функций. Даны рекомендации по их выбору для различных типов задач.

ОГЛАВЛЕНИЕ

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 является выбор направления исследований и разработка базовых компонентов генераторов автоматов управления системами со сложным поведением. Задачами этапа являются:

1. Проведение анализа существующих методов генерации автоматов.
2. Формирование требований к методам генерации автоматов.
3. Разработка базовых компонент генетического программирования.

В настоящее время работы по построению автоматов на основе генетического программирования ведутся 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 более простые автоматы, чем те, которые требуются 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 может быть решена, если учесть специфику автоматов, применяемых в системах управления, состоящую 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. Отметим, что при этом могут рассматриваться хромосомы не только для автомата в целом, но и для отдельных его состояний.

Специфика хромосом позволяет находить эффективные виды операций кроссовера и мутации, что также сокращает время поиска решения, приемлемого по выбранному критерию. Отметим, что эффективное представление хромосом и указанных операций применимо для широкого класса задач, тогда как эффективную функцию приспособленности требуется строить для каждой задачи в отдельности.

Патентные исследования, проведенные в рамках первого этапа работы (отчет о патентных исследованиях № 2007.08.31-01 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 превышать мировой уровень разработок в рассматриваемой области.

1. АНАЛИЗ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ГЕНЕРАЦИИ АВТОМАТОВ УПРАВЛЕНИЯ СИСТЕМАМИ СО СЛОЖНЫМ ПОВЕДЕНИЕМ

В данной главе рассмотрено понятие «система со сложным поведением». Обосновывается применение автоматного подхода для описания систем этого типа. Показывается, что генетические алгоритмы в ряде задач успешно применяются для автоматизированного построения автоматов. При анализе этих задач особое внимание уделяется методам генерации автоматов: способу представления автомата в виде особи генетического алгоритма и используемым генетическим операторам. При этом отмечается, что высокоуровневое представление автоматов оказывается более эффективным, нежели низкоуровневое описание. Сформированы требования к генерации автоматов управления системами со сложным поведением.

1.1. Анализ предметной области

1.1.1. Системы со сложным поведением

В процессе создания программного обеспечения часто возникает необходимость реализации систем со *сложным поведением*. Таким поведением обладают многие устройства управления, сетевые протоколы и т.д.

Будем считать, что система обладает «нетривиальным» поведением, если в ответ на возникновение некоторого *события* она, в зависимости от предыстории, может совершить одно из нескольких *действий.*

При традиционной программной реализации систем с таким поведением программистами используются переменные, называемые *флагами*, которым не соответствуют никакие элементы предметной области. Предназначение флагов – участвовать в многочисленных конструкциях ветвления, реализующих логику поведения. Флаги неявно задают отдельные компоненты состояний. Использование флагов трудно для понимания, подвержено ошибкам и практически не расширяемо.

Вместо этого в последнее время предлагается описывать системы со сложным поведением, приписывая каждой из них некоторое множество *управляющих состояний*. В этих состояниях поведение системы является простым и может быть описано явно. Связь управляющих состояний с действиями и механизм переходов между состояниями удобно описывать с помощью *конечных автоматов* с выходами [9]. При этом все поведение системы оказывается сосредоточенной в автомате или системе взаимодействующих автоматов. Такой подход к описанию поведения называют *автоматным* [5].

Будем считать, что система обладает сложным поведением, если она описывается автоматом, или системой взаимодействующих автоматов с достаточно большим числом состояний и переходов.

Одна из центральных идей автоматного программирования (http://is.ifmo.ru/works/_politeh.pdf) состоит в отделении описания *логики* поведения (при каких условиях необходимо выполнить те или иные действия) от описания его *семантики* (собственно смысла каждого из действий). Кроме того, описание логики при автоматном подходе жестко структурировано. Эти два свойства делают автоматное описание сложного поведения ясным и удобным.

Одним из наиболее широких классов систем, обладающих сложным поведением, являются реактивные системы. Системы этого класса могут быть также названы событийными. В таких системах в качестве входных воздействий используются события и входные переменные. События, в отличие от входных переменных, не опрашиваются программой, а вызывают соответствующие им обработчики. Входные переменные и выходные воздействия реализуются произвольными подпрограммами (функциями). Перечислим основные отличия реактивных систем от систем других классов.

Если в системах логического управления [10] в качестве входных воздействий используются опрашиваемые программой двоичные входные переменные и предикаты, соответствующие определенным состояниям автоматов, взаимодействующих с рассматриваемым автоматом, то для «реактивных» систем это понятие расширено. Во-первых, в качестве входных переменных применяются любые подпрограммы (функции), возвращающие двоичные значения, а, во-вторых, введены события, не только обеспечивающие возможность выполнения переходов в автоматах, но и инициирующие запуск автоматов. События могут также инициировать реализацию выходных воздействий в случае, когда состояние автомата не изменяется.

Другое отличие «реактивных» систем от систем логического управления состоит в том, что в них в качестве выходных воздействий применяются не двоичные переменные, а произвольные подпрограммы.

Также как и в системах логического управления, в «реактивных» системах алгоритмы представляются в виде системы взаимосвязанных автоматов. При этом если в системах первого типа взаимодействие автоматов в основном осуществляется за счет обмена номерами состояний, а вложенность присутствует в «зачаточном» состоянии, то в «реактивных» системах число способов взаимодействия увеличилось.

Кроме того, если в системах логического управления наиболее целесообразно применять такую структурную модель как автомат Мура, то в «реактивных» (событийных) системах часто более рационально использовать другую модель – смешанный автомат, совмещающий в себе свойства автоматов Мура и Мили.

В качестве примера системы со сложным поведением, управляемой автоматами, приведем систему управления дизель-генератором [8]. На рис. 1 изображена схема взаимодействия автоматов, которые образуют эту систему.

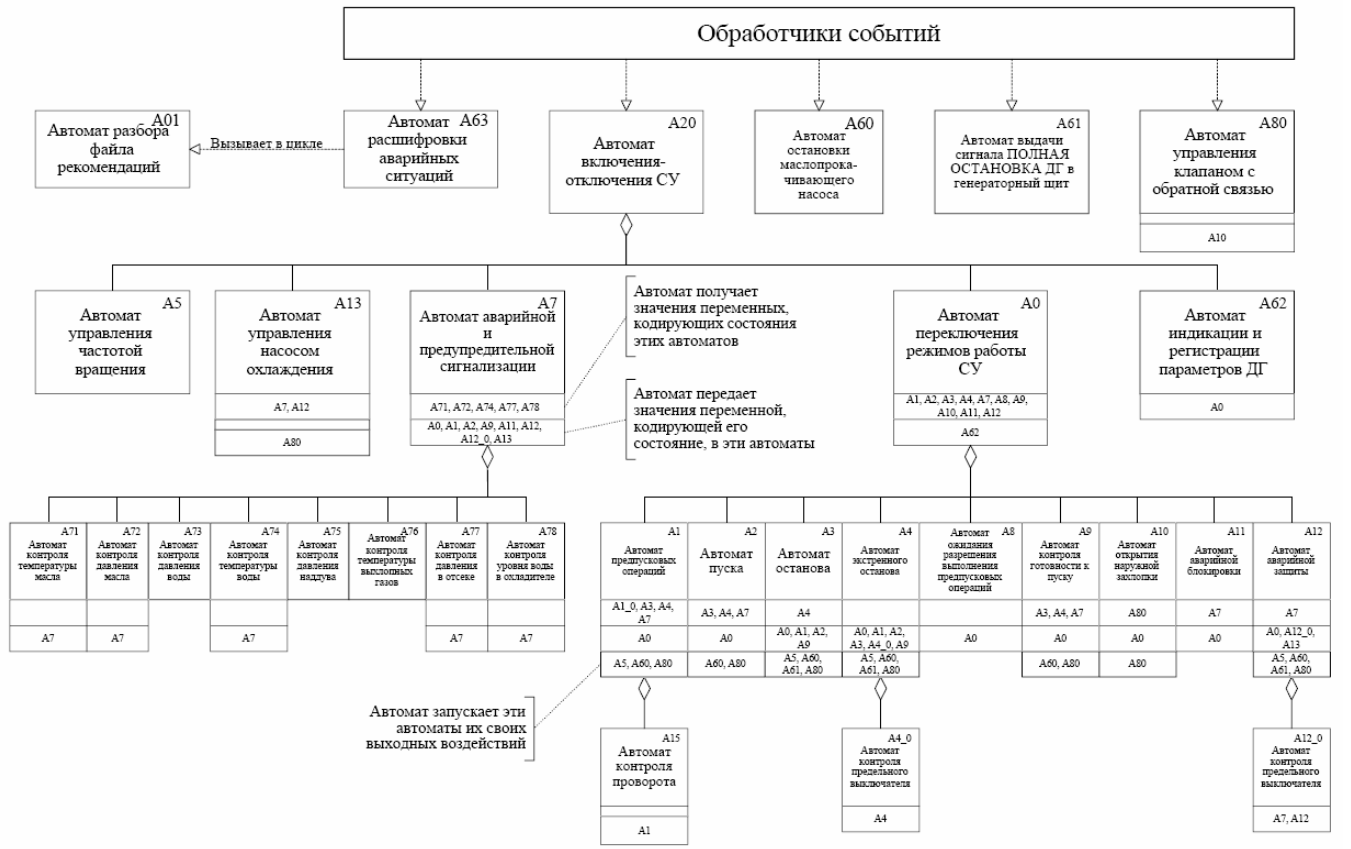


Рис. 1. Схема взаимодействия автоматов, управляющих системой со сложным поведением

В случае, когда автоматы описывают системы со сложным поведением, их проектирование является нетривиальной, трудоемкой, а иногда и неразрешимой задачей. Поэтому возникает естественное желание – *автоматизировать процесс построения автоматов*, поручив основную работу компьютеру. При этом нет необходимости, как при традиционном подходе, заранее учитывать все особенности решаемой задачи и действия, которые должна предпринимать программа. Одним из возможных методов проектирования автоматов, соответствующих этим требованием, являются *генетические алгоритмы.*

1.1.2. Генетические алгоритмы

Для природы характерна оптимальность различных биологических объектов, а также согласованностью и эффективностью их работы. Многих исследователей давно интересовал «философский» вопрос – возможно ли эффективное построение значимых и полезных для человека систем на основе принципов биологической эволюции?

Подобные идеи возникали у ряда исследователей. В 1962 г. Л. Фогель занялся изучением интеллектуального поведения индивида и его развития в процессе эволюции [23]. При этом поведение индивида задавалось конечным автоматом. Продолжая данные исследования, Л. Фогель, А. Оуэнс и М. Уолш предложили в 1966 г. схему эволюции конечных автоматов, решающих задачи предсказания [24]. Примерно в это же время (в середине 60-х годов) Дж. Холланд разработал новый метод поиска оптимальных решений – генетические алгоритмы. Результатом работы Дж. Холланда стала книга [30], вышедшая в 1975 г. Эти работы заложили основы эволюционных вычислений.

Приведем ряд определений, которые играют важную роль в теории эволюционных вычислений и необходимы для описания генетических алгоритмов.

Популяция – это совокупность особей, способная к устойчивому самовоспроизводству, относительно обособленная от других групп, с представителями которых потенциально возможен генетический обмен.

Индивид (особь) – единичный представитель популяции.

Фенотип – совокупность характеристик, присущих индивиду на определённой стадии развития. Фенотип формируется на основе генотипа, опосредованного рядом внешнесредовых факторов.

Генотип – наследственная информация, закодированная в хромосомах, которая вместе с факторами внешней среды определяет фенотип организма. Генотип не всегда соответствует одному и тому же фенотипу. Некоторые гены проявляются в фенотипе только в определенных условиях.

Хромосома – структура, содержащая генетический код индивида.

Ген – определенная часть хромосомы, кодирующая врожденное качество индивида.

Генетические алгоритмы – это оптимизационный метод, базирующийся на принципах естественной эволюции популяции особей (индивидов). Каждая особь характеризуется приспособленностью – функцией ее генов. Задача оптимизации состоит в максимизации функции приспособленности. В процессе эволюции – в результате отбора, рекомбинаций, мутаций геномов особей, а также применения ряда других генетических операторов, происходит поиск особей с высокой приспособленностью. По сравнению с обычными оптимизационными методами генетические алгоритмы имеют особенности: параллельный поиск, случайные мутации и рекомбинации уже найденных хороших решений. Приведем описание генетических операторов.

Как известно в теории эволюции, важную роль играет то, каким образом признаки родителей передаются потомкам. В генетических алгоритмах за передачу признаков родителей потомкам отвечает оператор, который называется *рекомбинация*. В литературе по генетическим алгоритмам этот оператор называют также кроссинговер, кроссовер, скрещивание. В данном обзоре будет использоваться термин рекомбинация. Итак, рекомбинация – процесс обмена генетическим материалом. Это операция, при которой две хромосомы обмениваются своими частями.

Мутация – случайное изменение одной или нескольких позиций в хромосоме. Мутации, которые проявляются на уровне фенотипа, могут иметь как пагубные последствия, так и положительные – приводить к появлению у индивида новых полезных признаков. Таким образом, мутации являются двигателем естественного отбора, так как являются механизмом поддержания разнообразия особей в популяции.

Из-за избыточности и неоднозначности кодирования одному и тому же фенотипу может соответствовать множество генотипов. При этом часто порядок генов в хромосоме является критическим для строительных блоков, позволяющих осуществлять эффективную работу алгоритма. В ряде работ были предложены методы для *переупорядочения* позиций генов в хромосоме во время работы алгоритма. Одним из таких методов является инверсия, выполняющая реверсирование порядка генов между двумя случайно выбранными позициями в хромосоме. Когда используются методы переупорядочения, гены должны содержать некоторую «метку», такую, чтобы их можно было правильно идентифицировать независимо от их позиции в хромосоме. Цель переупорядочения состоит в том, чтобы попытаться найти порядок генов, который имеет лучший эволюционный потенциал. Многие исследователи использовали инверсию в своих работах, однако мало кто пытался ее обосновать или определить ее вклад. Переупорядочение также значительно расширяет область поиска. Мало того, что генетический алгоритм пытается находить «хорошие» множества генов, но он также одновременно пробует находить хорошее упорядочение генов. Это гораздо более трудная

задача для решения. Еще раз отметим, что оператор переупорядочения не изменяет фенотипа индивидуума, а изменяет лишь структуру хромосомы.

Как правило, выделяют еще один оператор для генетических алгоритмов – *оператор генерации случайной особи*, который может быть использован при создании начальной популяции, при пополнении популяции случайными особями и при некоторых мутациях.

В настоящем разделе термины «генетические алгоритмы» и «генетическое программирование» понимаются в широком смысле – для обозначения эволюционных вычислений. В случае, когда речь идет о методе генетического программирования, предложенным *J.R.Koza* [37], употребляется термин «традиционное генетическое программирование». Для обозначения генетических алгоритмов, оперирующих с битовыми строками фиксированной длины (описанных, например, в работе [43]), используется термин «традиционные генетические алгоритмы».

1.1.3. Преимущества автоматизированного построения автоматов

В данном разделе будет рассмотрен ряд задач, в которых применение эволюционных алгоритмов для автоматизированного построения автоматов позволило улучшить существующие решения или получить решения, сравнимые по эффективности с существующими. Кроме того, следует принять во внимание, что трудозатраты человека при подходе, базирующемся на эволюционных алгоритмах, как правило, значительно меньше, чем при эвристическом построении автоматов. Важно отметить и то обстоятельство, что в некоторых случаях автоматизировано построенные автоматы могут быть изучены для выявления структуры «хороших» решений задачи.

При использовании эволюционных методов не требуется точно указывать, как решать задачу. Вместо этого достаточно ее сформулировать в терминах некоторой «цели» и «допустимых затрат», программа будет создана самой вычислительной системой в ходе эволюционного процесса. **Высказывание «машина никогда не знает больше программиста» при этом оказывается просто неверным.**

1.1.3.1. Итерированная дилемма узника (теория игр)

В теории игр известна следующая бескоалиционная матричная игра с ненулевой суммой, обычно называемая «Дилемма узника» («Prisoner’s dilemma»). Двое преступников пойманы и допрашиваются в отдельных камерах. Срок тюремного заключения, который получит каждый из них, зависит как от его показаний, так и от показаний соучастника. В табл. 1 приведены сроки заключения в зависимости от решений, принятых игроками – их стратегий.

Таблица 1. Матрица игры «Дилемма узника»

	Сознаться	Отрицать
Сознаться	3, 3	0, 5
Отрицать	5, 0	1, 1

В данной игре предательство (стратегия «Сознаться») строго доминирует над сотрудничеством (стратегией «Отрицать»). Единственное равновесие в игре (по Нэшу) – признание обоих преступников, что приведет ситуации (3, 3). При любой зафиксированной стратегии соучастника преступнику выгодно предать. Таким образом, действую по отдельности рационально, игроки приходят к нерациональному решению (3, 3), хотя могли бы получить «всего» по году заключения (1, 1), выбрав стратегию «Отрицать». Равновесие по Нэшу в данной игре не является Парето-оптимальным.

В 1980 году Роберт Аксельрод рассмотрел вариант данной игры, названный им «Iterated Prisoner’s Dilemma», в котором между игроками проводится несколько партий и каждый игрок

помнит историю трех предыдущих игр. Р. Аксельрод организовал турниры, на которые программу мог прислать любой желающий. В первом турнире участвовало 14 программ, во втором – 63. Результат программы в турнире равнялся количеству очков, набранных против остальных участников. Проведенные турниры показали [14], что «жадные» стратегии (предпочитающие сознаться, что являлось оптимальной стратегией в исходной игре «Дилемма узника») имеют низкую результативность. Победителем обоих турниров стала простая программа, названная *Tit for Tat*, которая состит всего из четырех строк на языке *BASIC*, присланная Анатолием Рапопортом. Эта программа первым ходом выбирает стратегию «Отрицать», а дальше делает тоже, что и оппонент на предыдущем ходу.

В 1987 году Р. Аксельрод изучил возможность эволюционного порождения стратегий с помощью генетических алгоритмов. Стратегия программы кодировалась битовой строкой из 70 символов. В результате была получена стратегия, которая дает чуть лучший результат, чем программа *Tit for Tat* при фиксированных соперниках. В дальнейшем Р. Аксельрод провел исследование, в котором среда (оппоненты) также эволюционировала. Было замечено, что в ходе первых двух десятков поколений преобладали «жадные» стратегии (сознаться), однако в ходе дальнейшей эволюции они уступали место «кооперирующимся» (подобным программе *Tit for Tat*). Наилучшие стратегии, полученные в ходе данного исследования, также являлись различными модификациями стратегии, использованной в программе *Tit for Tat*.

В 1991 году Д. Фогель провел ряд экспериментов, в которых эволюционное программирование использовалось для генерации *автоматов*, кодирующих стратегии игроков [22]. На рис. 2 приведен пример автомата, кодирующего стратегию игрока. На этом рисунке действию стратегии «Отрицать» соответствует символ *C*, а стратегии «Сознаться» – символ *D*. Стартовое состояние (состояние 1) выделено. Каждый переход помечен дугой вида (*P1*, *P2*)/*A*. Здесь *P1* – действие (стратегия) игрока на предыдущем ходу (C либо D), *P2* – действие оппонента на предыдущем ходу, *A* – действие игрока на текущем ходу. Стратегия игрока на первом ходу обозначена пунктирной линией.

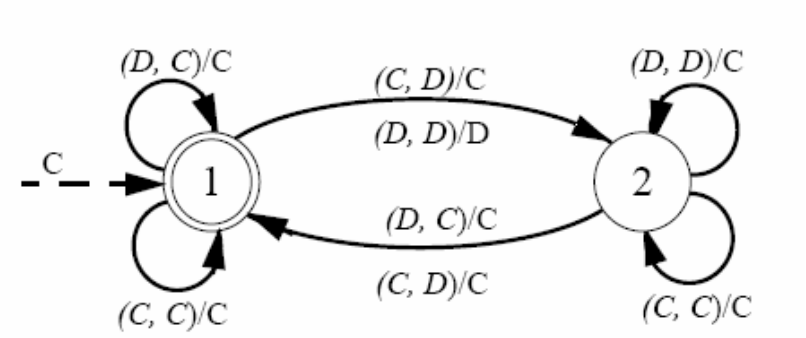


Рис. 2. Автомат, кодирующий стратегию игрока

Таким образом, особь эволюционного алгоритма являлась автоматом. Число состояний автомата находилось в диапазоне от одного до восьми. На каждом этапе эволюционного алгоритма (поколении) проводился турнир между особями популяции, на основе которого вычислялась функция приспособленности. Затем каждый автомат из лучшей половины популяции порождал одного потомка путем одной из шести мутаций. Возможными мутациями являлись: добавление состояния, удаление состояния, изменение перехода, изменение действия на переходе, изменение начального состояния и изменение действия игрока на первом ходу.

Результаты экспериментов Д. Фогеля в целом совпадают с результатами Р. Аксельрода, с той лишь разницей, что процесс эволюции шел немного быстрее – «кооперирующиеся» стратегии занимали место «жадных» после первых 5 – 10 поколений.

1.1.3.2. Распознавание изображений

Конечные автоматы успешно применялись в [17] для распознавания изображений. Задача была поставлена следующим образом: необходимо выделить корабли на изображении, полученном с радара. Пример изображения приведен на рис. 3 (белыми квадратами выделены корабли).

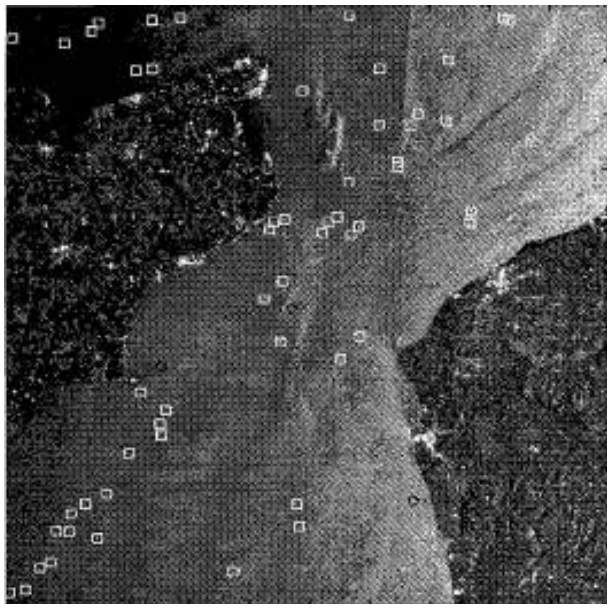


Рис. 3. Пример изображения с радара

Опишем, что подается на вход автомату. Первоначально каждому пикселю ставится в соответствие квадрат девять на девять пикселей с центром в ассоциированном пикселе. Затем автомату последовательно подается некая статистика по квадратам: средняя яркость по квадрату, средняя яркость по периметру, дисперсия яркости и яркость центра.

Заметим, что наивное хранение переходов для всех возможных наборов значений параметров в этой задаче неприменимо. Для того, чтобы сузить входной алфавит автомата применяется следующий подход: вычисляется некая функция-переходник от входных параметров, результат которой дискретизируется и подается на вход автомату, как входное воздействие. Функции, используемые в разных состояниях, различны.

Функция-переходник представляется с помощью дерева разбора. При этом используются следующие нетерминалы арности два: `ADD` (сложение), `SUB` (вычитание), `MUL` (умножение), `DIV` (деление), `MIN` (взятие минимума), `MAX` (взятие максимума). Терминалами являются входные параметры распознавателя.

Входной алфавит автомата сужается до двух входных воздействий – 0 и 1. Если значение функции-переходника больше определенного порога, то автомату на вход подается единица, иначе – ноль. Разным состояниям соответствуют различные пороговые значения.

Для построения автомата был применен эволюционный алгоритм. Мутации совершаются как над автоматом, так и над функцией-переходником и пороговыми значениями. Интересно, что вероятности мутаций коэволюционируют вместе с автоматом. Введены следующие мутации:

1. Добавление состояния.
2. Удаление состояния.
3. Изменение стартового состояния.
4. Мутация перехода.
5. Мутация порогового значения в состоянии.

6. Перестановка двух функций-переходников в различных состояниях.
7. Мутация функции-переходника в состоянии.
8. Рекомбинация двух состояний.
9. Рекомбинация двух функций-переходников.

Проведенное исследование показывает, что изложенный подход приводит к лучшим результатам, чем традиционное генетическое программирование и искусственные нейронные сети: при том же результате на тестовой коллекции выведенные автоматы лучше работают на реальных данных. Кроме того, построенный алгоритм проще воспринимается человеком, так как логика системы разбивается на отдельные слои – общую схему-автомат и конкретные условия переходов, найденные генетическим программированием.

1.1.3.3. Задача выбора праймера для полимеразной цепной реакции (молекулярная биология)

В работе [31] авторы предлагают новой подход к решению известной в молекулярной биологии задачи выбора праймера для полимеразной цепной реакции (ПЦР) [7]. Естественным способом проверки пригодности праймера является попытка проведения ПЦР, однако в случае неудачи такая реакция приведет к существенным как материальным, так и временным затратам. Поэтому желательно иметь способ определения пригодности праймера без проведения ПЦР. Существует ряд критериев пригодности праймера для ПЦР, перечисленных в работе [7], благодаря которым могут быть построены различные эвристические методы проверки праймера.

В рассматриваемой работе [31] предлагается принципиально другой подход, в котором эволюционный алгоритм используется для построения классификатора праймеров, являющгося *конечным автоматом*, который позволяет в ряде случаев предсказывать, подходит ли данный праймер для проведения ПЦР. Конечный автомат строится на основе множества заранее известных тренировочных данных – праймеров, для которых известно подходят ли они для проведения ПЦР или не подходят.

Особью генетического алгоритма является конечный автомат с выделенным начальным состоянием. В экспериментах, описываемых авторами, автоматы имеют 64 состояниями. Множество входных воздействий – множество всех подмножеств множества нуклеотидов {A, T, G, C}. Состояния автомата бывают трех типов: хорошие, плохие и промежуточные. На рис. 4 приведен пример автомата описанного вида, с той лишь разницей, что в нем для простоты изображено только пять состояний. Приведенные типы состояний обозначены символами «+»,«–» и «?» для хороших, плохих и промежуточных типов состояний соответственно.

Для предсказания пригодности праймера необходимо подать его на вход автомату, считая суммарное количество хороших и плохих состояний, в которые переходит автомат по мере увеличения принимаемого префикса. Если количество плохих состояний превосходит количество хороших, то считается, что праймер не подходит. Если количество плохих состояний меньше количества хороших, то считается, что праймер подходит для ПЦР. В случае же равенства – ответ не дается.

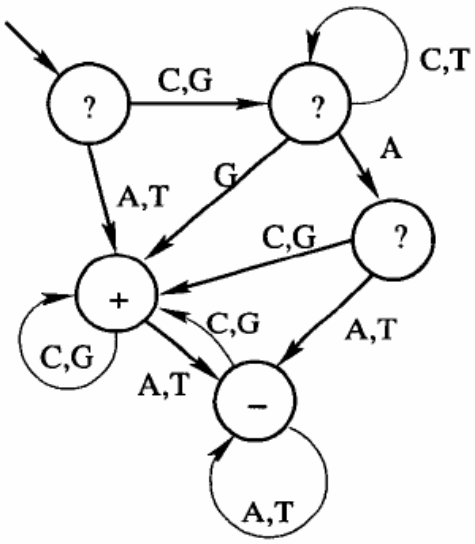


Рис. 4. Структура автомата, классифицирующего праймеры

Для задания функции приспособленности f автомата A используется тренировочное множество T из 695 праймеров, 440 из которых подходят для проведения ПЦР, а 255 – не подходят. Пусть $p \in T$ – некоторый праймер, p_i – префикс данного праймера длины i : $len(p_i) = i$, $s(p_i)$ – состояние, в которое переходит автомат A, принимая на вход префикс p_i, $g(p_i) := g'(type(s(p_i)), good(t))$ – функция, возвращающая некоторое целое число в зависимости от $type(s(p_i))$ – типа состояния, в которое переходит автомат по префиксу p_i, и $good(t)$ – является ли праймер подходящим для ПЦР. Функция g', используемая в эксперименте, приведена в табл. 2.

Таблица 2. Функция g'

Праймер	Тип состояния		
	+	–	?
Подходящий	3	-4	0
Не подходящий	-5	7	0

Данная функция означает прирост приспособленности при увеличении длины префикса и соответственно при переходе автомата в следующее состояние. При этом функция приспособленности автомата $f(A) = \sum_{p \in T} \sum_{i=1}^{len(p)} g(p_i)$.

В экспериментах использовалась популяция из 600 автоматов. Каждый автомат как особь популяции кодировался битовой строкой. Эволюционный алгоритм содержал 100 поколений. Начальное поколение автоматов создавалось случайным образом. После вычисления функции приспособленности на элементах популяции, она также случайным образом разбивалась на группы по четыре особи (автомата). Две наименее приспособленные особи в каждой четверке заменялись особями, полученными в результате рекомбинации двух наиболее приспособленных особей. Применялась двухточечная рекомбинация строк. После рекомбинации осуществлялась мутация, которая в каждом десятом случае изменяла начальное состояние, в 30% случаев – переход по одному из входных воздействий и в 60% случаев – тип состояния.

Было проведено две серии экспериментов. В первой из них эволюционный алгоритм был выполнен 100 раз. При этом перед каждым запуском начальная популяция инициализировалась случайными автоматами, а после запуска сохранялась наилучшая из полученных особей. Затем, из 100 наилучших особей была образована популяция P. Вторая серия состояла также из 100 экспериментов, однако, в каждом из них в качестве начальной популяции автоматов использовалась популяция P. Данный подход называется гибридизация.

Для тестирования наилучшего из автоматов, полученных в первой и во второй серии экспериментов, было выбрано тестовое множество, состоящее из 880 подходящих и 471 не подходящих для ПЦР праймеров. Наилучший из автоматов, полученных в результате первой серии экспериментов, показал результаты, приведенные в табл. 3.

Таблица 3. Результаты классификации для наилучшего автомата первой серии экспериментов

Праймер	Результат классификация		
	+	–	?
Подходящий	727	153	39
Не подходящий	208	263	

Данное качество предсказаний считается весьма хорошим. При этом наилучший из автоматов, полученных во второй серии экспериментов, показал еще более хорошее качество классификации.

1.1.3.4. Распознавание регулярных языков

Из теории языков и вычислений известно, что конечные детерминированные автоматы способны распознавать регулярные языки. В связи с этим актуальна задача построения автомата, распознающего по множеству примеров некий язык. Для решения этой задачи успешно применялись различные генетические алгоритмы.

Задача может быть усилена до построения автомата с минимальным количеством состояний. Однако, в работе [27] показано, что модифицированная задача NP-трудна.

В работе [16] автоматы представлялись с помощью таблицы переходов. Разработанный в этой работе метод позволяет поддерживать в популяции автоматы с разным числом состояний. Функция приспособленности учитывает три компонента – количество верно распознанных тестовых примеров, количество состояний и переходов автомата, степень общности языка, соответствующего построенному автомату. Это позволяет сузить область поиска, и находить языки, соответствующие определенным критериям. В тестовую коллекцию могут входить примеры слов, которые как принадлежат, так и не принадлежат языку.

Приведем описание генетической операции репродукции. Число состояний потомка выбирается случайно из диапазона $[N - 2, N + 2]$, где N — число состояний первого родителя. После этого каждый переход копируется из родителей, причем вероятность копирования перехода из заданной особи прямо пропорциональна значению ее функции приспособленности. Переходы, представленные только в одном из родителей, копируются из того родителя, в котором присутствуют. Переходы, отсутствующие в обоих родителях, генерируются случайно.

Генетическая операция мутации осуществляется изменением случайного перехода. При этом переход разрешается удалять. Это приводит к распаду графа переходов на несколько компонент. Результаты экспериментов показали, что удаление недостижимых состояний замедляет вывод. Поэтому оно не производится.

Предложенный подход был проверен на практических задачах. Авторам удалось построить автомат из семи состояний, который распознает по множеству из ста примеров русские двухсложные слова.

В работе [18] для генерации автомата был применен следующий вариант генетического программирования: выводилось выражение, результатом вычисления которого являлся автомат. Выражение описывало последовательность действий, преобразующих исходный автомат (рис. 5) в результирующий. Для вывода выражений, кодирующих развитие автомата, было применено традиционное генетическое программирование, оперирующее над деревьями выражений.

Рис. 5. Исходный автомат

Опишем процесс преобразования автомата. Построение автомата является неким аналогом онтогенеза – перехода клетки в организм путем деления и модификации. В каждый момент некая дуга автомата является активной. Операции, используемые в выражении, могут делить активную дугу или модифицировать ее, выбрав при этом другую дугу в качестве активной. Авторами этой работы вводятся следующие нетерминалы:

- `PARALLEL` – создать новое допускающее состояние автомата и направить туда активную дугу. При этом созданное состояние копируется из состояния, в которое активная дуга вела до операции;
- `PARALLEL-REJ` – создать новое отвергающее состояние и направить туда активную дугу аналогично `PARALLEL`;
- `RETRACT` – зациклить активную дугу: направить ее в состояние, из которого она исходит;
- `WRITE-NODE` – положить в стек состояние, в которое ведет активная дуга;
- `TO-NODE` – снять состояние со стека и направить в него активную дугу. Если стек пуст, то активная дуга не модифицируется.

В работе [18] введен единственный терминал DONE, означающий конец редактирования. Пример выражения, использующего эти функции, приведен на рис. 6. Для наглядности нетерминалы пронумерованы в порядке исполнения. Для нетерминалов `PARALLEL` и `PARALLEL-REJ` первый аргумент соответствует действиям, совершаемым над созданным состоянием. При этом активная дуга переходит в первую дугу, исходящую из нового состояния. Второй аргумент этих нетерминалов соответствует действиям, совершаемым над исходным состоянием. При этом активная дуга заменяется на следующую дугу, исходящую из состояния. Для остальных нетерминалов единственный аргумент вычисляется аналогично второму аргументу `PARALLEL` и `PARALLEL-REJ`.

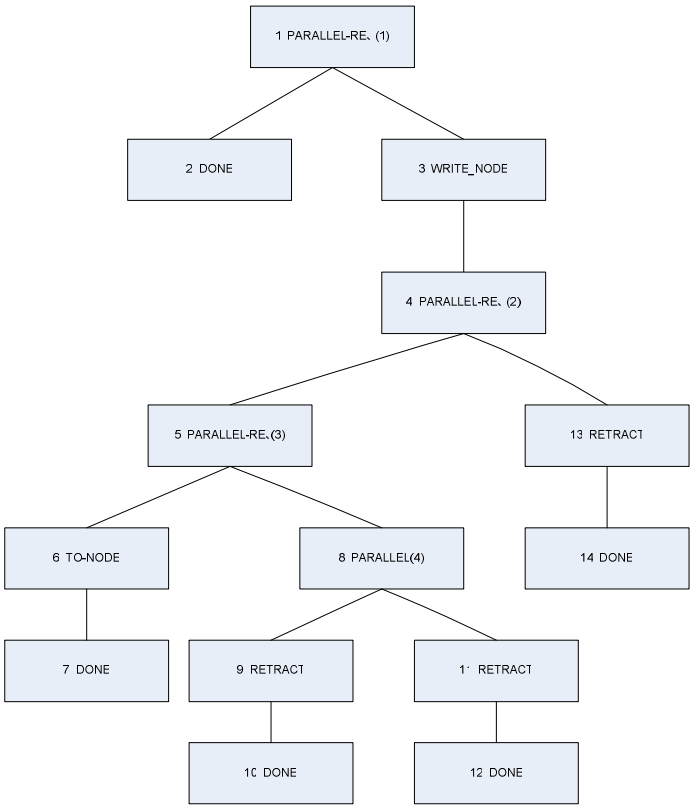


Рис. 6. Выражение, описывающее развитие автомата

На рис. 7 показан процесс развития автомата из исходного по приведенному выражению.

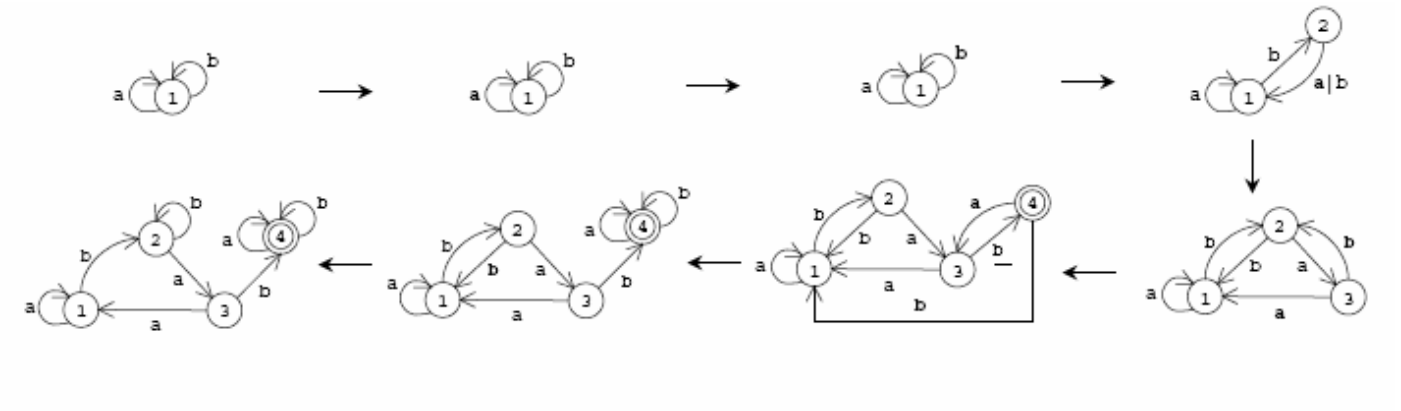


Рис. 7. Процесс построения автомата при вычислении выражения

Известно, что регулярные языки могут быть представлены как объединение, пересечение или дополнение других регулярных языков. Следовательно, можно произвести декомпозицию языка на более простые языки, для которых проще найти распознающие автоматы. В работе [18] определено решение в виде комбинации трех автоматов через нетерминалы `AND`, `OR` и `NOT`. Разложение также определяется с помощью генетического программирования. Пример выведенного с помощью этой методики распознавателя изображен на рис. 8.

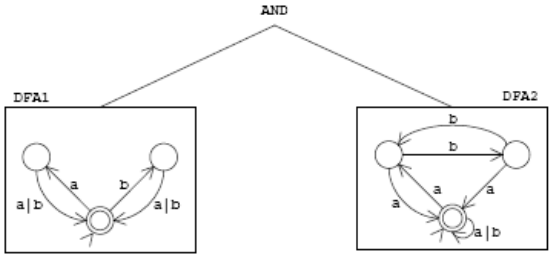


Рис. 8. Декомпозиция распознавателя

Описанный подход был протестирован на наборе из девяти регулярных языков. К сожалению, для одного из тестовых языков так и не удалось построить автоматический распознаватель.

В работе [40] произведено сравнение двух подходов к автоматическому построению автоматов: использование генетических алгоритмов и применение эвристики *Evidence Driven State Merging (EDSM)*. После анализа проведенных экспериментов был сделан вывод, что автомат, построенный с помощью генетических алгоритмов, лучше соответствует искомому языку. С другой стороны, эвристика *EDSM* всегда строит автомат, который верно распознает множество примеров.

1.1.3.5. Искусственная этология

Этология – это дисциплина зоологии, занимающаяся изучением поведения особи в естественной для неё среде. Искусственная этология – направление этологии, в котором особи и среда, в которой они обитают, являются объектом компьютерного моделирования. Преимущество данного подхода состоит в том, что в отличие от реального мира (среды обитания), обладающего огромным количеством факторов в разной степени влияющих на поведение особи, искусственный (виртуальный) мир, может быть создан достаточно простым для изучения. В то же время основные результаты, полученные в рамках упрощающих предположений, могут быть перенесены на реальный мир.

В работе [28] с помощью искусственной этологии изучаются процессы коммуникации между особями в ходе эволюции популяции данных особей. В рассматриваемой работе особи названы симоргами. Авторы работы определили требования к искусственной среде, необходимые для того, чтобы коммуникация имела смысл и была полезна симоргам. Эти требования таковы:

- каждый симорг должен обозревать часть среды обитания, которая недоступна обозрению других симоргов;
- среда обитания должна иметь возможность распространять сигналы, посылаемые одним симоргом другому.

В среде обитания, построенной исследователями, каждый симорг находится в своей *локальной среде*, доступной для обозрения только ему. Состояние локальной среды задается элементом из подмножества натуральных чисел $S = [1..8]$ и определяется случайным процессом – не может быть предсказано. Таким образом, единственным способом узнать состояние локальной среды другого симорга является коммуникация. Сообщение представляется элементом *S*. Для передачи сообщений между симоргами вводится *общая среда*, в которую симорг может передавать сообщения и из которой он может принимать сообщения других симоргов. Состояние общей среды (также элемент *S*) определяется только одним, последним, сообщением, из помещаемых в нее симоргами. Топология среды обитания симоргов приведена на рис. 9.

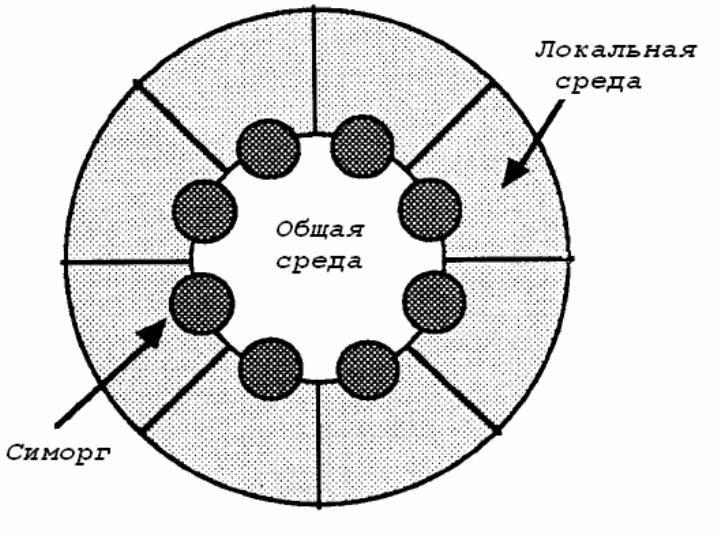


Рис. 9. Топология среды обитания симоргов

Симорг задается простейшим *конечным автоматом*, содержащим всего одно состояние. Следовательно, у симоргов нет памяти. Множеством входных воздействий автомата является декартово произведение множества состояний общей среды и множества состояний локальной среды (S^2). Определены два вида выходных воздействий автомата: *сообщение* (элемент S) и *действие* (также элемент S). Выходное воздействие первого вида изменяет состояние общей среды, а второго вида – обеспечивает «выживаемость» симорга. Оно должно быть как можно более «эффективно». Если действие симорга совпадает с состоянием общей среды (которое равно состоянию локальной среды последнего из симоргов отправивших сообщение), то «приз» (одно очко) получает как симорг, который последним отправил сообщение, так и симорг, действие которого совпало с этим сообщением.

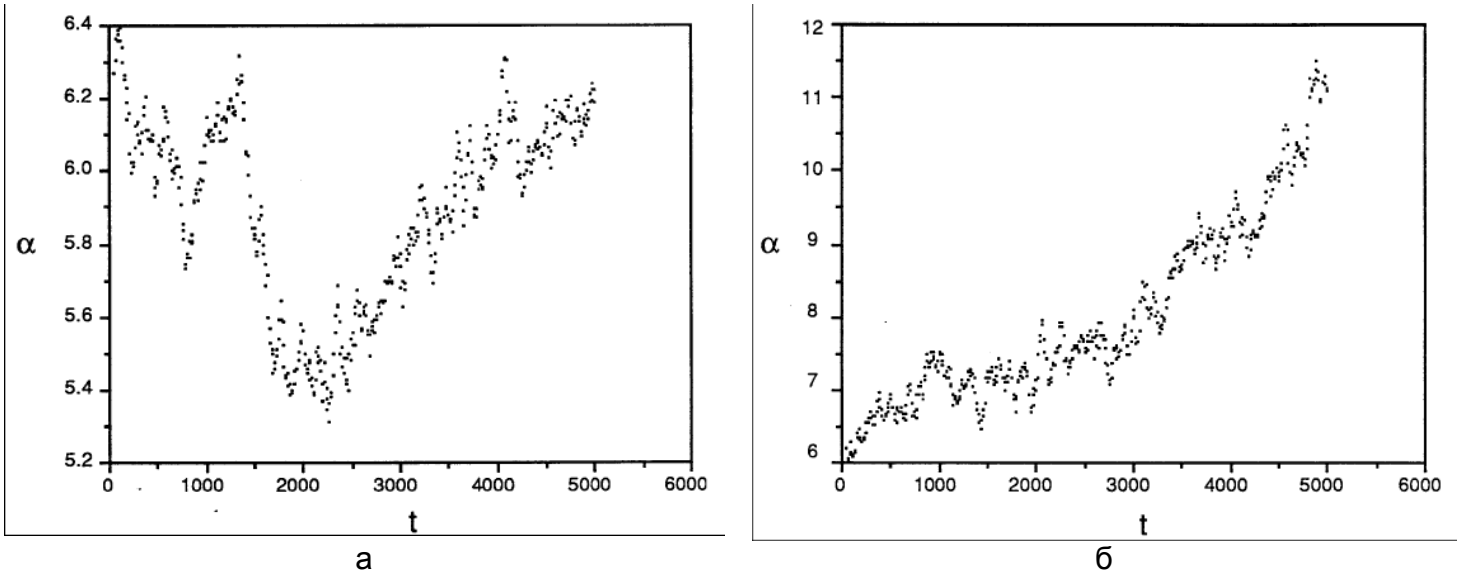
Функция приспособленности симорга вычисляется как сумма очков, набранных им в течение пяти раундов. В начале каждого раунда локальные среды симоргов инициализируется случайными значениями. Затем симорги по очереди (например, по часовой стрелке) осуществляют выходные воздействия. В зависимости от выходного воздействия симорга происходит изменение общей среды, добавление ему очков, а также тому симоргу, который последним отправил сообщение. При этом в течение раунда каждый симорг осуществляет десять входных воздействий.

На каждой итерации эволюционного алгоритма из популяции удаляется один симорг, затем выбираются два симорга для рекомбинации, в результате которой в популяцию добавляется новый симорг. Выбор симорга для удаления происходит обратно пропорционально его функции приспособленности, а выбор для рекомбинации – пропорционально данной функции.

Автомат, определяющий симорга, как особь генетического алгоритма, представляется в виде битовой строки, которая образуется следующим образом. Для каждого входного воздействия – пары (состояние общей среды, состояние локальной среды) в лексикографическом порядке в строку добавляется выходное воздействие. Выходное воздействие представляется в виде пары чисел, первое из которых равно нулю либо единице, в зависимости от того является ли воздействие действием либо сообщением, а второе равно действию (сообщению). Таким образом, длина строки, кодирующей автомат, составляет $8 \times 8 \times 2 = 128$ символов, каждый из которых является число из диапазона [0, 8]. Используемый оператор рекомбинации – двухточечная рекомбинация на строках. После рекомбинации с низкой вероятностью осуществлялась мутация полученной особи.

Размер популяции был выбран равным 100. Эволюционный алгоритм содержал 5000 итераций (поколений). За приспособленность популяции было выбрано среднее значение функции приспособленности на элементах популяции. Для изучения влияния коммуникации на приспособленности популяции было проведено два эксперимента, в первом из которых возможность коммуникации отсутствовала – общая среда постоянно изменялась случайным образом.

На рис. 10 приведены зависимости функции приспособленности популяции (α) от количества итераций эволюционного алгоритма (t) для случая, когда возможность коммуникации отсутствовала (а) и когда коммуникация была разрешена (б).



а б

Рис. 10. Результаты экспериментов

Результаты экспериментов показали, что приспособленность популяции в случае возможности коммуникации между особями значительно превосходит приспособленность в отсутствии таковой.

1.1.4. Преимущества высокоуровневого кодирование автоматов

В ряде работ из рассматриваемой области для представления автоматов в виде особи генетического алгоритма используется их кодирование в битовую строку. Данный вид кодирования является достаточно низкоуровневым. Это не позволяет сократить пространство поиска. Кроме того, представление в виде битовой строки, как правило, не является естественным для задачи управления. В подтверждение высказанных аргументов приведем ряд задач, в которых высокоуровневое кодирование привело к лучшим результатам, нежели кодирование в виде битовой строки.

1.1.4.1. Умный муравей (задача управления)

Приведем описание задачи «Умный муравей» на основе работ [34, 12]. Используется двумерный тор размером 32 на 32 клетки (рис. 11). На некоторых клетках поля расположены яблоки – черные клетки на рис. 11. Яблоки расположены вдоль некоторой ломаной линии, но не на всех ее клетках. Клетки ломаной, на которых яблок нет – серые. Белые клетки – не принадлежат ломаной и не содержат яблок. Всего на поле 89 яблок.

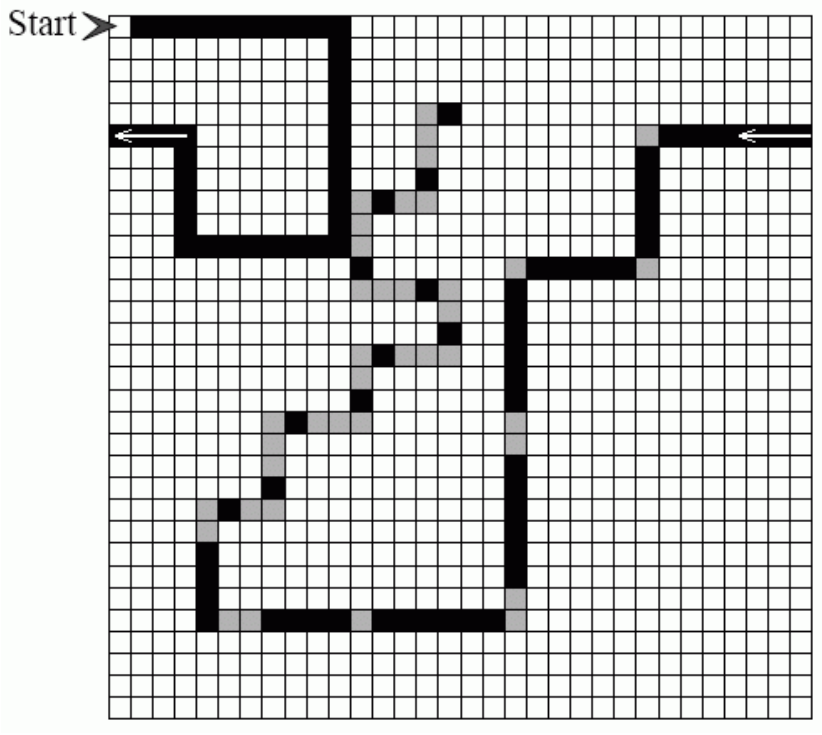


Рис. 11. Поле с яблоками

В клетке с пометкой «Start» находится муравей. Он занимает клетку поля и смотрит в одном из четырех направлений (север, запад, юг, восток). В начале игры муравей смотрит на восток. Он умеет определять находится ли яблоко непосредственно перед ним. За ход муравей совершает одно из четырех действий:

- идет вперед на одну клетку, съедая яблоко, если оно находится перед ним;
- поворачивается вправо;
- поворачивается влево;
- стоит на месте.

Съеденные муравьем яблоки не восполняются. Муравей жив на всем протяжении игры – еда не является необходимым ресурсом для его существования. Никаких других персонажей, кроме муравья, на поле нет. Ломаная *строго задана*. Муравей может ходить по любым клеткам поля. Игра длится 200 ходов, на каждом из которых муравей совершает одно из четырех описанных выше действий. В конце игры подсчитывается количество яблок, съеденных муравьем. Это значение – результат игры.

Цель игры – создать муравья, который за 200 ходов съест как можно больше яблок. Муравьи, съевшие одинаковое количество яблок, заканчивают игру с одинаковым результатом вне зависимости от числа ходов, затраченных каждым из них на процесс еды. Однако эта задача может иметь различные модификации, например, такую, в которой при одинаковом количестве съеденных яблок, лучшим считается муравей, съевший яблоки за меньшее число ходов. Ниже будет показано, что поведение муравья может быть задано конечным автоматом. При этом может быть поставлена задача о построении автомата с минимальным числом состояний для муравья, съедающего все яблоки, или автомата для муравья, съедающего максимальное количество яблок при заданном числе состояний.

Конечный автомат, изображенный на рис. 12, содержит всего пять состояний.

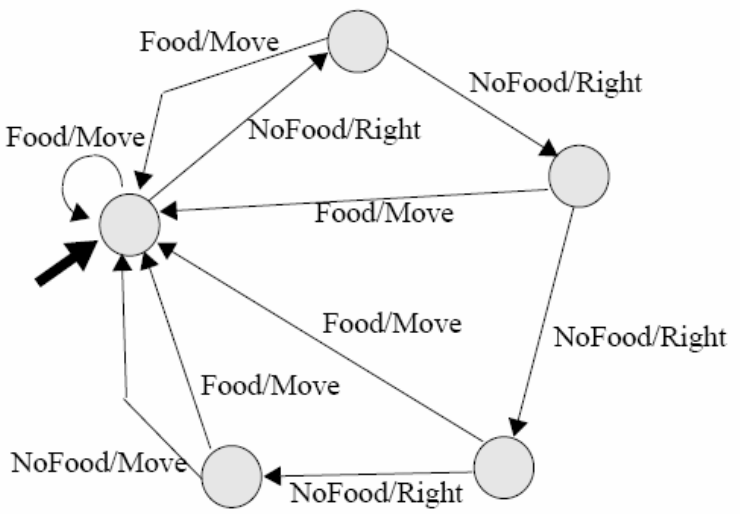


Рис. 12. Конечный автомат, задающий муравья

Этот автомат описывает поведение муравья, который не решает задачу – за 200 ходов съедает только 81 яблоко, а за 314 ходов — все 89 яблок. Муравей действует по принципу «Вижу яблоко – иду вперед. Не вижу – поворачиваю. Сделал круг, но яблок нет – иду вперед».

Приведем автомат (рис. 13), который построен в работе [34] с помощью генетических алгоритмов и решает поставленную задачу.

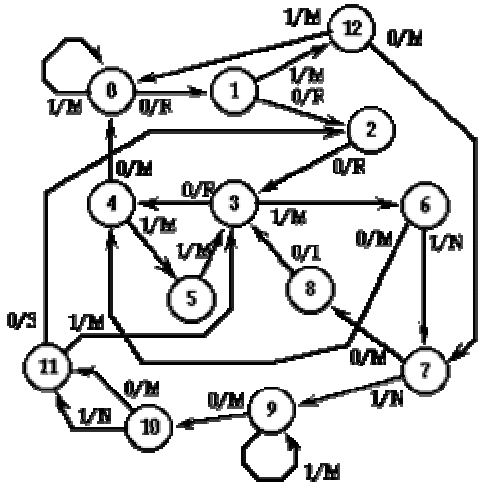


Рис. 13. Конечный автомат с 13 состояниями, задающий муравья, съедающего все яблоки за 200 ходов

На этом рисунке используются следующие обозначения: `N` – непосредственно перед муравьем нет еды, `F` – непосредственно перед муравьем есть еда; `M` – идти вперед, `L` – повернуть налево, `R` – повернуть направо, `NO` – стоять на месте. Можно заметить, что действие `NO` никогда не выполняется. Данный автомат изображен некорректно – из *одиннадцатого* состояния изображен переход, помеченный входным воздействием 0, с «непонятным» действием `S`. Однако если это действие заменить на `N`, то автомат корректно решает задачу.

Автомат из работы [12], приведенный на рис. 14, содержит 11 состояний. По утверждению авторов, он съедает все яблоки за 193 хода. Этот граф некорректен, что отмечено и указано как устранить в разд. 4.2.1.3.

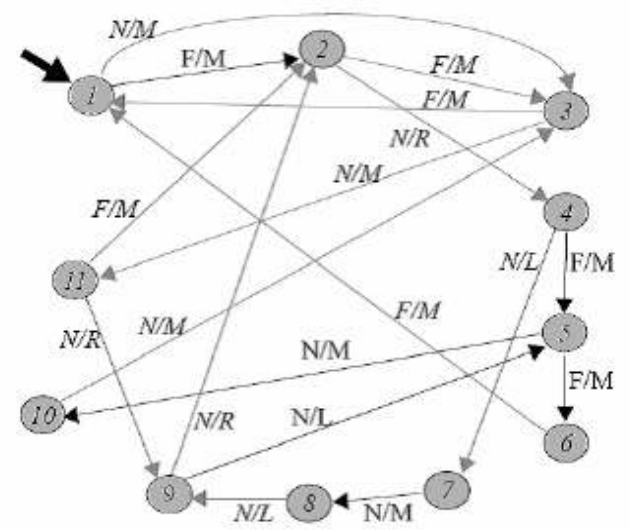


Рис. 14. Конечный автомат с 11 состояниями, задающий муравья, съедающего все яблоки за 193 хода

Для генерации конечных автоматов, задающего муравья, в работах [12, 34] были применены генетические алгоритмы. В работе [34] приспособленность особи (*fitness*) определяется как количество яблок, съеденное за 200 ходов. Кодирование автомата, задающего поведение муравья, в особь генетического алгоритма (битовую строку) в этой работе осуществлялось следующим образом: входному воздействию `F` сопоставлялась единица, а `N` – ноль. Каждое из четырех действий муравья кодировалось двоичными числами: 00 – `NO`, 01 – `L`, 10 – `R`, 11 – `M`.

Покажем, как выполняется кодирование автомата в целом. На рис. 15 приведен пример графа переходов автомата, описывающего поведение некоторого муравья.

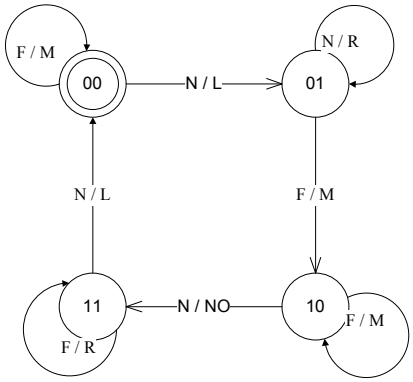


Рис. 15. Пример автомата, описывающего поведение муравья

Кодирование этого графа переходов приведено в табл. 4.

Таблица 4. Кодирование графа переходов

Состояние	Вход	Новое состояние	Действие
00	0	01	01
00	1	00	11
01	0	01	10
01	1	10	11
10	0	11	00
10	1	10	11
11	0	00	01
11	1	11	10

Преобразуем эту таблицу в битовую строку. Для этого сначала требуется запомнить число состояний автомата (в данном случае четыре). В начале строки задан двоичный номер начального состояния автомата (в данном примере – 00). Далее записаны пары (Новое состояние, Действие). Содержимое полей (Состояние, Вход) не записываются, так как они повторяются для каждого автомата с фиксированным числом состояний. Для рассматриваемого примера искомая строка имеет вид:

00 *01*01 *00*11 *01*10 *10*11 *11*00 *10*11 *00*01 *11*10

Здесь курсивом выделены биты, соответствующие новому состоянию, а обычным шрифтом — биты, соответствующие выполняемому на переходе действию. Жирным шрифтом выделены биты, кодирующие начальное состояние.

В книге [37] создатель генетического программирования *J. R. Koza* предлагает другой подход к представлению автомата, нежели кодирование с помощью битовой строки. При этом предлагается задавать поведение муравья в виде дерева выражений. Множество терминалов для данного дерева: `MOVE`, `LEFT`, `RIGHT`. Множество функций (нетерминалов): `IF-FOOD-AHEAD`, `PROGN2`, `PROGN3`. Назначение терминалов понятно. Смысл нетерминалов таков:

- `IF-FOOD-AHEAD A B` – условный переход: проверяем, есть ли яблоко непосредственно перед муравьем, и в случае если оно есть выполняем выражение `A`, иначе – выражение `B`.
- `PROGN2 A B` – безусловный переход: выполняем выражение `A`, а затем выражение `B`.
- `PROGN3` – аналогично `PROGN2`, с той лишь разницей что выполняются последовательно три аргумента данной функции.

В работе [37] отмечено, что функция `IF-FOOD-AHEAD` соответствует двум ребрам автомата с входными воздействиями «Есть яблоко», «Нет яблока» и с выходными воздействиями `A` и `B` соответственно. Заметим, что в этой работе не строит автоматов, указывая лишь, что используемая им структура аналогична конечным автоматам.

Апробация этого метода проводилась на другом игровом поле – *Santa Fe Trail*. Используемое поле приведено на рис. 16. В результате экспериментов, описанных в работе [37], был сгенерирован муравей, съедающий все яблоки на поле. На рис. 17 приведено задающее его дерево.

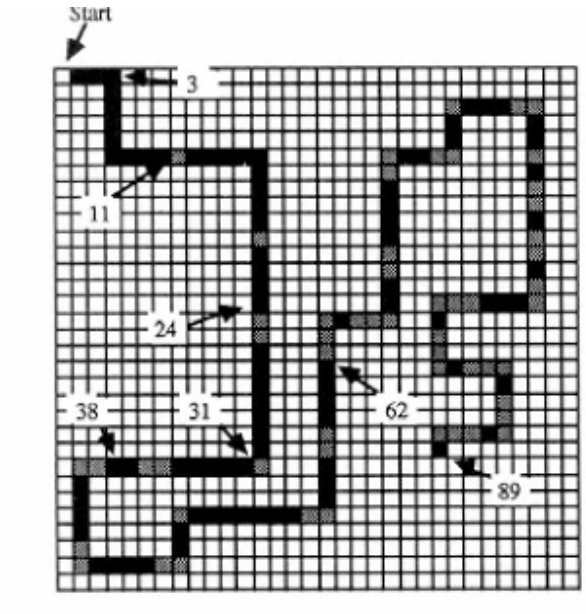


Рис. 16. Santa Fe Trail

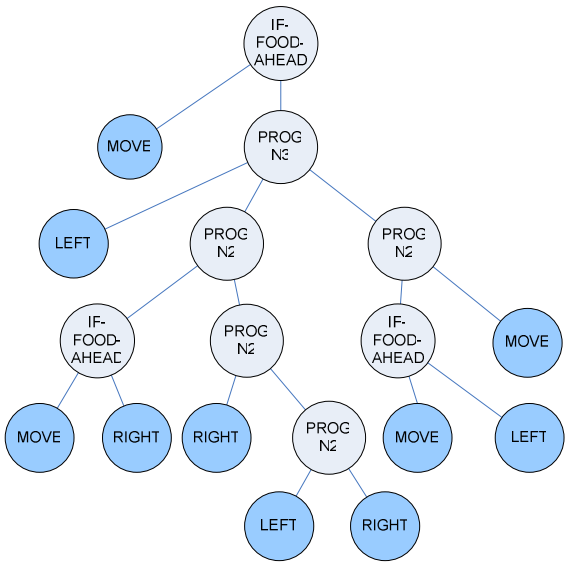


Рис. 17. Дерево выражений, задающее муравья

Автомат, соответствующий данному дереву, показан на рис. 18.

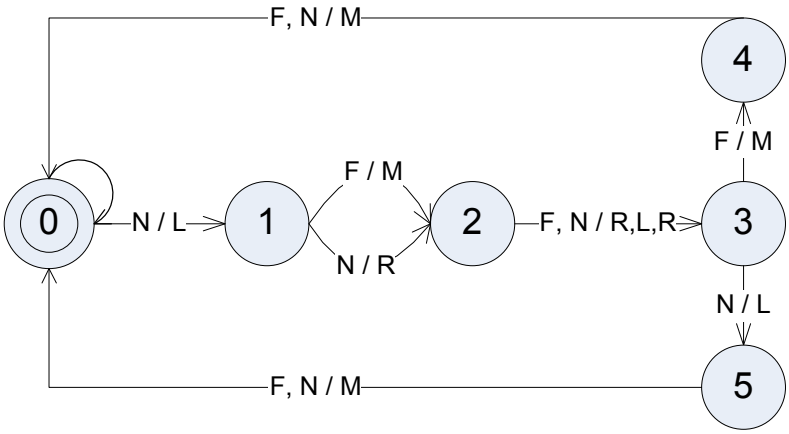


Рис. 18. Автомат, соответствующий дереву выражений

Стоит отметить, что данный муравей был выведен на 21-ой итерации (поколении) алгоритма, а количество особей в популяции равнялось 500. Таким образом, суммарное число вычислений функции приспособленности составило $21*500=10500$, в то время как в работе [34] на построение муравья, справляющегося с задачей, потребовалось 94 поколения, при том что размер популяции составлял 65536 автоматов. При этом число вычислений функции приспособленности равно $94*65536=6160384$.

1.1.4.2. Эволюционное построение клеточных автоматов

Клеточные автоматы – одни из наиболее интересных объектов дискретной математики. Идея клеточных автоматов, принадлежащая Джону фон Нейману, появилась в конце 40-х годов XX века. В нашей стране первые работы, в которых анализируются структура, поведение и свойства клеточных автоматов, появились в середине 70-х годов XX века в Московском государственном университете [2].

Большой интерес к клеточным автоматам вызван следующими фактами. Во-первых, клеточные автоматы эквивалентны, как модель вычислимости, машине Тьюринга. Следовательно, они могут быть использованы для исследования алгоритмической разрешимости различных задач как средство распознавания языков. Во-вторых, они представляют собой мультиагентную систему с рефлексивными агентами, частично наблюдающими дискретный мир (как вероятностный, так и детерминированный), а мультиагентные системы такого вида широко используются в задачах искусственного интеллекта. В-третьих, клеточные автоматы обладают интересной способностью – осуществлять распределенные вычисления. Локальность взаимодействий и слабая восприимчивость клеточных автоматов к помехам («шуму») являются их существенными достоинствами.

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 $(Px)_i = f(x_{i-r},\ldots,x_{i+r})$.

Задача классификации плотности (density classification task – DCT) является интересным и наглядным примером того, что клеточные автоматы с малым радиусом (ограниченным локальным взаимодействием) могут обладать достаточно сложным глобальным поведением, позволяющим

осуществлять вычисления. Сформулируем задачу классификации плотности для одномерных бинарных клеточных автоматов.

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 количества 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. 19 иллюстрирует данное определение.

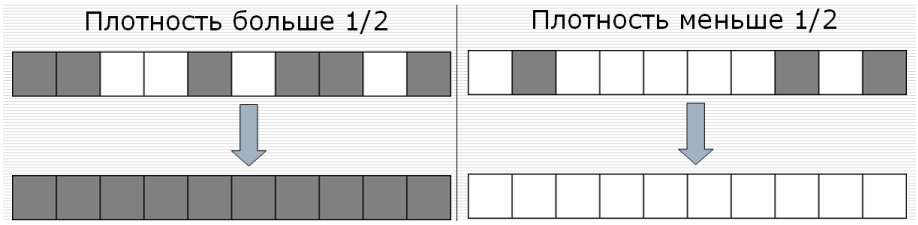


Рис. 19. Клеточный автомат распознает конфигурацию, где $\rho > \frac{1}{2}$ (слева) и $\rho < \frac{1}{2}$ (справа)

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 $g_{L,r,\rho^*}: \max_{f \in 2^{2^{2r+1}}} g_{L,r,\rho^*}(f) = g_{L,r,\rho^*}(f_{\max})$.

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

В неформальной постановке задача может быть сформулирована следующим образом: *требуется найти правило заданного радиуса, классифицирующее наибольшее число начальных конфигураций заданной длины.* На рис. 20 показан процесс распознавания автоматом начальной конфигурации, содержащей больше нулей, чем единиц (плотности меньше 0.5).

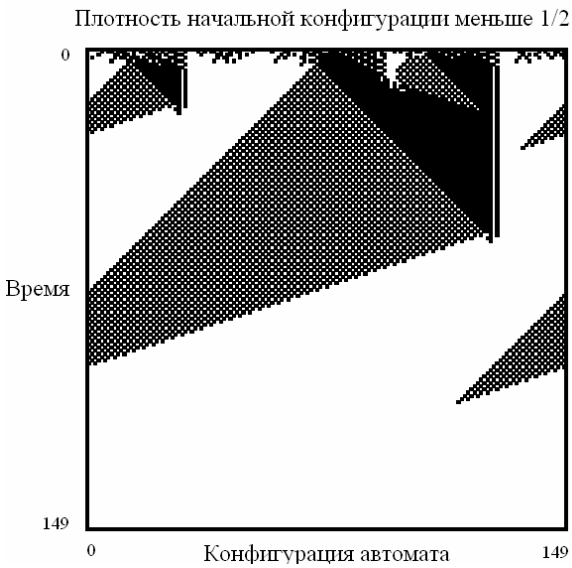


Рис. 20. Распознавания автоматом начальной конфигурации, $\rho < 1/2$

Первые публикации [44, 20], в которых поставлена и рассмотрена данная задача, появились в 1993 году. Исследователи положили $\rho^* = 1/2$. Затем выбрали значение *L* достаточно больш*и*м (равным 149) для того, чтобы вычисление индикаторной функции $I_{1/2}$ для заданного правила f на всевозможных конфигурациях *c* (а всего их 2^L) не являлось тривиальной задачей. Радиус автомата *r* был выбран достаточно большим (равным трем) для того, чтобы решение задачи невозможно было найти простым перебором всех правил (всего их $2^{2^{2r+1}}$), но в то же время достаточно малым для того, чтобы она не потеряла смысл и интерес — чтобы автоматы обладали лишь ограниченным, локальным взаимодействием. Действительно, при *r* = *L* существует тривиальное правило $f(x) = \begin{cases} 0, \rho(x)<\rho^* \\ 1, \rho(x)\geq\rho^* \end{cases}$, на котором $g_{L,r,1/2}$ принимает единичное значение. Получили задачу: найти клеточный автомат (правило, его задающее) радиуса три, классифицирующий как можно больше начальных конфигураций длины 149.

В указанных работах использовались *стандартные генетические алгоритмы, как средство для построения клеточных автоматов*, решающих рассматриваемую задачу. При ее решении возникают трудности с определением функции приспособленности – *apriori* непросто найти количество конфигураций, классифицируемых конкретным правилом, так как всего таких конфигураций 2^{149}. Поэтому авторами этих работ была введена следующая функция приспособленности: процент конфигураций из репрезентативного тестового набора, классифицируемых правилом. Тестовый набор может быть получен генерацией больш*о*го количества конфигураций случайным образом – каждый из 149 битов конфигураций равновероятно принимает значение 0 либо 1. Особь популяции, представляемая битовой строкой, задавала таблицу истинности булевой функции, определяющей правило. Применялись стандартные генетические операторы рекомбинации и мутации над строками. Однако подход оказался неэффективным – исследователями не удалось найти правило лучшее, чем *GKL*-правило созданное *без применения генетических алгоритмов* [2].

В дальнейшем для решения данной задачи применялся подход на основе генетического программирования [11]. За счет высокоуровневого представления хромосом (деревья выражений)

авторам удалось получить значительно лучшие результаты, нежели полученные в работах [44, 20] — было построено правило классифицирующее около 82.33% конфигураций.

В работе [21] для решения рассматриваемой задачи был примерен метод программирования с экспрессией генов. Хромосомы в этом методе представляются в виде так называемых *Karva*-деревьев, которые могут быть легко линеаризованы – записаны в массив (некоторый аналог – структура данных *heap*). Название метода отражает идею его работы, заимствованную из биологии: существует первичная структура хромосом – линеаризованные *Karva*-деревья, которые в последствии экспрессируются в обычные (не линеаризованные) деревья – вторичную структуру. Интересно также, что некоторые участки первичной структуры не экспрессируются. Таким образом, одной вторичной структуре (дереву) может соответствовать несколько первичных (массивов). Аналог в биологии: последовательность нуклеотидов – первичная структура белка. В дальнейшем она экспрессируется (транслируется на рибосомах) в последовательность аминокислот – вторичную структуру белка. При этом последовательность нуклеотидов содержит как транслируемые участки – *экзоны*, так и не транслируемые – *интроны*, которые удаляются в процессе *сплайсинга*. *Преимущества* данного подхода:

- высокоуровневая структура хромосом по сравнению с представлением в виде битовых строк, используемым традиционными генетическими алгоритмами;
- главное достоинство рассматриваемого метода – *Karva*-деревья обладают существенным преимуществом перед деревьями выражений (используемыми в генетическом программировании) – все генетические операции с ними корректны по определению. Таким образом, алгоритм не тратит время на проверку корректности деревьев, получаемых в результате рекомбинаций и мутаций. При этом отметим, что выполнять генетические операции над линеаризованными деревьями не сложнее и не дольше, чем над битовыми строками;
- мультигенная структура хромосом – хромосома состоит из нескольких генов;
- наличие незначащих мутаций в интронах (областях, не экспрессирующихся в деревья).

Результаты, полученные методом программирования с экспрессией генов, превосходят те, что были получены методом генетического программирования. Так в работе [21] приводится правило, классифицирующее 82.55% конфигураций.

1.1.5. Обзор методов генерации автоматов на основе генетических алгоритмов

1.1.5.1. Эксперименты Фогеля (регрессия)

Один из создателей эволюционного программирования Л.Фогель рассматривал интеллектуальное поведение индивида, как способность успешно предсказывать поведение среды, в которой он находится, и сообразно с этим действовать. В 60-х прошлого века годах Фогель поставил ряд экспериментов [24] по созданию искусственных систем, способных адаптироваться к первоначально не известной им среде.

В проведенных экспериментах Л. Фогель моделировал поведение простейшего живого существа, которое способно предсказывать изменения параметра среды, обладающего *периодичностью*. Это живое существо моделировалось конечным автоматом с действиями на переходах – автоматом Мили. В качестве среды выступала последовательность символов над двоичным алфавитом, например, $(1111010010)^*$. На вход автомата в каждый момент времени подавалась битовое значение параметра окружающей среды. Автомат формировал значение выходной переменной – возможное значение рассматриваемого параметра в следующий момент

времени. На рис. 21 приведен один из возможных автоматов, который построен для распознавания, указанной выше последовательности.

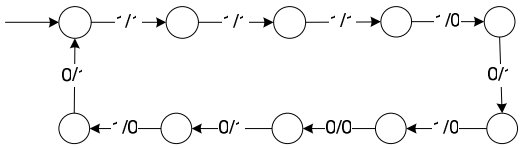


Рис. 21. Граф переходов эвристически построенного автомата

Задача состояла в эволюционном построении автомата, способного как можно более точно в смысле какой-либо разумной функции приспособленности от входа и выхода (например, количества совпавших символов) предсказывать среду – угадывать следующий символ последовательности. Кроме того, Фогель накладывал ограничения на сложность порождаемого автомата, так как строить автоматы равные длине обозреваемой последовательности не представляет труда. Таким образом, предпочтение отдавалось автоматам, угадывающим как можно лучше, и в то же время имеющим как можно меньше состояний.

В начале эксперимента задавалась периодическая последовательность символов над двоичным алфавитом, например $(101110011101)^*$. На начальной фазе выбирался префикс данной последовательности малой длины. После этого создавалась популяция автоматов с небольшим числом состояний (около пяти). Затем каждый из автоматов путем одной из пяти мутаций (выбираемой случайным образом равномерно) производил потомка. Далее над потомком подобным образом производилось еще несколько мутаций (количество определялось случайным образом). Получившийся в результате мутаций автомат добавлялся в популяцию. Были допустимы следующие мутации автомата:

- добавление состояния;
- удаление состояния (в случае, если число состояний больше единицы);
- замена стартового состояния (в случае, если число состояний больше единицы);
- замена перехода;
- замена действия на переходе.

После добавления потомков в популяцию на всех ее особях вычислялась функция приспособленности. Половина наиболее приспособленных особей переносилась в популяцию следующего поколения, а менее приспособленные автоматы – отбрасывались. Таким образом, размер популяции оставался постоянным (стоит отметить, что в силу ограниченных возможностей компьютеров того времени, он был мал – всего несколько особей). Данный процесс продолжался до тех пор, пока не удавалось достигнуть желаемого результата – максимальное значение функции приспособленности не превосходило заданного порога. После этого к битовой последовательности, которая определяла среду, добавлялся очередной символ, и эволюционный процесс переходил в очередную фазу.

По мнению Л. Фогеля, результаты экспериментов показали, что эволюционное программирование может быть успешно применено для построения «интеллектуальных» искусственных систем. При этом было отмечено, что построение вручную автоматов столь же результативных и простых, как те, что были построены эволюционным алгоритмом, является крайне сложной задачей.

1.1.5.2. Задача оптимизации

В работе [25] предлагается подход, позволяющий свести задачу глобальной оптимизации к задаче управления. Процесс нахождения глобального максимума сводится к перемещению по

дискретизированному пространству поиска. Для управления перемещением вводится автомат Мили. Сформулируем рассматриваемую задачу: задано некоторое множество $\Omega \subset R^2$, и на нем определена функция $\Phi : \Omega \to R_+$. Перемещение объекта управления по пространству Ω описывается следующим образом: $x_{t+1} = x_t + f(\Phi(x_t), \Phi(x_{t-1}))$, где $x_t \subset \Omega$, функция $f : R_+ \times R_+ \to \{x \mid x_t + x \in \Omega\}$ – стратегия перемещения. Таким образом, перемещение на текущем шаге (в текущий момент времени) зависит от значений функции Φ, исследованных на прошлом и позапрошлом шагах. Пусть $T \subset N, T > 1$ – время, отведенное на перемещение по пространству Ω. Задачей является максимизация функционала $\Psi(f,T) = \frac{\upsilon(\Phi(x_T)+1)}{\max\limits_{0 \le t \le T} \Phi(x_t) + 1} + \max\limits_{0 \le t \le T} \Phi(x_t)$ по стратегии перемещения f. Таким образом, производится поиск стратегий, которые, с одной стороны, должны искать глобальный максимум (этому требованию соответствует второе слагаемое в сумме), а, с другой стороны, к окончанию поиска текущее положение x_T должно быть «недалеко» от максимального из исследованных (этому требованию соответствует первое слагаемое). Константа υ в числителе первого слагаемого устанавливает приоритет между первым и вторым требованиями.

Функция $\Psi(f,T)$ является функцией приспособленности для генетического программирования. Стратегия перемещения f реализуется, как показано на рис. 22.

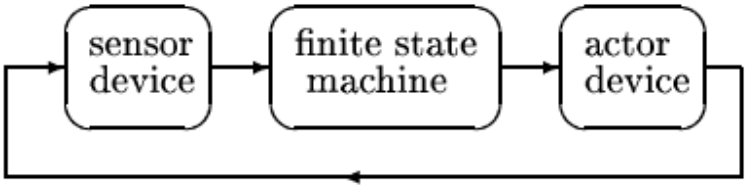


Рис. 22. Структура стратегии перемещения

Sensor device (преобразователь) переводит непрерывные значения $\Phi(x_t), \Phi(x_{t-1})$ из R_+ во множество X дискретных входных воздействий автомата Мили A, реализуя функцию $\sigma : R_+^2 \to X$. Данная функция наряду с автоматом A будет выводиться генетическим алгоритмом.

Finite state machine (автомат) A задается множеством входных воздействий X, действий Y, состояний S, начальным состоянием $s_1 \in S$, функцией переходов автомата $\delta : X \times S \to S$ и функцией действий автомата $\gamma : X \times S \to Y$.

Actor device (устройство перемещения) переводит выходные воздействия автомата в команду по перемещению объекта управления по двумерной сетке, дискретизирующей Ω. Устройство перемещения хранит информацию о текущем и предыдущем положении устройства управления x_t и x_{t-1}, а также вектор перемещения d_t, который определяет как направление перемещения, так и величину шага, которая может быть равна шагу сетки, либо удвоенному шагу сетки. В зависимости от выходных воздействий автомата устройство перемещения оставляет устройство управления на месте $x_{t+1} = x_t$, либо перемещает его в одно из четырех ортогональных направлений. При этом шаг перемещения может остаться неизменным, либо удвоиться, либо уменьшиться вдвое. Определяется множество возможных перемещений $Y = \{F, B, L, R, S, 2F, \frac{1}{2}F\}$, элементы которого соответствует перемещению вперед, назад, влево, вправо, вперед с удвоением шага, вперед с уменьшением шага вдвое.

Традиционное генетическое программирование конструирует функции σ,δ,γ одновременно – дерево, являющееся особью генетического программирования, определяет сразу три функции. Вершины дерева удовлетворяют следующей грамматике:

```
cond     ::= ≥0(<arith>,{<rule>},{<rule>})
rule     ::= γ(<state>,<output>) | δ (<state>,<state>) | <cond>
arith    ::= <arith> + <arith> | <arith> - <arith> | <arith> <arith>
| <arith> % <arith> | A | B | c∈R
state    ::= s∈S
output   ::= y∈Y
```

Дерево параметризуется константами *A, B*, которые при вычислении заменяются на $\Phi(x_t),\Phi(x_{t-1})$. Корнем дерева является *cond*-выражение. Пример дерева выражений приведен на рис. 23.

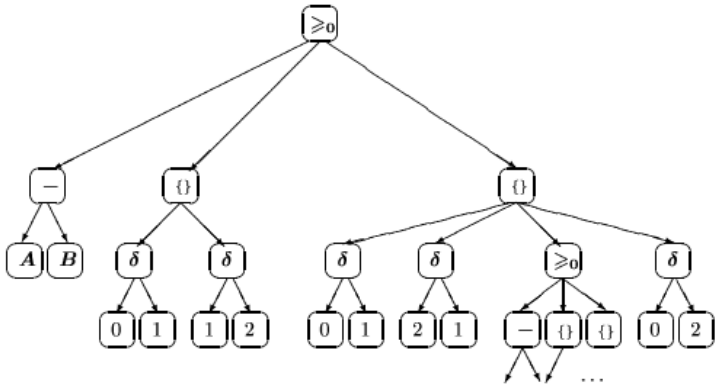


Рис. 23. Дерево выражений

Для наглядности иллюстрации в данном дереве не отображена функция γ. Результатом вычисления выражения приведенного вида после подстановки значений $\Phi(x_t),\Phi(x_{t-1})$ являются конечное множество переходов (<state>,<state>) и конечное множество выходных воздействий (<state>,<output>), которые определяют функции δ,γ. Если в множестве переходов (выходных воздействий) присутствуют два различных перехода (действия) из одного состояния, то выбирается самый левый из них (в приведенном примере это будет переход (0, 1)).

Генетические операции (рекомбинация, мутация) определены таким образом, чтобы при их выполнении порождались корректные деревья. Используются три дополнительные генетические операции над вершинами, являющимися списком инструкции (данные вершины обозначены символом { } на рис. 23): удаление поддерева, добавление поддерева, изменение порядка на поддеревьях.

Эксперименты проводились на трех различных функциях Φ:

$$\Phi_1(x) := -50\mu_1 \sum_{i=1}^{n} x_i \sin\sqrt{50x_i};$$

$$\Phi_2(x) := \mu_2 \sum_{i=1}^{n} \left(\tfrac{1}{4}x_i^2 - 10\cos(\pi x_i) + 10\right);$$

$$\Phi_3(x) := \mu_3 \sum_{i=1}^{n-1} 100\left(\tfrac{1}{5}x_{i+1} - \tfrac{1}{25}x_i^2\right)^2 + \left(\tfrac{1}{5}x_i - 1\right)^2.$$

Параметр n принимал значение равное двум, $v = 10$, $T = 50$, а константы μ_i были выбраны таким образом, чтобы $\Phi_i \in [0,1]$. Пространством поиска являлось множество $\Omega := [0,10]^2$. Автоматы содержали десять состояний. Популяция состояла из 10 автоматов, для каждого из которых случайным образом выбиралось $x_0 \in \Omega$ – положение объекта управления в начальный момент времени.

Графики первой и третьей функций приведены на рис. 24.

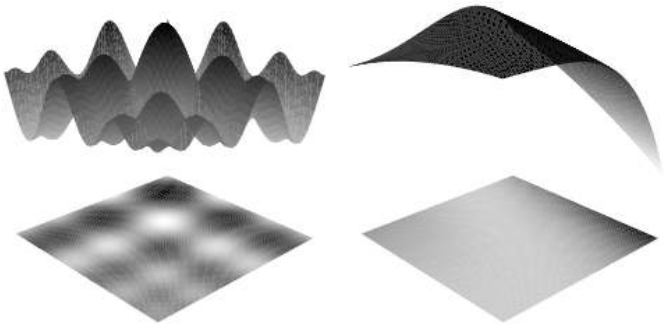


Рис. 24. Тестовые функции приспособленности Φ_1 и Φ_3

В результате экспериментов традиционное генетическое программирование сгенерировало стратегии перемещения f, для которых значение функции приспособленности близко к максимальному (единице). Кроме того, найденные стратегии являются устойчивыми как к изменению позиции x_0 объекта управления в начальный момент времени, так и к замене функции приспособленности Φ на другую, которая обладает аналогичными свойствами.

1.1.5.3. Построение автоматических преобразователей

Одной из областей применения автоматического построения автоматов является создание автоматических преобразователей (*Finite State Transducers*), которые преобразуют входной поток в последовательность символов выходного алфавита. На рис. 25 приведен пример простого преобразователя двоичной записи числа `N` в двоичную запись числа `N + 1`. При этом двоичная запись числа подается с конца.

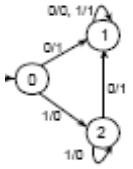


Рис. 25. Пример простого преобразователя

Более сложным примером использования автоматических преобразователей является синтез речи — отображение последовательности букв в последовательность фонем. Важным достоинством применения конечных автоматов для описания преобразователей является легкость их реализации.

В работе [45] предлагается метод генетического программирования для построения конечных автоматов, реализующих преобразователи. Автоматы в этой работе представляются в виде графов переходов, генетические операции определяются аналогично генетическим операциям над абстрактными помеченными графами. При этом ребра помечаются входным и выходным воздействиями.

Приведем описание операции рекомбинации. В используемом методе оператор рекомбинации принимает на вход две особи и возвращает две порожденных особи. Используется следующий алгоритм.

1. В двух рекомбинируемых графах выбирается по одной случайной вершине.
2. Подграфы, соответствующие вершинам меняются местами.
3. Ребра, которые ведут наружу, направляются в случайные вершины.

На рис. 26 приведен пример рекомбинации.

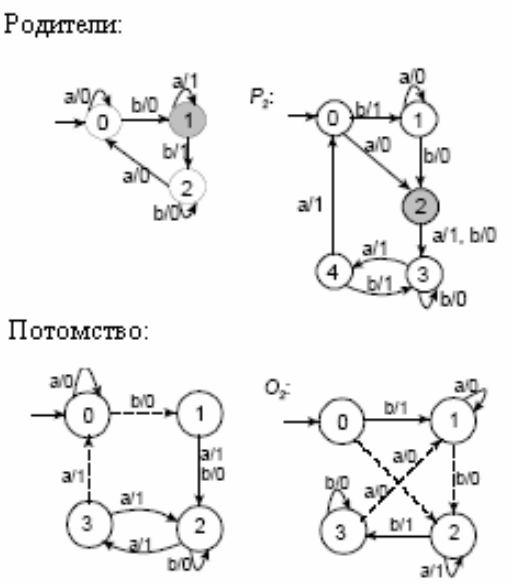


Рис. 26. Пример рекомбинации графов переходов

Операция мутации реализуется следующим образом.

1. Выбирается случайная вершина графа.
2. Подграф, соответствующий выбранной вершине, удаляется.
3. Из вершины выращивается случайный подграф.

Определенные таким образом генетические операции часто порождают потомство, имеющее меньшее значение функции приспособленности, чем у родителей. Для того чтобы исключить это, реализуется следующая эвристика: генетические операции повторяются, пока не будет достигнуто потомство, превосходящее родителей. Если же после фиксированного числа итераций приемлемого результата не достигается, то родители переходят в следующую популяцию.

Предложенный подход был проверен на шести простых задачах построения преобразователей. Результаты экспериментов подтверждают работоспособность метода.

1.1.5.4. Автоматические переговоры

Приведем описание задачи автоматических переговоров. Для начала выделяется фиксированный набор возможных предложений. После этого два переговорщика попеременно выдвигают одно из них. Любой из участников может либо принять предложение соперника, либо выдвинуть встречное предложение.

Каждый из переговорщиков стремится максимизировать известную только ему функцию полезности, определенную на множестве возможных предложений. Однако, если переговоры затягиваются, то переговорщики не получают никакой прибыли.

В работе [49] для выражения стратегий переговорщиков были применены конечные автоматы. Входными и выходными воздействиями являются возможные предложения. Кроме того, вводится дополнительное выходное воздействие, соответствующее принятию предложения.

Приведем описание используемой операции рекомбинации. Операция принимает на вход два автомата и выдает один. Для этого множество состояний каждого автомата разбивается на два подмножества. После этого подмножество первого автомата, содержащее указанную вершину, объединяется с подмножеством второго автомата, которое не содержит стартовую вершину. Заметим, что некоторые ребра теперь ведут наружу или исходят из вершин, не представленных в полученном автомате. Эти ребра изменяются для корректности автомата. Предложенный метод напоминает рекомбинацию абстрактных графов, используемую в работе [45].

Разработанный метод был апробирован на достаточно большом наборе моделей переговоров. Среди тестов присутствовали как ситуации, в которых интересы переговорщиков совпадали, так и ситуации, в которых их интересы были диаметрально противоположными. Кроме того, была осуществлена проверка на сложных функциях оценки, для которых не удается построить оптимальную стратегию. В результате было подтверждено, что метод, представляющий стратегию с помощью конечных автоматов, применим к задаче автоматических переговоров.

1.1.5.5. Распознавание нерегулярных языков

На практике языки, которые необходимо классифицировать, редко являются регулярными. Поэтому предпринимались попытки автоматизировать процесс построения распознавателей нерегулярных языков. Простейшим обобщением автомата на более широкий класс языков является *автомат с магазинной памятью*. Автомат с магазинной памятью отличается от конечного автомата тем, что может использовать в процессе работы стек: по комбинации текущего состояния, входного символа и символа на вершине стека автомат выбирает следующее состояние и, возможно, символ для записи в магазин. Может быть показано, что для любого контекстно-свободного языка можно построить недетерминированный автомат с магазинной памятью, распознающий требуемый язык. Построение же детерминированного автомата с магазинной памятью возможно не всегда. Классическим примером такого языка является язык палиндромов.

В работе [31] рассматривается задача построения недетерминированного автомата с магазинной памятью по множеству примеров. При этом используется построение автоматов, допускающих вход по пустому стеку. В автоматах запрещены ε-переходы. Может быть показано, что определенные таким образом недетерминированные автоматы с магазинной памятью по прежнему способны распознавать любые контекстно-свободные языки.

Остановимся на функции приспособленности выводимых автоматов. В работе [31] утверждается, что доля верно классифицированных тестовых слов является плохой функцией приспособленности. Возрастание значения этой функции вовсе не означает, что автомат стал лучше распознавать как слова принадлежащие языку, так и не принадлежащие ему. Приведем построенную в этой работе функцию:

$$F(M,S_+,S_-)=(cor(M,S_+)/3+pref(M,S_+)/3+stack(M,S_+)/3)\times cor(M,S_-).$$

Здесь S_+ — множество примеров цепочек, принадлежащих языку, S_- — множество примеров не принадлежащих языку цепочек, а $cor(M,S)$ — доля верно распознанных автоматом M слов из множества S.

Приведем описание функции $pref(M,S)$, смысл которой заключается в том, чтобы премировать автоматы, принимающие как можно более длинные префиксы слов, принадлежащих языку. Пусть при этом

$pref^*(M,w)=\max(\frac{|v|}{|w|})$, где v является префиксом w и принимается автоматом;

$$pref(M,S)=\frac{\sum_{w\in S} pref^*(M,w)}{|S|}.$$

Функция $stack(M,S)$ премирует автоматы, имеющие меньшее количество символов в стеке в конце разбора цепочек из множества примеров S. При этом

$stack^*(M,w)=\frac{1}{1+\gamma}$, где γ — количество символов в стеке в конце разбора цепочки w;

$$stack(M,S)=\frac{\sum_{w\in S} stack^*(M,w)}{|S|}.$$

Автоматы представляются с помощью строк. Для того чтобы избежать экспоненциального роста длины строки, в работе [31] введено ограничение на количество переходов автомата. Автомат записывается в виде строки следующим образом: для каждого перехода выписываются начальное состояние, конечное состояние, входной символ, символ на вершине стека, следующее состояние и набор символов, которые размещаются в стеке. Максимальное количество символов, которые можно положить в стек фиксируется. Поэтому все строки имеют одинаковую длину. В работе [31] в качестве рекомбинации используется двухточечная рекомбинация строк, а качестве мутации – классическая мутация над строками. Возможна запись, как в двоичную строку, так и в строку над числами.

Разработанный метод был апробирован на десяти тестовых языках. В набор входило шесть регулярных языков, три детерминированных контекстно-свободных языков и один недетерминированный контекстно-свободный язык. Для каждого из тестовых языков удалось построить соответствующий автомат с помощью предложенного генетического алгоритма. Сравнение использования битовых строк и строк над числами показало, что первые имеют преимущество.

В работе [52] предлагается другой способ применения генетического программирования для построения автоматов с магазинной памятью. В работе строится детерминированный автомат, тем самым сужая класс языков до детерминированных контекстно-свободных грамматик.

В работе используется специфическое представление автомата с магазинной памятью, названное автором диаграммой псевдосостояний. Представление аналогично диаграмме переходов конечного автомата Мура, но его состояния разделены на различные классы:

- `READ`-состояния — осуществляют чтение символа из входного потока. После этого в зависимости от прочитанного символа автомат переходит в следующее состояние;
- `POP`-состояния — осуществляют чтение символа с вершины стека. Переход выполняется в зависимости от прочитанного символа;
- `PUSH`-состояния — помещают в стек определенный символ. Они имеют единственный переход;

- ACCEPT-состояния — допускающие состояния автомата. Они не имеют выходов;
- REJECT-состояния — отвергающие состояния автомата. Они также не имеют выходов.

Пример диаграммы псевдосостояний приведен на рис. 27.

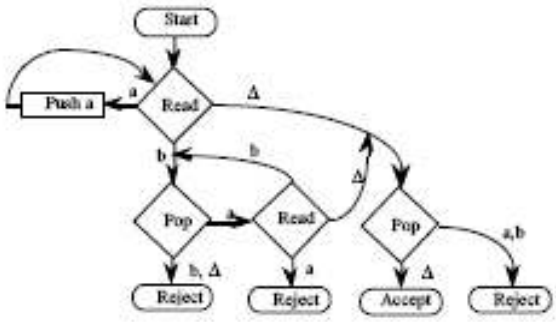


Рис. 27. Диаграмма переходов псевдосостояний

Диаграммы переходов получаются в результате вычисления выражений на специальном макроязыке *APDAL*. Выражения, соответствующие искомому автомату, выводятся с использованием традиционного генетического программирования. Для простоты входной алфавит автомата полагается равным {*a, b*}. Стековый алфавит совпадает с входным. Автор вводит следующие инструкции:

- READ — создать новое READ-состояние;
- PUSHA — создать новое PUSH-состояние, помещающее в стек символ *a*;
- PUSHB — создать новое PUSH-состояние, помещающее в стек символ *b*;
- POP — создать новое POP-состояние;
- REJECT — создать новое отвергающее состояние;
- ACCEPT — создать новое допускающее состояние.

В качестве аргументов инструкциям передаются состояния, в которые должны вести переходы, исходящие из вершины. Таким образом, инструкции READ и POP имеют два аргумента, инструкции PUSHA и PUSHB — один аргумент, а инструкции REJECT и ACCEPT не имеют аргументов.

Вершины автомата создаются и нумеруются в порядке вычисления выражения. В процессе исполнения поддерживается специальная переменная LASTSTATE, соответствующая номеру последнего созданного состояния. Значение переменной может быть передано в инструкции языка с помощью макроса TOLAST. Макрос DEC уменьшает значение переменной LASTSTATE на единицу.

Автомат, изображенный на рис. 27, описывается на макроязыке *APDAL* следующей программой:

```
(READ
  (PUSHA
    (TOLAST))
  (POP
    (READ
      (REJECT)
      (DEC (TOLAST))
      (POP (REJECT) (REJECT) (ACCEPT)))
    (REJECT)
    (REJECT))
```

```
(TOLAST))
```

С помощью разработанного подхода в указанной работе удалось построить автомат, распознающий язык правильных скобочных последовательностей, и автомат, распознающий язык $a^n b^n$. Заметим, что язык $a^n b^n$ является подмножеством языка правильных скобочных последовательностей.

1.1.5.6. Проектирование логических схем для автоматов Мили

При аппаратной реализации автоматов Мили возникает задача построения логической схемы, которая реализует данный автомат. Эта задача обычно разбивается на ряд подзадач:

1. Минимизация количества состояний автомата.
2. Нумерация состояний – сопоставление битовой строки, как номера, каждому состоянию.
3. Минимизация логической схемы.
4. Физическая реализация спроектированной схемы.

В работе [46] рассматривается решение второй и третьей задачи.

Задача нумерации состояний

Приведем пример из указанной работы, иллюстрирующий важность решения задачи нумерации состояний для минимизации количества логических элементов схемы, а, следовательно, площади ее физической реализации и стоимости изготовления. Задача состоит в том, чтобы сопоставить каждому состоянию автомата, имеющего N состояний, номер в диапазоне $[0..2^K - 1]$, где K – минимальное натуральное число, при котором $2^K \geq N$.

Табл. 5 задает некоторый автомат Мили.

Таблица 5. Таблица, задающая автомат Мили

Текущее состояние	Следующее состояние		Выходное воздействие	
Входное воздействие	0	1	0	1
q_0	q_0	q_0	0	0
q_1	q_2	q_2	0	1
q_2	q_0	q_0	1	0
q_3	q_2	q_2	1	1

Если произвести нумерацию автоматов следующим образом: $A_0 = \{s_0 = 00, s_1 = 11, s_2 = 01, s_3 = 10\}$, то будет построена схема, приведенная на рис. 28. Эта схема содержит три элемента `INV`, пять элементов `AND`, три элемента `OR` и два `D`-триггера.

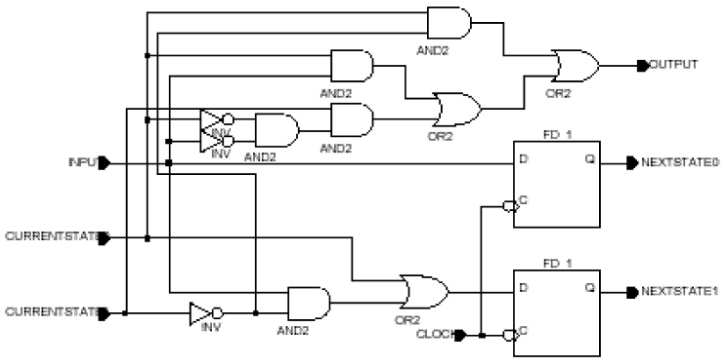


Рис. 28. Реализация автомата Мили при нумерации A_0

Если же занумеровать состояния иначе: $A_1 = \{s_0 = 00, s_1 = 10, s_2 = 01, s_3 = 11\}$, то возможна более экономичная реализация автомата (рис. 29), содержащая всего два элемента `INV`, пять элементов `AND`, два элемента `OR` и два `D`-триггера.

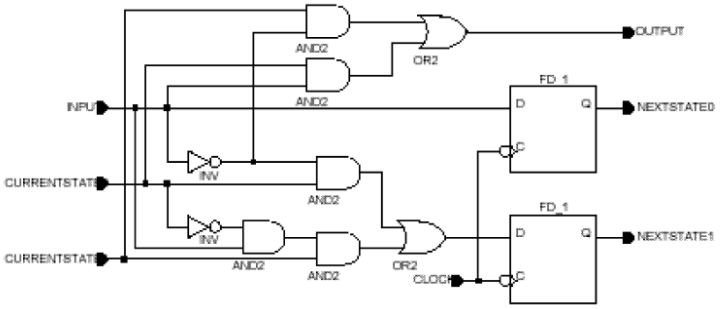


Рис. 29. Более экономичная реализация автомата Мили при нумерации A_1

Наивным решением задачи нумерации состояний является перебор всех возможных нумераций, построение минимальной схемы при заданной нумерации, выбор нумерации на которой достигается минимизация количества элементов схемы. Однако данный подход имеет экспоненциальную сложность по количеству состояний, в силу того, что такой сложностью обладает перебор всех возможных нумераций. Более того, установлено, что задача нумерации состояний (*State Assignment Problem*) является NP-полной, что делает существование полиномиального алгоритма ее решения маловероятным.

В рассматриваемой работе применяется генетический алгоритм для решения задачи нумерации состояний. Особь генетического алгоритма, задающая решение задачи, кодировалась в хромосому в виде строки длины N над целыми числами из диапазона $[0..2^K - 1]$. Пример кодирования особи указанным образом для автомата с шестью состояниями приведен на рис. 30.

S_0	S_1	S_2	S_3	S_4	S_5
4	2	1	0	7	6

Рис. 30. Особь генетического алгоритма

Для рекомбинации использовалась одноточечная, двухточечная и однородная рекомбинация строк. Мутация осуществлялась путем замены номера некоторого состояния (в позиции от 0 до $N - 1$) на другой допустимый номер. Как в случае мутации, так и в случае рекомбинации могла получиться хромосома, определяющая некорректную нумерацию, когда два состояния кодировались

одним числом, в то время как некоторому состоянию не был сопоставлен никакой номер. В этом случае особь «штрафовалась», что препятствовало осуществлению недопустимых операций в дальнейшем.

Для вычисления функции приспособленности использовалась эвристика, которая дает хорошие результаты для задачи нумерации состояний, описанная в работе [13]. Данная эвристика формулируется следующим образом:

- два состояния *a*, *b*, для каждого, из которых существует переход в состояние *c*, должны быть занумерованы «близкими» значениями;
- два состояния *a*, *b*, в каждое из которых существует переход из состояния *c,* должны быть занумерованы «близкими» значениями;
- у первого из указанных правил имеет приоритет над вторым.

«Близость» номеров состояний определяется расстоянием Хемминга между битовыми строками, задающими эти номера: номера, битовые строки которых отличаются не более чем в одной позиции, считаются «близкими», иначе не считаются таковыми. Например, номера 0101 и 1101 являются «близкими», в то время как номера 1100 и 1111 такими не являются. Для вычисления функции приспособленности подсчитывалось число случаев x, противоречащих первому требованию, и число случаев y, противоречащих второму требованию. Если при этом хромосома кодировала недопустимую нумерацию (по описанным выше причинам), то ей назначался «штраф» – некоторое натуральное число z. Прн этом функция приспособленности принималась равной $f = 2x + y + z$.

Таким образом, наиболее «хорошие» особи имели малые значения функции приспособленности, а задачей генетического алгоритма являлось нахождение особи с минимальным значением функции приспособленности.

В результате проведенных экспериментов авторам рассматриваемой работы удалось на некоторых известных контрольных задачах (*benchmark*) получить значения функции приспособленности меньшие, чем те, что были получены известными ранее методами. На рис. 31 в столбце, обозначенном *Our GA*, приведены полученные результаты. В строках таблицы приведены сравнительные результаты по *benchmark*.

State machine	*#AdjRes*	Our GA	GA [5]	NOVA1	NOVA2
Shiftreg	24	0	0	8	0
Lion9	69	21	27	25	30
Train11	57	18	19	23	28
Bbara	225	127	130	135	149
Dk14	137	68	75	72	76
Bbsse	305	203	215	220	220
Donfile	408	241	267	326	291

Рис. 31. Сравнение полученных решений с известными ранее

Впрочем, неудивительно, что программа авторов рассматриваемой статьи показала столь высокие результаты, если учесть, что другие подходы задавались лишь целью решить исходную задачу (нумерации состояний), а не минимизировать функцию приспособленности, определенную на основе эвристики решения исходной задачи.

В рассматриваемой работе также было предложено решение задачи минимизации логической схемы. Количество элементов схемы (следовательно, и площадь ее физической реализации) является лишь одним из критериев минимизации логической схемы. Рассмотренная задача нумерации состоянии ориентирована именно на этот критерий. Другим критерием является минимизация

времени обработки сигнала. Различные логические элементы обладают различным временем обработки сигнала (задержки). На рис. 32 приведено время задержки для некоторых логических элементов, вместе с обозначениями самих элементов.

Name	Symbol	Gate Code	Gate Equiv.	Delay
NOT		0	1	0.0625
AND		1	2	0.209
OR		2	2	0.216
XOR		3	3	0.212
NAND		4	1	0.13
NOR		5	1	0.156
XNOR		6	3	0.211
MUX		7	3	0.212

Рис. 32. Время задержки логических элементов

Таким образом, в указанной работе решалась задача построения схемы, которая:

- соответствует автомату Мили;
- имеет минимальную площадь;
- имеет ограниченное время обработки сигнала.

Хромосома, кодирующая схему, задавалась матрицей ячеек. Каждая из ячеек содержала два или три (для `MUX` элемента) входных сигнала, логический элемент и выходной сигнала. Выходные сигналы ячеек предыдущего уровня являлись входными для ячеек следующего уровня. В соответствии с описанной структурой, ячейка кодировалась вектором целых чисел, соответствующих входным сигналам, логическому элементу и выходному сигналу. Пример хромосомы и соответствующей ей логической схемы приведен на рис. 33

$\langle 1, 0, 2, 8\rangle$	$\langle 5, 10, 9, 12\rangle$	$\langle 7, 13, 14, 11, 16\rangle$
$\langle 2, 4, 3, 9\rangle$	$\langle 1, 8, 10, 13\rangle$	$\langle 3, 11, 12, 17\rangle$
$\langle 3, 1, 6, 10\rangle$	$\langle 4, 9, 8, 14\rangle$	$\langle 7, 15, 14, 15, 18\rangle$
$\langle 7, 5, 7, 7, 11\rangle$	$\langle 4, 10, 11, 15\rangle$	$\langle 1, 11, 15, 19\rangle$

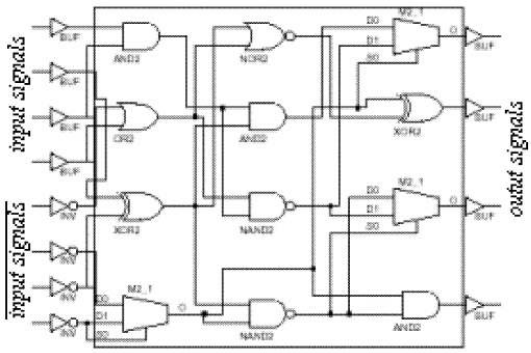


Рис. 33. Пример хромосомы, кодирующей схему

В качестве рекомбинации использовалась четырехточечная рекомбинация матриц. Оператор мутации изменял либо входные сигналы, либо логический элемент. При вычислении функции приспособленности учитывались с определенными весами число элементов в схеме и время обработки сигнала.

В работе отмечено, что схемы, сгенерированные с помощью описанного выше эволюционного алгоритма, на ряде контрольных задач имели лучшие характеристики, нежели схемы, полученные другими методами.

1.1.5.7. Построение управляющей программы для человекоподобного робота (роботехника)

Создание человекоподобных роботов одно из наиболее актуальных и перспективных направлений науки. Роботы такого типа могут бы выполнять различные задачи в мире, «приспособленном для человека». Одной из задач роботехники является создание двуногих роботов, способных передвигаться подобно человеку.

В работе [51] предлагается использовать традиционное генетическое программирование для эволюционного построения автомата, осуществляющего управление движением робота *Elvina.* Данный робот изображен на рис. 34.

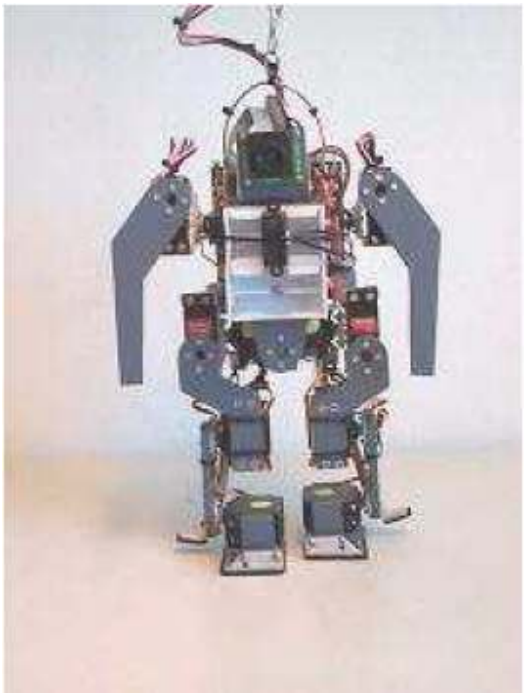


Рис. 34. Человекоподобный робот *Elvina*

Каждая из ног робота *Elvina* имеет пять степеней свободы (одна из которых пассивная). Голова туловище и руки имеют одну степень свободы. Таким образом, в сумме у робота 14 степеней свободы. Двигатели, изменяющие положение частей тела (ног, рук, головы, туловища), управляются контроллером, который выдает одно из 256 значений (0..255) определяющих положение управляемой части тела. Для управления ходьбой необходимо задать последовательность состояний робота, осуществляющую один цикл ходьбы.

Неподвижное состояние робота задается вектором $J_P = [j_1, j_2, ..., j_k]$, где k – число его степеней свободы. Тогда пространство W всех состояний робота (включая те, в которых он находится

во время перемещения частей тела) – линейная оболочка векторов $J_1, J_2, ...J_j$, где $j = 256^k$. Некоторые из этих состояний не являются статически устойчивыми – переход в такое состояние приводит к падению работа. Выделяют подпространство *G* векторов из пространства *W*, определяющих статически устойчивые положения робота. Для осуществления одного цикла ходьбы достаточно задать *m* устойчивых состояний $P_1, P_2, ..., P_m$, в которых последовательно будет находиться робот в процессе движения. Поскольку пространство поиска достаточно велико, в работе построена функцию перехода $f(j_1, j_2, ..., j_k) = [j'_1, j'_2, ..., j'_k]$, которая принимает текущее состояние робота и возвращает состояние, в которое необходимо перейти. *Данная функция реализует автомат без входных воздействий, действиями на переходах в котором являются команды двигателям частей тела об изменении положения.*

Для построения искомой функции используется традиционное генетическое программирование. Вводятся 2*k* регистров, половина из которых соответствует входным данным $j_1, j_2, ..., j_k$, а половина – выходным $j'_1, j'_2, ..., j'_k$. Как входные, так и выходные регистры инициализируются значениями $j_1, j_2, ..., j_k$. Особь алгоритма представляет собой набор инструкций, записанных последовательно, которые осуществляют операции над содержимым регистров. Каждая инструкция – четверка (*k1*, *k2*, *op*, *res*). Первые два параметра – номера регистров. Третий параметр – одна из пяти функций, аргументами которой являются первые два параметра. Последний параметр – выходной регистр, в который следует записать результат операции. После выполнения последней инструкции значения выходных регистров копируются в регистрах $j'_1, j'_2, ..., j'_k$.

Функция приспособленности вычислялась путем моделирования поведения робота в течение, примерно, 20 с. Записывались начальные координаты частей тела и конечные. Затем значение функции принималось равным $fitness = W[1 - \frac{h_{start}}{h_{stop}}] - (d_{left} + d_{right})$. Здесь $\frac{h_{start}}{h_{stop}}$ – отношение начальной высоты положения робота к конечной, $d_{left} + d_{right}$ – отклонение робота от прямолинейной траектории, *W* – некоторый постоянный коэффициент. Таким образом, предпочтение отдавалось роботам, способным поддерживать высоту положения (тем более не падать), которые как можно меньше отклоняются от прямолинейной траектории движения.

В результате применения генетического алгоритма удалось построить автомат, управляющий двуногим роботом, который справляется с задачей прямолинейного человекоподобного движения.

1.2. Требования к методам генерации автоматов управления системами со сложным поведением

Используемые в настоящие время методы почти всегда значительно используют специфику решаемой задачи, делая полученную технику генерации автоматов неприменимой к остальным задачам. Возникает вопрос создания единой технологии, применимой к широкому кругу задач. Специфика задачи может быть учтена в подсчете функции приспособленности или других параметрах используемого генетического алгоритма, но не должна ограничивать применимость разработанного подхода.

Автоматы, используемые на практике, часто имеют сложные условия переходов, описываемые булевыми формулами, содержащими нескольких переменных, в то время, как в практически во всех аналогах, переходы помечаются одиночными символами входных воздействий.

Как правило, эффективность алгоритма зависит от параметров запуска, таких как размер популяции, вероятности различных мутаций и т.д.

Часто подбор параметров генетического алгоритма производится на интуитивном уровне, так как пока не существует объективных доказательств преимущества тех или иных настроек. Однако не следует забывать, что сама суть генетических алгоритмов заключается в динамике и «мягкости» алгоритма и производимых вычислений. Таким образом, возникает следующее требование: алгоритм должен сам настраиваться во время решения задачи и адаптироваться к ней.

Одной из целей работы является возможность дальнейшей модификации сгенерированного автомата. Следовательно, представление автомата должно быть понятно человеку. С этой целью актуально разложение автомата в систему взаимодействующих и вложенных автоматов, что может существенно упростить логику системы для восприятия. Также актуален вопрос минимизации количества состояний автомата. Заметим, что иногда системы сложны настолько, что не могут быть преобразованы в понятный человеку вид.

Эффективность разработанного метода должна быть как минимум сравнима с аналогами: с его помощью должны находиться решения, не хуже, чем при использовании известных алгоритмов. Время работы генетического алгоритма не столь критично, как результат, однако, и оно должно быть приемлемым.

Таким образом, разработанный метод должен позволить генерировать конечные автоматы, с переходами, помеченными сложными булевыми формулами, содержащими до 10 символов входных воздействий. Время, затрачиваемое на генерацию таких автоматов, не должно превышать 300 часов.

1.3. Выводы

На основании рассмотренных задач и их решений, можно сделать вывод о том, что применение эволюционных алгоритмов для построения автоматов позволяет в ряде случаев получать более эффективные решения задач, нежели решения построенные вручную.

При этом использование высокоуровневых способов кодирования особи оказывается предпочтительнее, чем низкоуровневые описания, в силу того, что позволяет сократить пространство поиска. Отметим, что, большинство работ посвящено описанию решения отдельных задач уникальными методами. При этом не известны работы, в которых выполняется сравнение различных методов при решении одинаковых задач.

Также отметим, что рассмотренный в работе [18] подход к генетическому построению систем автоматов является весьма интересным. К сожалению, в указанной работе, рассмотрен только один способ взаимодействия автоматов — совместное принятие решений, который не может быть обобщен на другие варианты взаимодействия автоматов.

Таким образом, основными задачами исследований, проводимых в рамках этапа, должны быть:

- сравнение различных методов построения автоматов, анализ их достоинств и недостатков на ряде известных задач, для которых ранее подобного анализа не проводилось;
- разработка новых высокоуровневых способов представления автоматов в виде особей генетического алгоритма, обладающих б*о*льшей эффективностью, чем известные способы;
- рассмотрение генетических операторов, соответствующих разрабатываемым представлениям автоматов.

2. БАЗОВЫЕ КОМПОНЕНТЫ ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ, ПРЕДНАЗНАЧЕННЫЕ ДЛЯ ГЕНЕРАЦИИ АВТОМАТОВ УПРАВЛЕНИЯ СИСТЕМАМИ СО СЛОЖНЫМ ПОВЕДЕНИЕМ

В настоящем разделе описаны базовые компоненты генетического программирования, предназначенные для генерации конечных автоматов, управляющих системами со сложным поведением.

Генетические алгоритмы и генетическое программирование являются основными формами эволюционного моделирования. Каждая из этих форм имеет отличительные черты и особенности. С практической точки зрения необходимо установить для каких конкретных задач лучше подходит та или иная форма моделирования эволюции. Форма моделирования неразрывно связана с используемой структурой хромосом.

В работе [3] отмечено, что доказано то, что не существует идеальной структуры представления хромосом, а для создания хорошей структуры требуется анализ, перебор и эвристические подходы.

Поэтому ниже обоснование выбора структуры хромосом выполняется на примерах некоторых классов задач, решение которых состоит в построении автоматов.

2.1. Методы описания структуры хромосом

Для использования генетических алгоритмов необходимо задать способ представления автомата в виде хромосомы. В данной работе будет проведен анализ существующих подходов к представлению автоматов в виде хромосомы, а так же предложено несколько новых способов представления.

2.1.1. Анализ существующих структур хромосом

В работах [12, 34] используется следующий подход к кодированию автоматов.

1. Фиксируется число состояний автомата.
2. Логика работы автомата представляется в виде таблицы, в которой для каждого состояния и каждого входного воздействия записывается пара (новое состояние, действие).
3. Полученная таблица записывается в виде строки.

Поясним на примере, как таблица преобразуется в строку. На рис. 35 изображен граф переходов автомата, который может быть представлен в виде табл. 6.

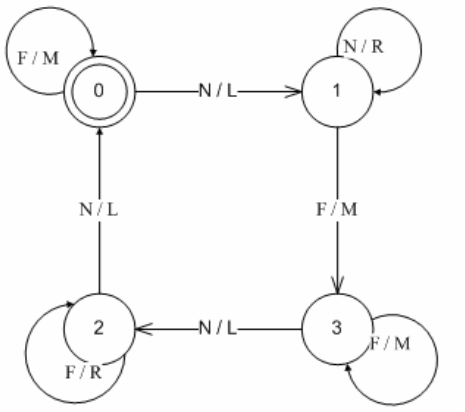


Рис. 35. Пример управляющего автомата

Таблица 6. Таблица переходов управляющего автомата

Состояние	Вход	Новое состояние	Действие
0	N	1	L
0	F	0	M
1	N	1	R
1	F	3	M
2	N	0	L
2	F	2	R
3	N	2	L
3	F	3	M

Теперь можно записать элементы этой таблицы в строку, выписав двойки (новое состояние, действие), соответствующие переходам. Для этого переходы упорядочиваются в порядке возрастания начального состояния, а в случае равенства – по номеру входного воздействия. Далее возможно применение одного из двух подходов: битовые строки и строки над числовым алфавитом.

Кроме того, возможно хранение начального состояния в строке, как это показано в разд. 3.1.4.1. Однако, анализ, проведенный в работе [12], показывает, что такой вариант не дает выигрыша по сравнению с зафиксированным стартовым состоянием.

Для задачи построения автомата, распознающего регулярный язык, кроме того, необходимо ввести для всех состояний бит, указывающий на то, является ли это состояние допускающим.

Все таблицы, соответствующие состояниям одного автомата, имеют одинаковую размерность, так как число входных воздействий и действий объекта управления задано по условию задачи. Что касается управляющих состояний, то их число также должно быть известно.

Известен подход, при котором количество управляющих состояний задается перед началом оптимизации и далее не изменяется. При таком подходе число состояний можно задать исходя из априорных представлений о сложности задачи, причем задать 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.

Теперь опишем генетические операторы над хромосомами состояний, записанными в виде полных таблиц.

На рис. 36 приведена иллюстрация процесса адаптации к предложенному представлению управляющих автоматов одного способа скрещивания, известного как одноточечное скрещивание. Руководствуясь схожим подходом, можно адаптировать к табличному представлению и другие способы скрещивания.

Алгоритм мутации состояния, представленного таблицей, проиллюстрирован на рис. 37 (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.

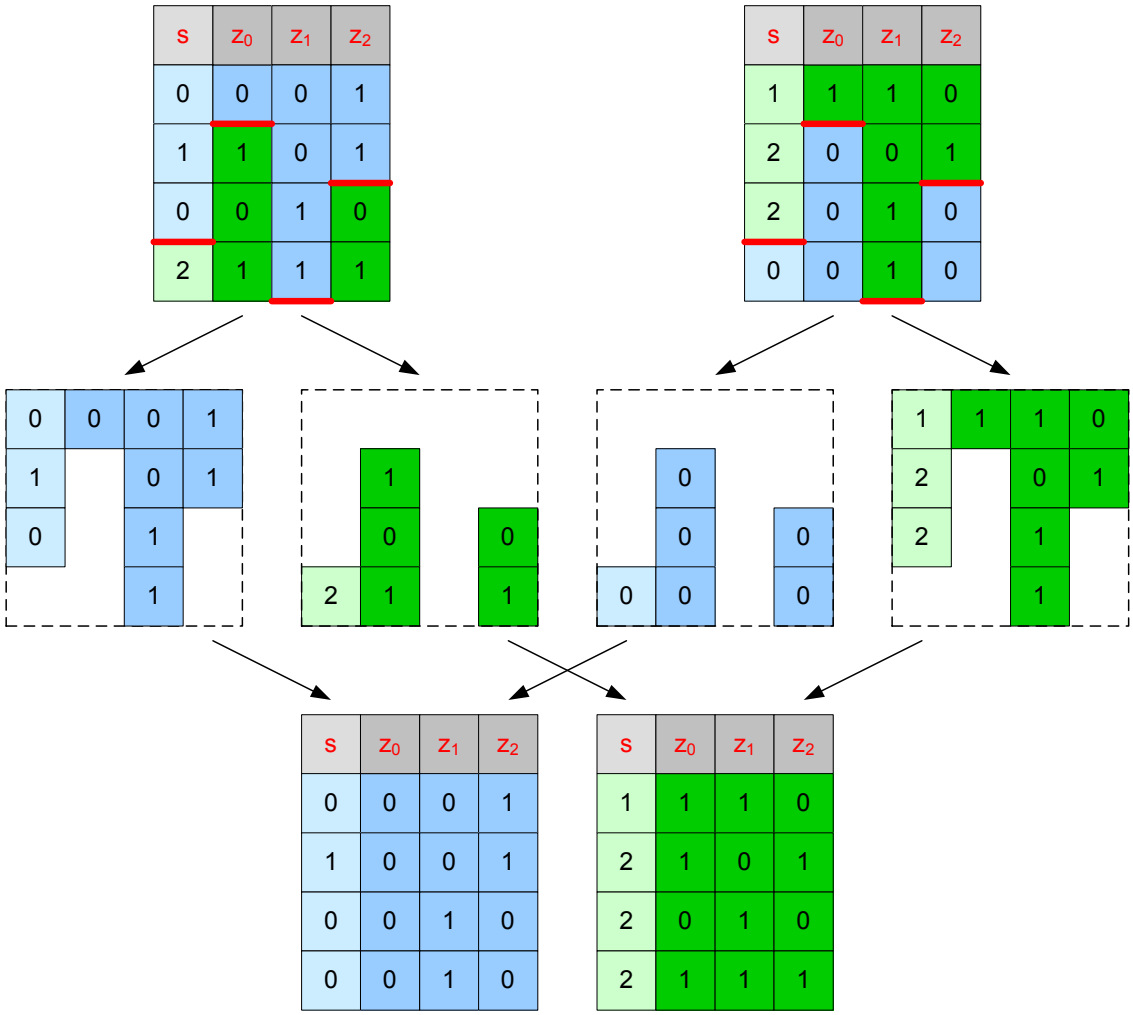


Рис. 36. Пример скрещивания полных таблиц переходов

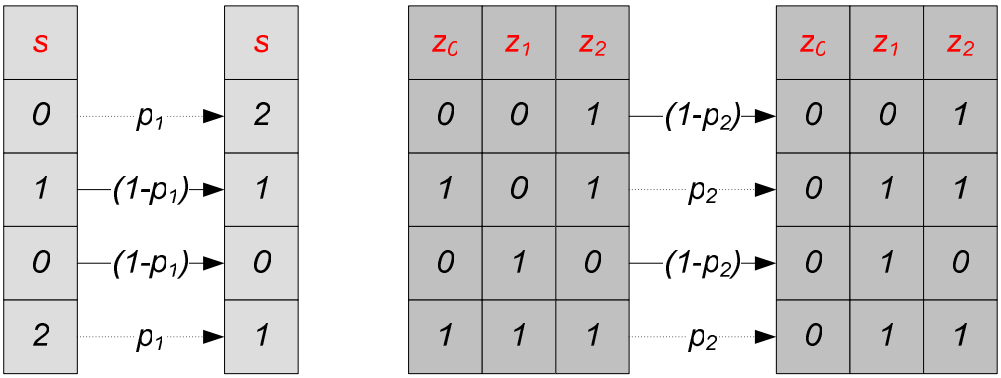


Рис. 37. Пример мутации полных таблиц переходов

Вывод автоматов с помощью генетических алгоритмов может иметь низкую эффективность в силу следующих причин.

- Автомат, соответствующий сгенерированной строке, может иметь недостижимые состояния. Информация, хранящаяся для этих состояний, является генетическим мусором.
- Представление не является естественным – двум разным строкам может соответствовать один и тот же автомат (одному фенотипу может соответствовать два и более генотипов). Это приводит к тому, что изоморфные автоматы соревнуются друг с другом, замедляя работу генетического алгоритма.

Одним из возможных подходов к решению данных проблем может являться применение эвристического оператора переупорядочивания. В качестве такого оператора на полных таблицах переходов может выступать метод *MTF (Move To Front)* [19]. Он заключается в перенумеровывании вершин автомата для приведения всех изоморфных автоматов к каноническому виду. Для этого из стартовой вершины запускается обход в ширину. После этого вершины нумеруются в порядке обхода. Заметим, что недостижимые вершины автоматически помещаются в конец таблицы, соответствующей автомату. Пример такого преобразования приведен на рис. 38.

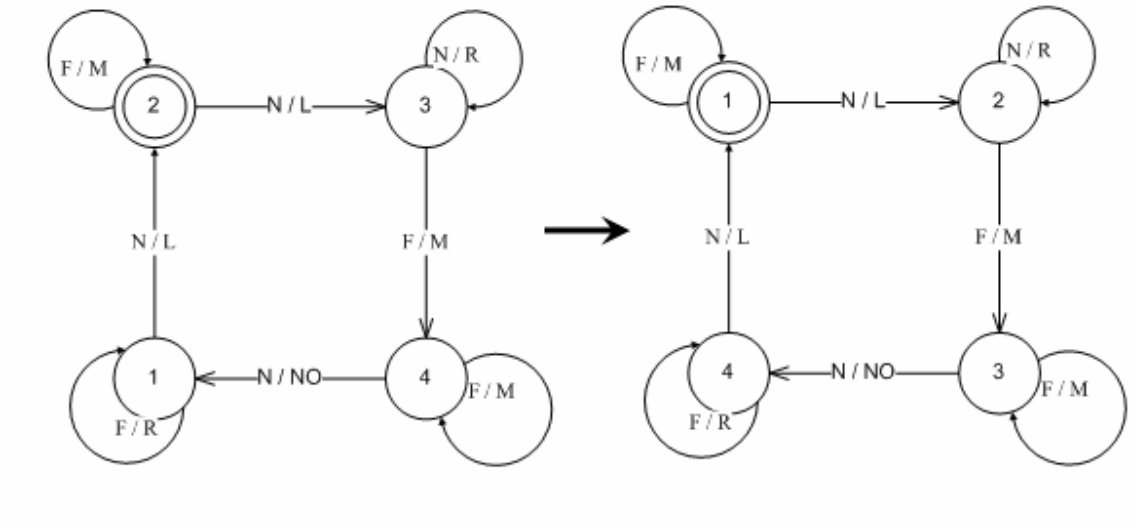


Рис. 38. Пример применения эвристики MTF

В данном примере обход графа переходов начинается из вершины, первоначально имеющей номер 2, и заканчивается вершиной 1, пройдя по пути 2 – 3 – 4 – 1. После этого производится перенумерация вершин согласно этому пути.

Анализ, проведенный в работе [19], подтверждает предположение о том, что эвристика *MTF* ускоряет вывод автоматов.

2.1.1.1. Битовые строки

Использование битовых строк для представления полных таблиц переходов накладывает следующее ограничение – для того, чтобы строке всегда соответствовал корректный автомат необходимо, чтобы число его состояний и выходных воздействий было степенью двух. Как правило, это ведет к добавлению фиктивных выходных воздействий автомата.

2.1.1.2. Строки над числами

При использовании строк над числовым алфавитом проблем с корректностью не возникает. Заметим, что алфавит для состояний и выходных воздействий может различаться. Этот метод эквивалентен методу, использующему представление автомата в виде таблицы переходов [16].

2.1.1.3. Достоинства и недостатки

Достоинством применения генетических алгоритмов над строками фиксированной длины являются простота реализации и естественность представления для генетических алгоритмов, что позволяет применять различные виды мутаций и скрещиваний для достижения лучшего результата.

Однако этот метод плохо подходит для задач со многими входными переменными, так как приходится задавать переходы автомата для всех комбинаций значений переменных. Длина строки, соответствующей автомату, растет экспоненциально. Поэтому увеличиваются затраты памяти на хранение данного представления и растет время работы генетического алгоритма.

Опыт показывает, что в реальных задачах, управляющие автоматы, построенные вручную, имеют гораздо меньше переходов. Причина этого, видимо, в том, что в большинстве задач входные воздействия имеют “локальную природу” по отношению к управляющим состояниям. В каждом состоянии значимым является лишь определенный, небольшой набор входных воздействий, остальные же не влияют на выбор перехода. Поэтому возникает задача разработки представления, которое бы позволило ограничивать число значимых для состояния воздействий. Кроме того, навязывание этого свойства в процессе оптимизации позволяет получить результат, более похожий на автомат, построенный вручную, а, следовательно, более понятный человеку.

2.1.2. Предлагаемые структуры хромосом

Как следует из предыдущего раздела, низкоуровневая запись хромосомы не очень удобна. В генетическом программировании распространен другой подход: для каждого класса оптимизируемых моделей вычисления выбирается свое представление, обладающее высокоуровневой структурой. Определение генетических операторов в терминах этой структуры с учетом семантики хромосомы позволяет значительно ускорить процесс эволюции. В данном разделе описывается три варианта представления автоматов в виде хромосомы.

2.1.2.1. Деревья решений

Дерево решений является удобным способом представления дискретных функций, зависящих от конечного числа дискретных атрибутов. Оно представляет собой помеченное дерево (рис. 39):

- в листьях записаны значения целевой функции;
- в остальных узлах – атрибуты, по которым различаются случаи;

- на ребрах записаны значения атрибутов, от которых зависит целевая функция.

Гипотезой [6] будем называть путь от корня до листа. Таким образом, дерево решений представляет собой набор упорядоченных гипотез, соответствующих листьям этого дерева.

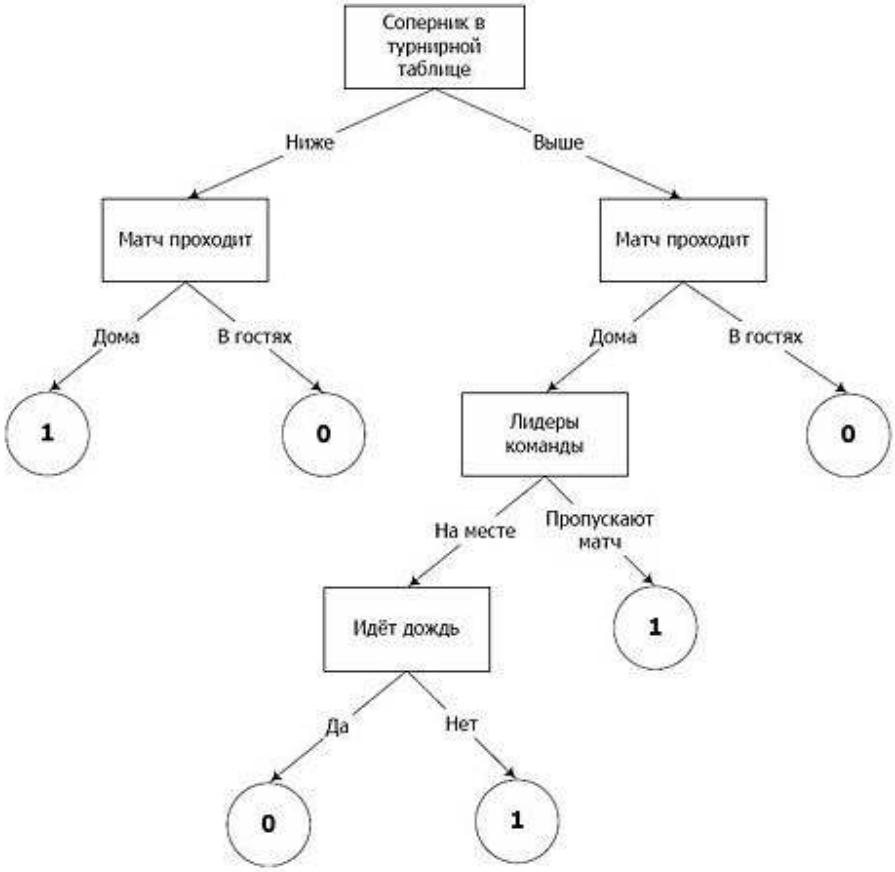


Рис. 39. Пример дерева решений

Такая структура деревьев накладывает ограничение – атрибуты на любом пути от корня до листа должны быть различны, иначе в дереве будут недостижимые листья. Действительно, если какой-нибудь атрибут встречается на пути дважды, то его значение уже известно. Следовательно, одна из ветвей будет недостижима. Само по себе это не ведет к некорректности деревьев, но может вести к нарастанию высоты деревьев с ходом эволюции, что будет экспоненциально увеличивать затраты памяти на их хранение. Кроме того, недостижимые поддеревья, являющиеся плохими приближениями искомой функции, могут копироваться в порождаемые в процессе кроссовера особи, что может замедлять работу генетического алгоритма. Таким образом, нам понадобится введение операции устранения подобной ошибки – операции обрезки. Она может быть реализована следующим образом: узел, одна из дочерних вершин которого недостижима, заменяется на свою достижимую дочернюю вершину.

В случае, когда деревья решений выражают переходы автомата (каждое дерево представляет собой управляющее состояние), атрибутами выступают входные воздействия, а целевой функцией – пара “функция перехода и функция действий”. Допускается, что набор гипотез в данном случае может быть неполон. Тогда для обеспечения полноты целесообразно ввести значение функции по умолчанию.

Отметим также, что данное представление позволяет естественным образом ввести ограничение на число используемых в гипотезе атрибутов и на число переходов, так как число атрибутов соответствует глубине дерева, а число переходов – количеству листьев. Ограничения можно задать следующим образом: ввести функцию штрафа за нарушение ограничений и прибавлять ее к фитнесс-функции. Таким образом, неподходящие под ограничение хромосомы будут отсеяны самим эволюционным процессом. Возможно применение и других подходов, например повторение генетических операций до получения особи, удовлетворяющей ограничениям.

Скрещивание управляющих автоматов в таком случае производится путем попарного скрещивания всех состояний, представляющих данные автоматы. Для скрещивания двух деревьев решений *D1* и *D2* (рис. 40) необходимо выбрать два случайных узла *d1* и *d2* в скрещиваемых деревьях и поменять местами поддеревья, соответствующие этим узлам (рис. 41).

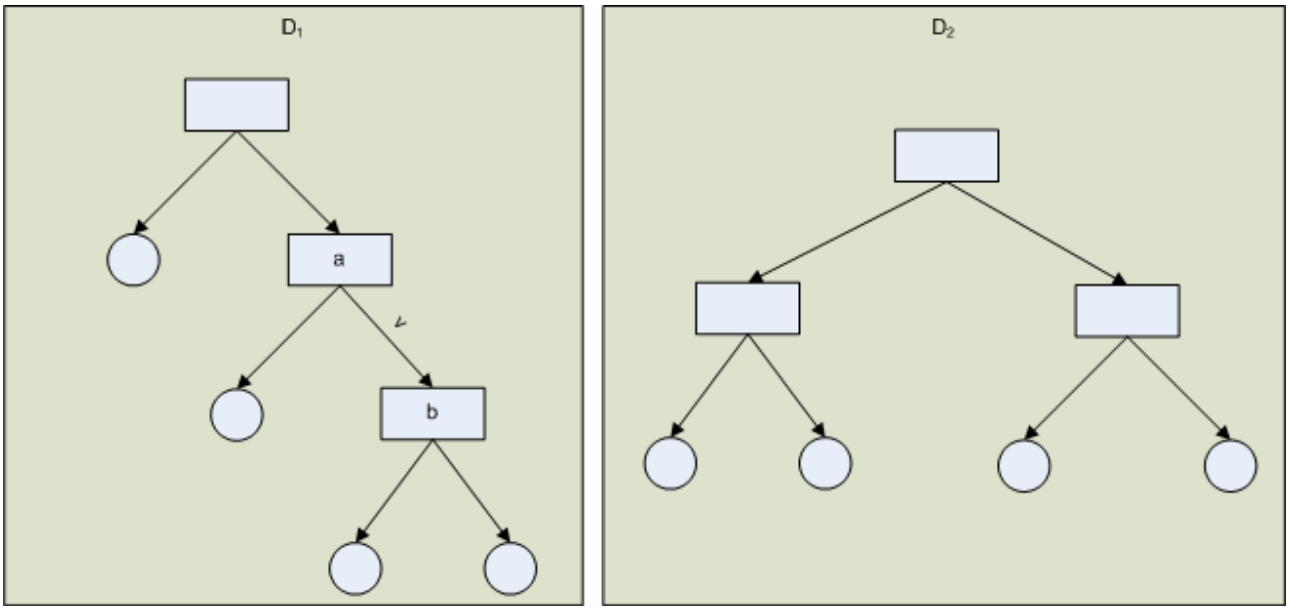


Рис. 40. Пример скрещиваемых деревьев решений

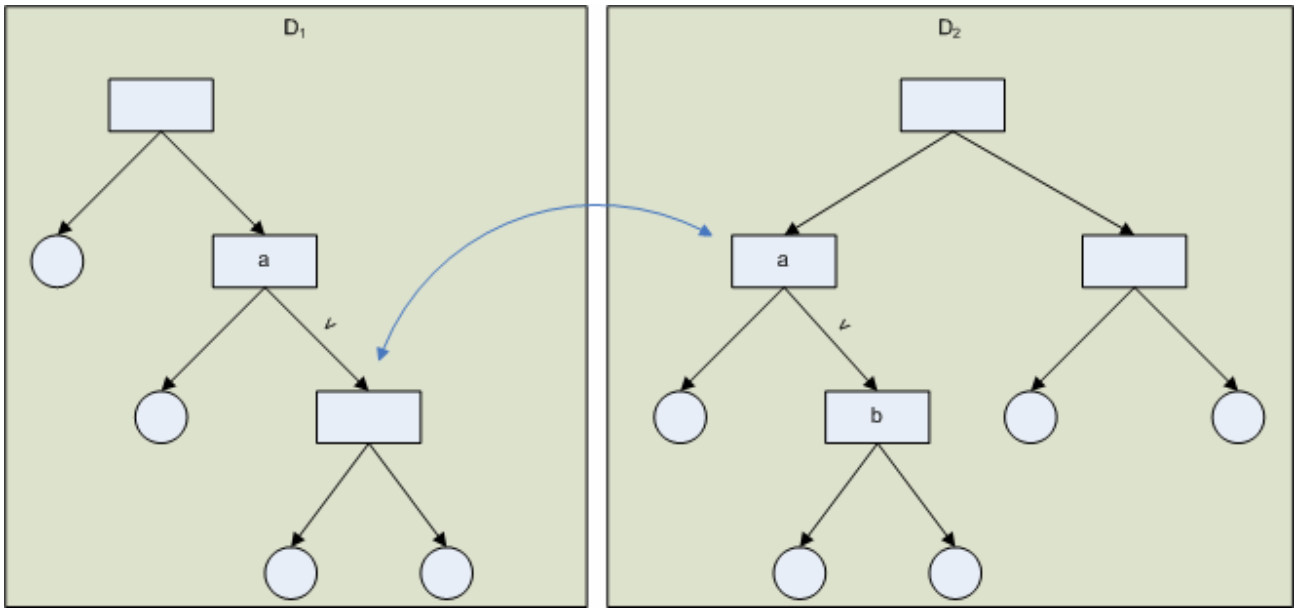


Рис. 41. Скрещивание деревьев решений

После этого необходимо произвести обрезку лишних (недостижимых) ветвей в дереве. Операция выполняется однозначно, так как если какой-то атрибут *a* встречается на пути дважды, то его значение *v* уже известно. Поэтому узел, в котором атрибут повторился можно заменить на того из детей, который соответствует известному значению атрибута.

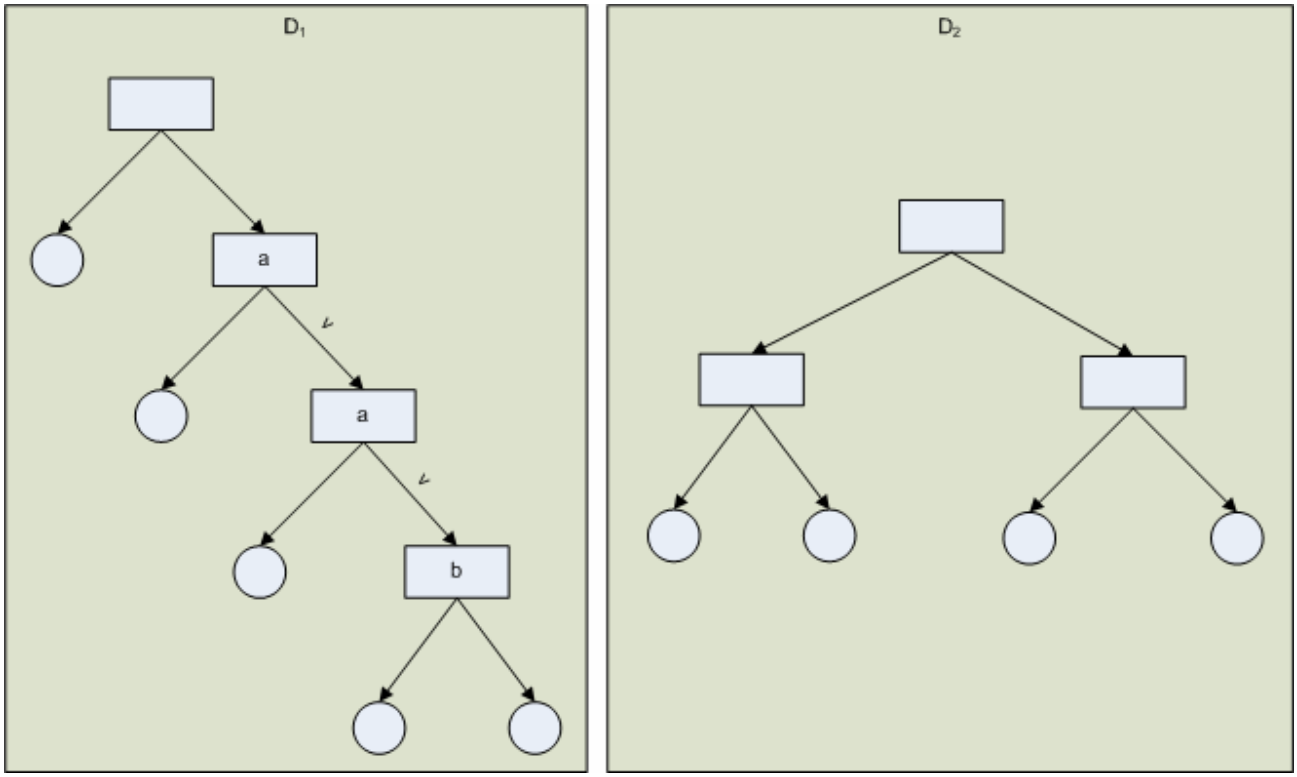


Рис. 42. Обрезка недостижимых ветвей в дереве

Мутация автоматов производится путем мутации некоторого набора состояний. Оператор мутации над деревом решений можно проводить путем замены случайного узла дерева на случайно порожденное поддерево, с учетом использованных на пути от корня до этого узла атрибутов.

Аналогично представлению в виде таблиц переходов при представлении управляющих автоматов в виде набора деревьев решений возникает несколько проблемных ситуаций: одному и тому же автомату может соответствовать несколько наборов деревьев, более того, одному и тому же состоянию автомата может соответствовать несколько различных деревьев решений. Решить данные проблемы можно опять же применением оператора переупорядочения.

Применение эвристики *MTF* в данном случае может помочь решить лишь проблему с представлением автомата в виде набора состояний опять же путем их перенумерации. Решить проблему с представлением состояний в виде различных деревьев решений могут помочь алгоритмы построения деревьев решений по известному набору гипотез.

Общая схема работы таких алгоритмов выглядит следующим образом.

1. Выбирается очередной атрибут *a*, и он помещается в корень.
2. Для всех его значений *v*:
   - Оставляем из набора гипотез только те, у которых значение атрибута *a* равно *v*.
   - Рекурсивно строим дерево в этом потомке.

Существуют различные способы выбора очередного атрибута [6]:

- Алгоритм *ID3*.
- Алгоритм *ID3* с выбором атрибута с помощью *Gain Ratio*.
- Алгоритм *ID3* с выбором атрибута с помощью *индекса Гини*.

Описание этих алгоритмов выходит за рамки данной работы, поэтому оно опущено.

2.1.2.2. Сокращенные таблицы

Основная проблема, возникающая при использовании полных таблиц рассмотренного вида – это экспоненциальный рост размерности хромосомы с увеличением числа входных воздействий объекта управления (напомним, что количество строк в таблице 2^n, где n – число входных воздействий управляющего автомата). Опыт показывает, что в реальных задачах управляющие автоматы, построенные вручную, имеют гораздо меньше переходов, чем 2^n. Причина этого, как отмечалось выше, в том, что в большинстве задач входные воздействия имеют "локальную природу" по отношению к управляющим состояниям. В каждом состоянии *значимым* является лишь определенный, небольшой поднабор воздействий, остальные же воздействия не влияют на значение управляющей функции. Именно это свойство позволяет существенно сократить размер описания состояний.

Свойство локальности входных воздействий можно использовать для сокращения описания управляющего состояния разными способами. Рассмотрим один из таких подходов, при котором количество значимых в состоянии воздействий ограничивается некоторой константой r.

К таблице, задающей сужение управляющей функции на данное состояние, в этом случае добавляется битовый вектор, описывающий множество значимых воздействий (рис. 43). В примере значимыми являются воздействия `x1` и `x3`. Управляющая функция этого состояния описывается полной таблицей переходов для автомата, имеющего только r воздействий. Так как значение r меньше общего числа воздействий, размеры таблицы существенно сокращаются.

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

Число строк таблицы в этом случае равно 2^r, однако, константа r обычно невелика. Ее выбор зависит от сложности задачи. Как показывает опыт, для большинства автоматизированных объектов достаточно $r \le 5$.

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

Для создания случайной особи выбирается r значимых воздействий, а затем таблица переходов заполняется произвольными данными.

Поскольку родительские хромосомы, представленные сокращенными таблицами, могут иметь разные множества значимых воздействий, сначала необходимо выбрать, какие из этих воздействий будут значимы для хромосом детей. Опишем алгоритм выбора значимых воздействий детей. В первую очередь, отметим, что воздействия, использующиеся в обоих родителях, будут использованы и в обоих детях. Далее воздействия, присутствующие только у одного из родителей, случайным образом перераспределяются между детьми. Работа этого алгоритма для родительских хромосом, представленных на рис. 44, проиллюстрирована на рис. 45. В данном случае воздействие `x1` будет значимым для обоих детей, так как является значимым и для обоих родителей. Воздействия же `x0` и `x3` используются только у одного родителя каждое. Так как общее число значимых воздействий равно двум, то эти воздействия будут поделены между родителями поровну.

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. 44. Родительские хромосомы, представленные сокращенными таблицами

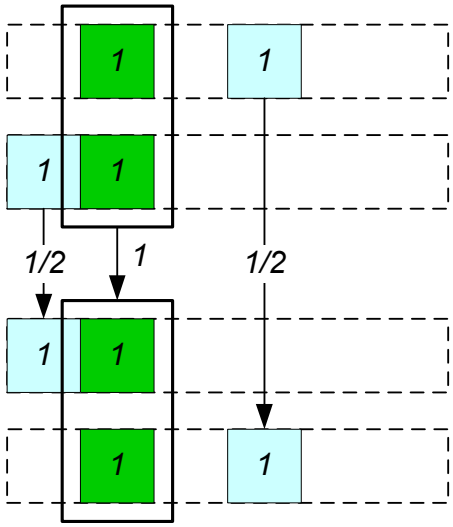


Рис. 45. Пример выбора значимых воздействий детей

После выбора значимых воздействий и точки деления родительских таблиц заполняются таблицы обоих детей. 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. Опишем предлагаемый алгоритм заполнения *i*-й строки ребенка более детально.

Выбираются те строки родительских таблиц, в которых воздействия, значимые для ребенка, имеют те же значения, что и в текущей строке одного из родителей. Причем, если воздействие значимо для обоих родителей, то его значение учитывается только для одного из родителей (в зависимости от положения строки *i* относительно точки деления). Затем данные всех выбранных строк усредняются – подсчитывается вероятность использования различных конечных состояний и выходных действий. Конечные значения для *i*-й строки ребенка вычисляются исходя из полученных вероятностей. Алгоритм заполнения проиллюстрирован на рис. 46.

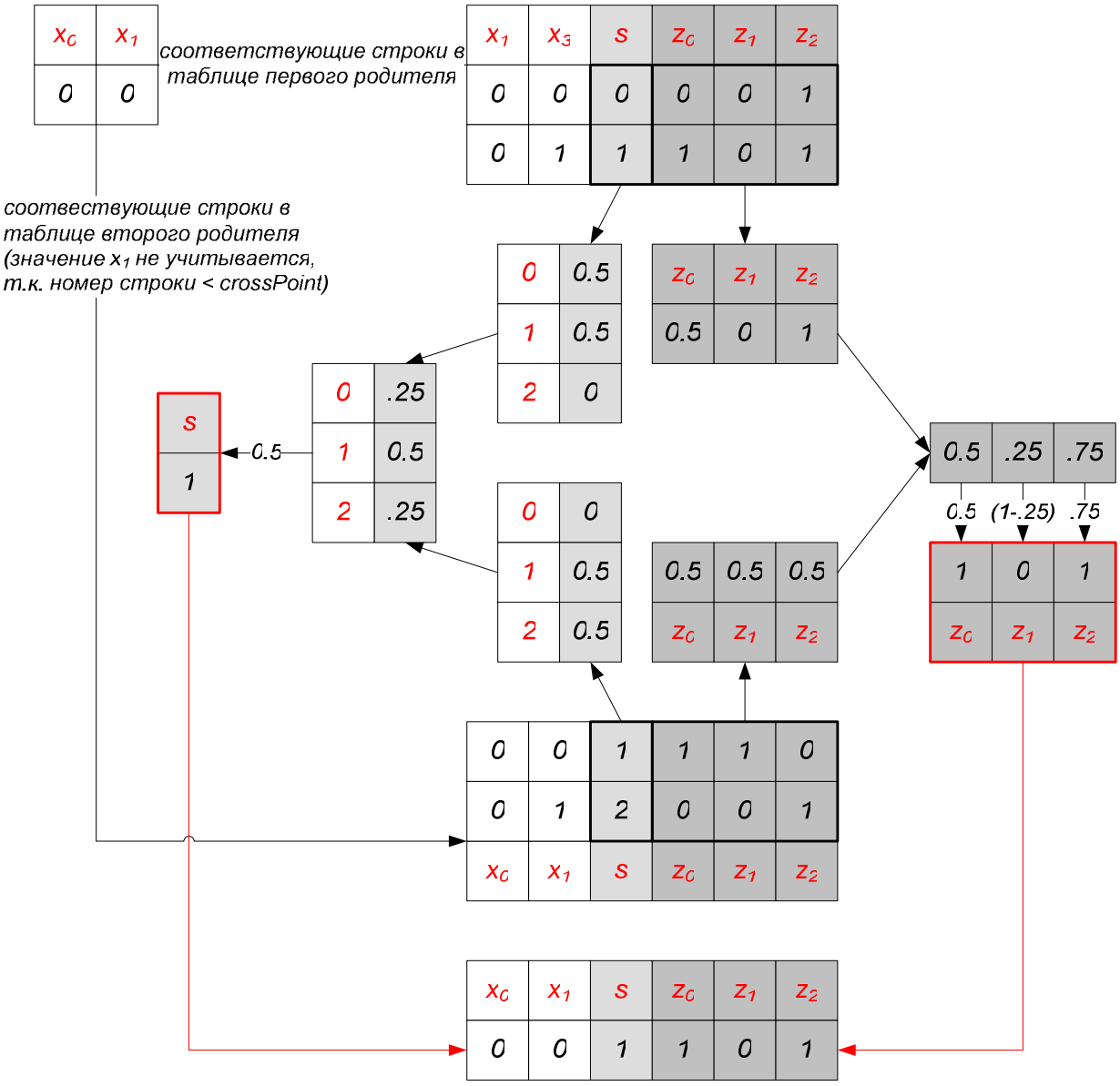


Рис. 46. Пример заполнения строки таблицы ребенка при скрещивании сокращенных таблиц

Мутация сокращенных таблиц происходит в два этапа – мутирует множество значимых воздействий и сама таблица. Сначала каждое из значимых воздействий с некоторой вероятностью заменяется другим, которое не принадлежит множеству (рис. 47). Далее 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 (рис. 48).

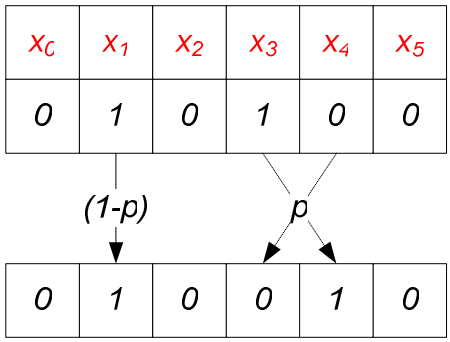


Рис. 47. Пример мутации множества значимых воздействий

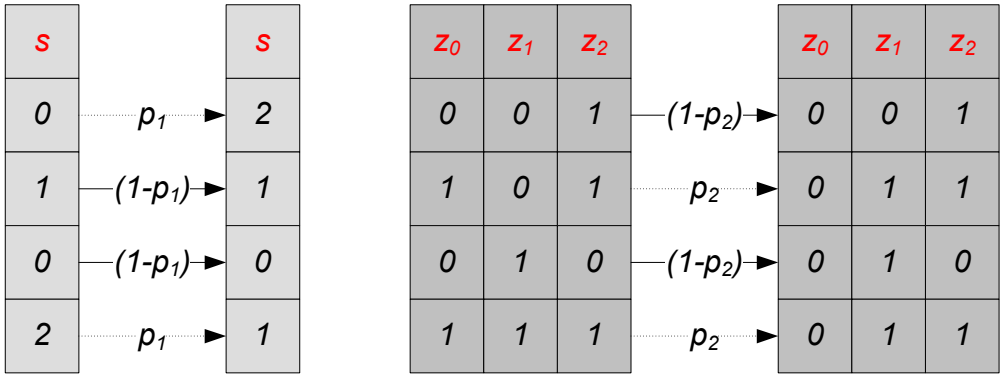


Рис. 48. Пример мутации таблиц

Предложенный алгоритм оперирует константой r, заданной для всего процесса оптимизации. Однако алгоритм легко расширяется на случай разного числа значимых воздействий у пары родителей. В этом случае необходимо добавить процедуру выбора r для каждого из детей. Такой выбор целесообразно делать случайным образом с математическим ожиданием, равным среднему арифметическому значений r пары родителей.

2.1.2.3. Графы

Естественным для управляющих автоматов представлением является представление в виде графа переходов между состояниями автомата. Таким образом, можно рассматривать автомат как граф с n состояниями и переходами при фиксированном n для популяции. Этот подход удобен тем, что все операторы работают с уже высокоуровневым представлением особи. Как следствие, их работа существенно проще и нагляднее, по сравнению с низкоуровневыми представлениями. Конкретный способ хранения графа переходов зависит от реализации и не влияет на общее поведение генетического алгоритма.

Для создания случайной особи в пустой граф добавляется необходимое количество состояний и случайные переходы между ними. Число переходов может быть константой или случайной величиной, ограниченной n^2.

Опишем оператор скрещивания для графов. Обозначим графы родителей как *P1* и *P2*, начальное состояние автомата *A* как *A.is*. Тогда для потомков *S1* и *S2* с одинаковой вероятностью будет справедливо одно из двух:

- *S1.is* = *P1.is* и *S2.is* = *P2.is*, либо
- *S1.is* = *P2.is* и *S2.is* = *P1.is*.

Затем производится попарное скрещивание всех состояний. Обозначим переход из состояния номер *i* в автомате *A* по условию *x* как *A(i, x)*. Тогда для переходов из состояния *i* в автоматах-потомках *S1* и *S2* справедливо:

- *S1(i, x) = P1(i, x)* и *S2(i, x) = P2(i, x)*, либо
- *S1(i, x) = P2(i, x)* и *S2(i, x) = P1(i, x),*

причем оба варианта равновероятны. Иллюстрация алгоритма скрещивания для состояний с двумя переходами приведена на рис. 49

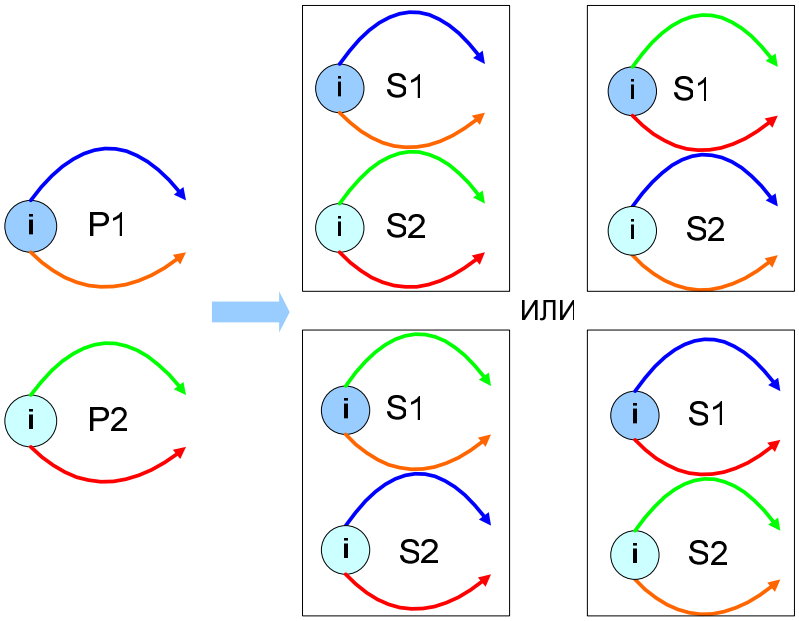


Рис. 49. Пример скрещивания графов

При мутации случайно выбирается один из равновероятных вариантов изменения:

- начального состояния;
- условия на переходе;
- действия на переходе;
- состояния, в которое ведет переход.

В случае если изменение условия на переходе приводит к конфликту, один из лишних переходов удаляется из результата.

К достоинствам описанных в этом разделе методов представления управляющих автоматов в виде хромосомы можно отнести возможность экономии памяти, за счет ограничения числа значимых входных воздействий при выборе очередного перехода. Как следствие – увеличение скорости работы генетических алгоритмов на данных представлениях.

К недостаткам можно отнести нестандартность представлений, что требует применения специфических методов скрещиваний и мутаций, а также введение дополнительных генетических операторов, таких как оператор переупорядочения.

2.2. Методы выбора структуры хромосом для отдельных задач

В настоящем разделе рассматриваются различные структуры хромосом, и анализируется их применение к решению различных задач методом генетического программирования.

2.2.1. Рассмотрение различных структур хромосом на примере некоторых задач

2.2.1.1. Битовые строки

Как отмечалось в разд. 3.1.4.1, в работе [34] для построения конечного автомата, решающего задачу об «Умном муравье», применяется генетический алгоритм, использующий битовые строки для представления хромосом.

Кодирование автомата в битовую строку осуществляется следующим образом. Сначала выполняется кодирование переходов: входному воздействию F сопоставляется единица, входному воздействию N – ноль. Далее, каждое из четырех действий кодируется следующим образом: 00 – ничего не делать, 01 – повернуть налево, 10 – повернуть направо, 11 – идти вперед.

Далее, закодированные описанным образом переходы записываются в таблицу переходов автомата. Этот процесс проиллюстрирован далее на примере некоторого автомата (рис. 50) с четырьмя состояниями, описывающего поведение муравья. Этот автомат приведен здесь для удобства чтения, так в настоящем отчете он впервые приведен на рис. 15.

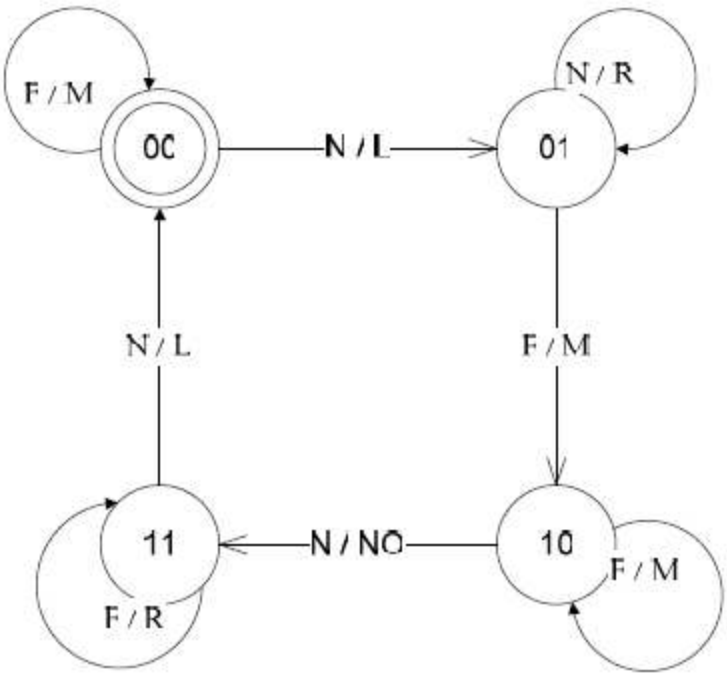


Рис. 50. Автомат, описывающий поведение некоторого муравья

В табл. 7 является таблицей переходов рассматриваемого автомата (она впервые приведена в табл. 4).

Таблица 7. Таблица переходов рассматриваемого автомата

Старое состояние	Входное воздействие	Новое состояние	Выполняемое действие
00	0	01	01
00	1	00	11
01	0	01	10
01	1	10	11
10	0	11	00
10	1	10	11
11	0	00	01
11	1	11	10

После этого производится запись построенной таблицы в виде строки. Для этого в начало строки записывается начальное состояние, после которого записывается содержимое двух последних столбцов таблицы в некотором фиксированном порядке (например, сверху вниз, а внутри строк – слева направо). Например, для рассматриваемого автомата кодирующая его строка имеет следующий вид: `0001010011011010111100101100011110`.

Отметим, что для однозначного кодирования и декодирования битовой строки необходимо зафиксировать число состояний в автомате. В рассматриваемом примере автомат содержит четыре состояния, а в работе [34] рассматривались автоматы с 32 состояниями. Таким образом, для кодирования номера состояния требовалось пять битов, таблица переходов содержала 64 строки, а длина ее битового представления составляла 5+64*(5+2)=453 бита. Отметим, что при таком способе кодирования любая битовая строка соответствует некоторому автомату. При этом размер пространства поиска составляет 2^{453}.

Для построения автоматов, решающих задачу об «Умном муравье», в работе [34] использовался генетический алгоритм, хромосомы в котором кодируются с помощью битовых строк. Ниже приведено его описание.

Для вычисления *функции приспособленности* выполняется моделирование прохождения игрового поля муравьем, поведение которого задается автоматом, описываемым битовой строкой. При этом *количество еды, съеденной этим муравьем, и считается значением функции приспособленности*.

Размер поколения при всех запусках алгоритма равен 65536. Очередное поколение особей строилось следующим образом. Сначала особи текущего поколения упорядочивались в порядке убывания их значения функции приспособленности.

После этого фиксированная доля (от одного до десяти процентов при разных запусках алгоритма) лучших особей выбиралась для дальнейшей обработки, а остальные особи отбрасывались. После этого выбиралось 65536 случайных пар строк из выбранных для скрещивания.

Из каждой выбранной пары с помощью *случайного кроссовера* строилась одна битовая строка. Кроссовер производился следующим образом: каждый бит одной из строк с вероятностью от 0.5% до 1% заменялся на соответствующий бит второй строки.

Далее, все построенные хромосомы подвергались *мутации*. Мутация состояла в том, что каждый бит строки с вероятностью от 0.1% до 1% заменялся на противоположный.

С помощью описанного алгоритма в [34] за 200 поколений построен автомат, содержащий 13 состояний, который решает задачу об «Умном муравье» ровно за 200 ходов. Его граф переходов приведен на рис. 51 (первый раз этот граф был приведен на рис. 13).

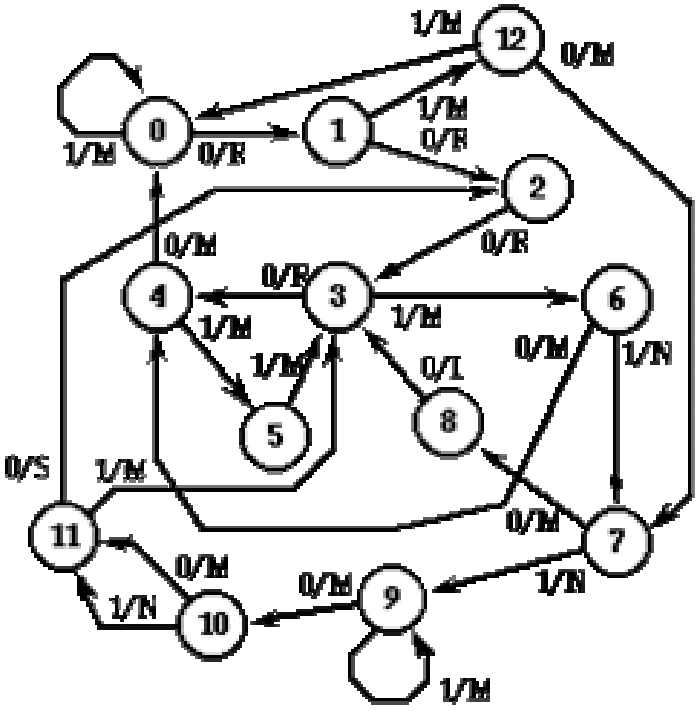


Рис. 51. Автомат из 13 состояний, решающий задачу об «Умном муравье»

На этом рисунке используются следующие обозначения:

- для входных воздействий:
  - 0 – перед муравьем нет еды;
  - 1 – перед муравьем есть еда.
- для выходных воздействий:
  - N – ничего не делать;
  - L – повернуть налево;
  - R – повернуть направо;
  - M – идти вперед.

То, что указанный автомат использует все 200 ходов для решения задачи об «Умном муравье» связано с используемой функцией приспособленности, которая не дает преимуществ автоматам, требующим меньшего количества ходов.

Как уже отмечалось в разд. 3.1.4.1, автомат на рис. 51 изображен некорректно – из состояния 11 изображен переход, помеченный входным воздействием 0, с “непонятным” действием S. Однако, если это действие заменить на N, то автомат корректно решает задачу (http://is.ifmo.ru/works/_ant.pdf).

Приведем для сравнения автомат из 17 состояний, оказавшийся лучшим в первом поколении, состоящем из случайно сгенерированных автоматов (какие у него свойства? решает ли он задачу?). Его граф переходов приведен на рис. 52.

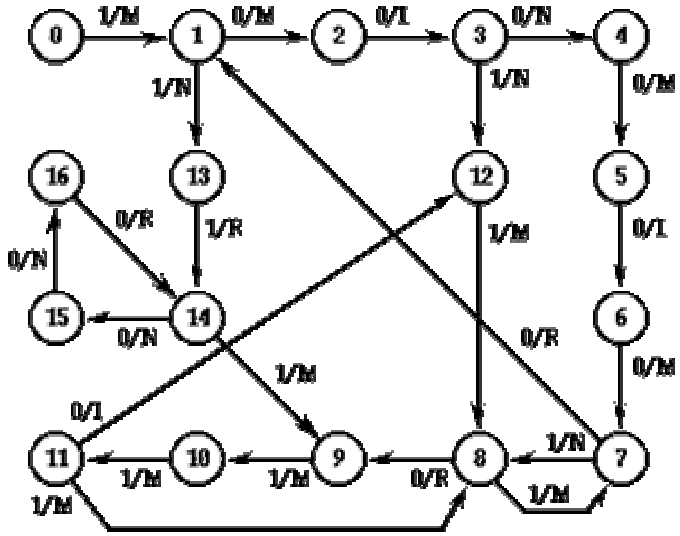


Рис. 52. Автомат из 17 состояний – лучший в первом поколении

Приведем также результаты исследования пространства поиска методом Монте-Карло, проведенного в работе [34]. Для этого авторами [34] было случайно сгенерировано более 10^9 битовых строк, кодирующих особи. Далее, каждая строка была преобразована в конечный автомат, задающий поведение муравья, и для автомата была вычислена функция приспособленности – количество съеденной еды.

Результаты этого вычислительного эксперимента представлены в табл. 8.

Таблица 8. Результаты вычислительного эксперимента

Значение функции	Количество автоматов	Значение функции	Количество автоматов	Значение функции	Количество автоматов
0	557258091	30	1087674	60	842
1	283207491	31	1047454	61	646
2	132410979	32	9751593	62	641
3	63272402	33	1971378	63	690
4	31819298	34	1104609	64	4066
5	18120801	35	807494	65	524
6	12985610	36	680257	66	230
7	10063926	37	614186	67	202
8	9603340	38	624746	68	156
9	8472800	39	509548	69	183
10	97907623	40	522627	70	872
11	20571404	41	500301	71	157
12	9917884	42	12110275	72	117
13	6084789	43	369162	73	103
14	4700955	44	170037	74	756
15	3694985	45	110622	75	255
16	3392771	46	117108	76	51
17	3091910	47	638272	77	45
18	3164998	48	92670	78	354
19	3665297	49	34361	79	188
20	6877796	50	24808	80	64
21	8973385	51	23300	81	10
22	2800666	52	105961	82	0
23	1831525	53	24029	83	0
24	1672638	54	12380	84	0
25	1399715	55	9505	85	0
26	1466050	56	9684	86	0
27	4830484	57	10218	87	0
28	1691564	58	125296	88	0
29	1339940	59	5488	89	0

Результаты этого вычислительного эксперимента представлены также и в виде гистограммы (рис. 53).

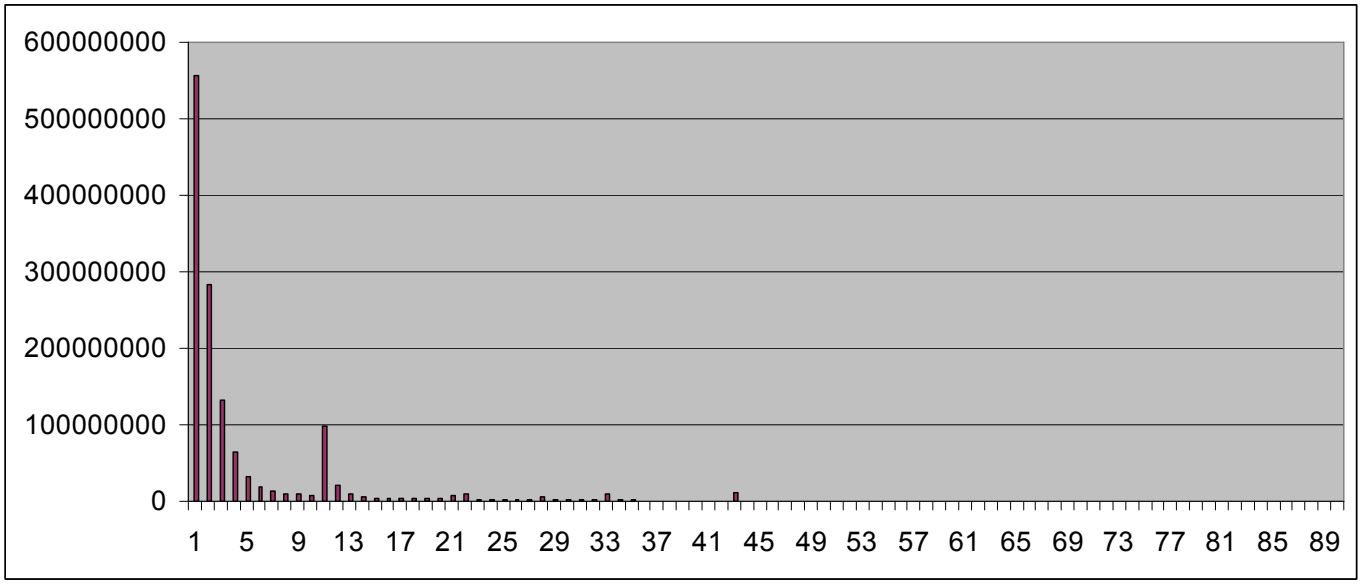


Рис. 53. Распределение значения функции приспособленности у случайно сгенерированных автоматов

По оси абсцисс на рис. 53 отложено значение функции приспособленности (количество съеденной еды), а по оси ординат – количество автоматов, которые имеют соответствующее значение функции приспособленности. Рассмотрение этой гистограммы позволяет выяснить некоторые весьма интересные свойства пространства поиска.

Так, например, примерно 41% всех сгенерированных случайно автоматов имеет значение функции приспособленности, равное нулю, а 90% всех автоматов не может съесть больше десяти единиц еды вследствие того, что они не могут правильно выполнить первый поворот направо. Далее, из структуры расположения еды понятно, что значение функции приспособленности, большее 32, означает умение выполнить левый поворот, а значение, большее 42, – умение выполнить левый поворот в том случае, если угловой квадрат не содержит еду.

Примерно один из ста тысяч автоматов имеет значение функции приспособленности, большее 58. Такое значение достигается, если автомат способен пройти две клетки подряд, которые не содержат еду. *Наилучший из сгенерированных автоматов имеет значение функции приспособленности, равное 81. Таких автоматов было найдено 10, что, скорее всего, означает, что их очень мало.* Количество автоматов, имеющих значение функции приспособленности 82 и выше, настолько мало, что их не удалось обнаружить при исследовании пространства поиска методом Монте-Карло. Отметим, что при построении автоматов с помощью генетического алгоритма, как правило, автомат с таким же значением функции приспособленности строился за 100 поколений по 65536 автоматов – после обработки примерно одного миллиона особей.

Таким образом, можно сделать вывод о том, что задача об «Умном муравье» является достаточно сложной. При этом основная сложность состоит не в том, что сложно съесть все 89 единиц еды, а в том, что это сложно сделать за 200 ходов. Обычно более трети времени автоматы тратили на прохождении последних восьми квадратов, содержащих еду.

Отметим также, что аналогичный анализ пространства поиска был выполнен в работе [34] проведен и для искусственных нейронных сетей, задающих поведение муравья. Его результаты аналогичны результатам для конечных автоматов.

2.2.1.2. Битовые строки и приведение автоматов к каноническому виду

В работе [19] приводится генетический алгоритм, являющийся более совершенным вариантом алгоритма, описанного в работе [34]. Основное его отличие состоит в том, что перед преобразованием в битовые строки автоматы преобразуются к некоторому каноническому виду.

В работе [34] размер пространства поиска составлял 2^{453} автоматов. Это пространство может быть разбито на классы эквивалентности, внутри каждого из которых находятся строки, по-разному описывающие один и тот же автомат.

Например, автоматы, таблицы переходов которых приведены в табл. 9 и табл. 10, изоморфны и задают одинаковое поведение. Однако их представление в виде битовых строк различается. Таким образом, в одном поколении могут присутствовать одинаковые автоматы, а их представление в виде битовых строк при этом может существенно отличаться.

Таблица 9. Таблица переходов первого из изоморфных автоматов

Старое состояние	Входное воздействие	Новое состояние	Выполняемое действие
00	0	10	11
00	1	01	11
01	0	10	11
01	1	10	11
10	0	01	11
10	1	01	11

Таблица 10. Таблица переходов второго из изоморфных автоматов

Старое состояние	Входное воздействие	Новое состояние	Выполняемое действие
00	0	01	11
00	1	10	11
01	0	10	11
01	1	10	11
10	0	01	11
10	1	01	11

На основании приведенных выше неформальных рассуждений и теоремы о шаблонах, в работе [19] сделан вывод о том, что процесс сходимости алгоритма замедляется, в случае если в одной популяции могут находиться автоматы, обладающие одинаковым поведением, но различными представлениями в виде битовых строк.

Это происходит в силу того, что разные представления одного и того же автомата не обязательно соответствуют одному и тому же шаблону. Поэтому имеет место «соревнование» между различными «хорошими» шаблонами, которое ведет к замедлению сходимости алгоритма.

Для решения этой проблемы в работе [19] предлагается перед кодированием приводить автоматы к некоторому каноническому виду. Это означает, что различные представления одного и того же автомата (отличающиеся, например, нумерацией состояний) будут представлены одной и той же битовой строкой. Таким образом, «соревнование» между шаблонами будет иметь место в меньшей степени, что должно привести к ускорению сходимости алгоритма.

В работе [19] предлагается использовать два алгоритма приведения автомата к каноническому виду: *SFS* (*Standardizing transitions to the Future or next States according to a mathematical function*) и *MTF* (*Move To Front*). Ниже приведено описание каждого из этих алгоритмов.

Напомним, что структура хромосомы в рассматриваемом алгоритме такая же, как и в алгоритме из [34]. Таким образом, хромосома является битовой строкой длиной 453 бита, которая получена из таблицы переходов автомата.

Алгоритм SFS. В процессе работы алгоритма *SFS* некоторые состояния могут быть переименованы. Поэтому для начала опишем, как переименовываются состояния. Так как размер хромосомы ограничен, то для изменения номера состояния необходимо поменять местами его описание с описанием другого состояния. Таким образом, переименование состояния состоит в обмене его описания местами с описанием некоторого другого состояния и изменении всех его упоминаний в таблице переходов.

Алгоритм *SFS* напоминает обход графа переходов автомата в ширину из начального состояния, однако при этом номера состояниям присваиваются не в порядке их посещения, а соответствии с некоторой функцией двух аргументов.

Рассматриваемый алгоритм состоит из двух шагов. На первом из них начальному состоянию автомата присваивается номер ноль, и оно помещается в очередь.

На втором шаге алгоритма до тех пор, пока очередь не пуста, из нее выбирается очередное состояние `i` и происходит его обработка. При обработке рассматриваются переходы из этого состояния. Если обрабатываемый переход (соответствующий значению `j` входной переменной) ведет в еще не просмотренное состояние, то ему присваивается номер, равный значению функции `desired_state(i,j)`, и это состояние добавляется в очередь. Обозначим переменной `n` число состояний в автомате.

Тогда указанная функция определяется следующим образом: `desired_state(i, j) =`

- `2i + j + 1`, если `i ≤ n/2 - 2`;
- `n – 1`, если `i = n/2 - 1`;
- `2(n – 2 – i) + j + 1`, если `n/2 – 1 < i < n – 1`;
- `0`, если `i = n – 1`.

Отметим, что эта функция построена специально для автоматов с двумя переходами из каждого состояния, и ее обобщение на другое число переходов может быть весьма затруднительным.

Приведем пример работы алгоритма *SFS*. Рассмотрим автомат, задаваемый функцией переходов, показанной в табл. 11. Начальным состоянием этого автомата является состояние 13.

Таблица 11. Таблица переходов автомата для иллюстрации работы алгоритма *SFS*

Старое состояние	Входное воздействие	Новое состояние	Выполняемое действие
13	0	5	00
13	1	9	10
5	0	13	10
5	1	5	10
9	0	5	01
9	1	24	10
24	0	13	11
24	1	24	10

После первого шага алгоритма *SFS* таблица переходов автомата примет следующий вид (табл. 12). Кроме этого, состояние 0 будет находиться в очереди.

Таблица 12. Таблица переходов автомата после первого шага алгоритма *SFS*

Старое состояние	Входное воздействие	Новое состояние	Выполняемое действие
0	0	5	00
0	1	9	10
5	0	*0*	10
5	1	5	10
9	0	5	01
9	1	24	10
24	0	*0*	11
24	1	24	10

Далее, в процессе работы алгоритма из очереди будет извлечено состояние 0, и будут проанализированы переходы из него. В результате этого состояние 5 будет переименовано в состояние 1, а состояние 9 – в состояние 2, а в очереди будут находиться состояние 1 и 2. Далее при обработке состояния 2 состояние номер 24 будет переименовано в состояние 6. Таким образом, таблица переходов автомата примет следующий вид (табл. 13).

Таблица 13. Таблица переходов автомата после первого шага алгоритма *SFS*

Старое состояние	Входное воздействие	Новое состояние	Выполняемое действие
0	0	1	00
0	1	2	10
1	0	0	10
1	1	1	10
2	0	1	01
2	1	6	10
6	0	0	11
6	1	6	10

Отметим, что любой автомат, отличающийся от приведенного в табл. 11 только нумерацией состояний, с помощью рассмотренного алгоритма будет приведен к виду, показанному в табл. 13.

Алгоритм MTF. Второй алгоритм, используемый в работе [19], называется *MTF* (*Move the reachable states of a finite state machine To Front*). Как было сказано ранее, код любой особи содержит 32 состояния, однако не все эти состояния обязательно достижимы из начального состояния. Основная задача алгоритма *MTF* – перенумеровать состояния так, чтобы достижимые из начального состояния имели номера от `0` до `n-1`. Приведем описание алгоритма *MTF*.

Также как и алгоритм *SFS*, алгоритм *MTF* присваивает начальному состоянию автомата номер ноль. Далее, производится обход автомата в глубину и каждому обрабатываемому состоянию присваивается наименьший, еще не присвоенный ни одному состоянию, номер. Таким образом,

применение алгоритма *MTF* к автомату, приведенному в табл. 11, приводит к автомату, функция переходов которого приведена в табл. 14.

Таблица 14. Таблица переходов автомата после применения алгоритма *MTF*

Старое состояние	Входное воздействие	Новое состояние	Выполняемое действие
0	0	1	00
0	1	2	10
1	0	0	10
1	1	1	10
2	0	1	01
2	1	3	10
3	0	0	11
3	1	3	10

По результатам вычислительных экспериментов, описанных в работе [19], можно сделать вывод о том, что реорганизация информации внутри хромосомы позволяет существенно ускорить процесс сходимости генетического алгоритма к достаточно хорошему решению. Это связано с тем, что преобразование *SFS* улучшает формирование шаблонов за счет приведения переходов автомата к стандартному виду, а преобразование *MTF* уменьшает длину «полезных» шаблонов.

Кратко приведем результаты построения автоматов из работы [19]. Наилучшему из построенных автоматов с двенадцатью состояниями требуется 181 ход на решение задачи об «Умном муравье». Граф переходов этого автомата приведен на рис. 51. Здесь выходные воздействия имеют следующий смысл: 0 – ничего не делать, 1 – поворот налево, 2 – поворот направо, 3 – идти вперед.

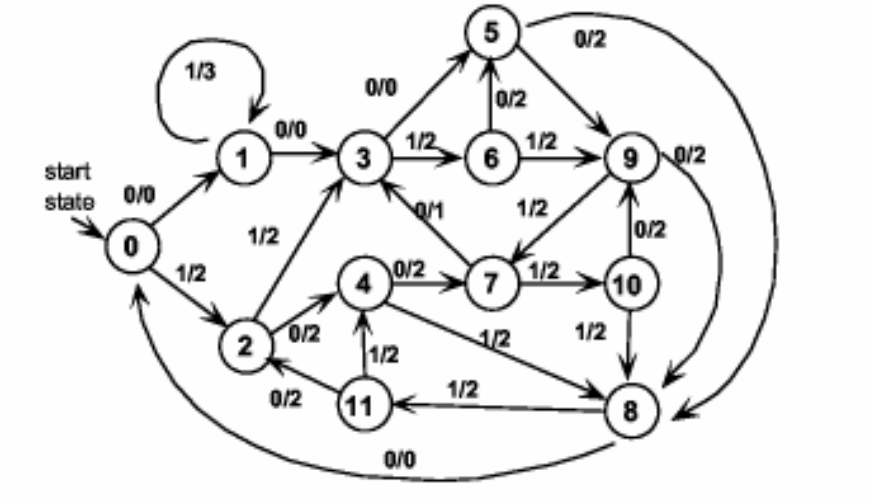


Рис. 54. Автомат из 12 состояний, решающий задачу об «Умном муравье»

Отметим, что в процессе прохождения игрового поля в задаче об «Умном муравье» указанным автоматом из 12 состояний переходы из состояний один и девять по входному воздействию «1» никогда не используются. Если их исключить, то получим автомат с 11 состояниями.

В работе [19] построен также автомат из 11 состояний, которому требуется 188 ходов на решение задачи об «Умном муравье». Его граф переходов приведен на рис. 55.

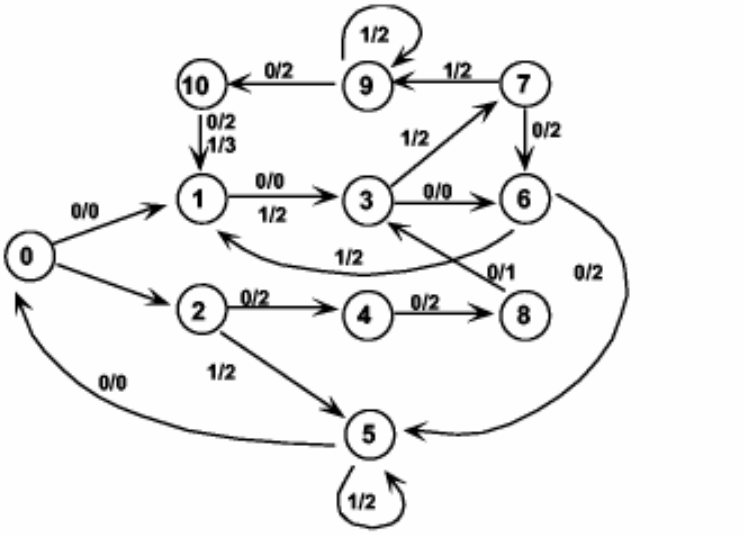


Рис. 55. Автомат из 11 состояний, решающий задачу об «Умном муравье»

Кроме этого автомата, в работе [19] представлен автомат, требующий меньшего времени и содержащий меньшее число состояний, чем показанный на рис. 55. Он содержит десять состояний и требует 185 ходов на решение задачи об «Умном муравье». Его граф переходов приведен на рис. 56.

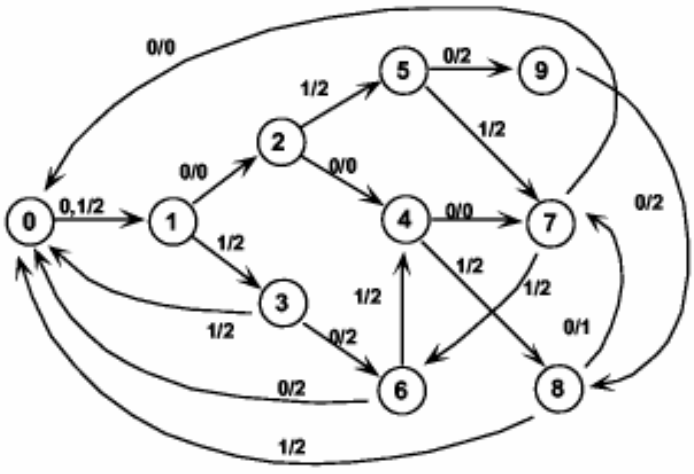


Рис. 56. Автомат из 10 состояний, решающий задачу об «Умном муравье»

В работе [19] также построен автомат, который решает задачу об «Умном муравье» и содержит девять состояний. Его граф переходов приведен на рис. 57.

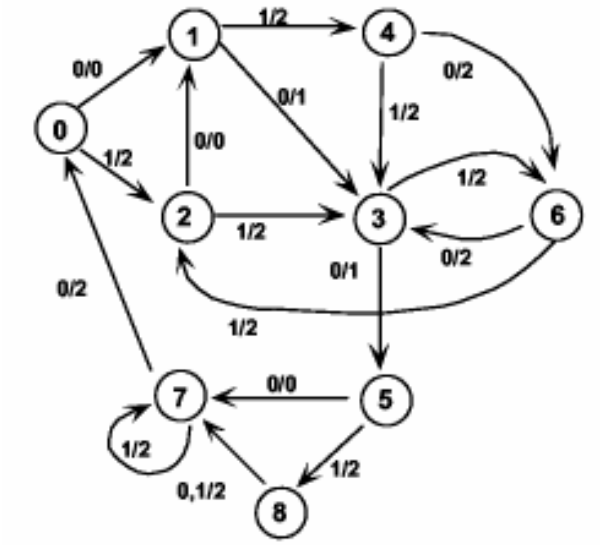


Рис. 57. Автомат из 9 состояний, решающий задачу об «Умном муравье»

Наименьший построенный в работе [19] автомат, решающий задачу об «Умном муравье», содержит восемь состояний. Его граф переходов показан на рис. 58.

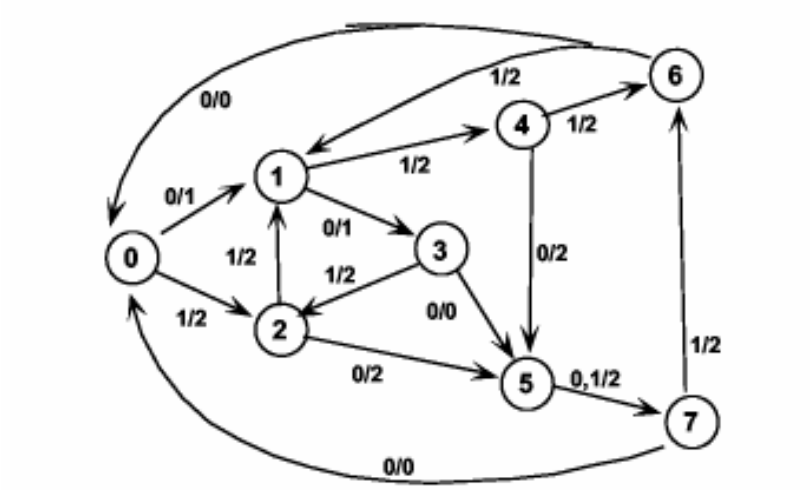


Рис. 58. Автомат из 8 состояний, решающий задачу об «Умном муравье»

Этому автомату требуется 193 хода на решение задачи об «Умном муравье». Отметим, что изменением некоторых переходов этого автомата можно получить другие автоматы, которые решают задачу об «Умном муравье». Им, однако, требуется больше времени для этого. Например, если направить переход из состояния 5 по значению входной переменной 1 в состояние 5, получится автомат, которому требуется 197 ходов.

Кроме указанного автомата, в работе [19] приводится еще один автомат (рис. 59), который содержит восемь состояний и решает задачу об «Умном муравье». Он интересен тем, что муравей, управляемый им не совершает поворотов налево.

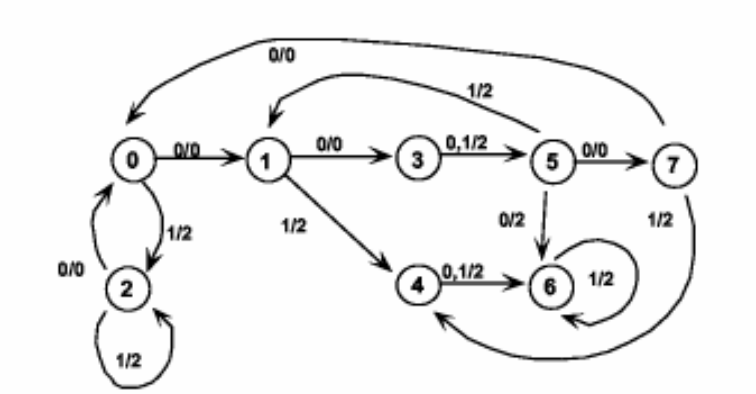


Рис. 59. Автомат из 8 состояний, решающий задачу об «Умном муравье» без поворотов налево

На основании приведенных результатов можно сделать вывод о том, что реорганизация данных внутри хромосомы с помощью описанных алгоритмов позволяет строить автоматы с меньшим числом состояний по сравнению с генетическим алгоритмом, приведенным в работе [34].

2.2.1.3. Генетическое программирование и объединение в модули

В работе [12] для решения задачи об «Умном муравье» используется генетическое программирование. При этом автомат хранится в виде графа переходов. Главной особенностью этой работы является то, что помимо традиционно используемых в генетическом программировании операций мутации и скрещивания (рекомбинации, кроссовера) к хромосомам также применяется так называемые операции «сжатие» (*compress*) и «расширение» (*expand*).

Применение операции «сжатие» соответствует объединению некоторой части особи в модуль – некоторую сущность, которая не может быть «разрушена» генетическими операциями мутации и скрещивания. Операция «расширение» является обратной к операции «сжатие» – после ее применения модуль разрушается, и его части снова могут быть изменены с помощью генетических операций.

Поскольку общего метода, позволяющего определить, какую именно часть особи следует «сжать», не существует, выбор этой части осуществляется случайно. Таким образом, если случайно выбранная часть особи оказывается «очень ценной» с точки зрения функции приспособленности особи, эта часть переходит к потомкам. В результате этого может ускориться процесс нахождения оптимального решения задачи.

Реализация операций «сжатие» и «расширение» существенно зависит от способа представления особи. Единственным важным требованием, которое к ним предъявляется, является требование *прозрачности* относительно функции приспособленности – после применения любой последовательности этих операций к особи ее значение функции приспособленности не должно измениться. Таким образом, можно говорить, что эти операции приводят только к синтаксическим, но не к семантическим изменениям. Таким образом, применение операций «сжатие» и «расширение» влияет только на то, каковы будут результаты применения генетических операций к особи.

В работе [12] описываются конкретные способы реализации этих операций применительно к хромосомам, представляемым в виде графов переходов – операции «замораживание» и «размораживание». После применения операции «замораживание» некоторая часть особи фиксируется и не может быть в дальнейшем изменена генетическими операциями мутации и скрещивания. Операция «размораживание» производит обратный эффект – после ее применения часть особи снова может быть изменена генетическими операциями.

Для решения задачи об «Умном муравье» в работе [12] применяется генетическое программирование. При этом используется представление хромосом в виде графов переходов с операциями «замораживание» и «размораживание».

При выполнении операции «замораживание» случайно выбирались: одно состояние и не более пяти переходов. После этого выбранные переходы и состояние не модифицировались генетическими операторами. Для операции «размораживания» также случайно выбиралось одно состояние и не более пяти переходов. При построении потомка с вероятностью 10% происходило «замораживание», а с вероятностью 20% – «размораживание». При этом не более 75% состояний и 75% переходов могли быть «замороженными» одновременно у одной особи.

Приведем описание используемой операции мутации. При выполнении мутации с равной вероятностью происходит мутация состояния или перехода. При выборе мутации состояния происходит либо добавление состояния, либо удаление состояния. Вероятность того, что будет удалено состояние, составляет $p = \frac{numStates}{statesMax}$, где `numStates` – число состояний в автомате, а `statesMax` – максимально возможное число состояний. Таким образом, если число состояний автомата превышает половину максимально возможного, то более вероятно удаление состояния, иначе – более вероятно добавление состояния. Следовательно, сглаживается тенденция к генерации автоматов с все большим и большим числом состояний и распределение числа состояний автоматов внутри поколения становится более равномерным.

Число мутаций, происходящих в особи, определяется формулой $c = 1 + \mathrm{round}(|\,N(0,T) \cdot size\,|)$, где `N(0,T)` – случайная величина с нормальным распределением с центром в нуле и дисперсией, пропорциональной значению функции приспособленности особи, а `size` – число состояний в автомате, описывающем поведение особи.

Для перехода к следующему поколению использовался соревновательный метод. Если две особи обладали одинаковым значением функции приспособленности, то «победитель» из них выбирался случайно.

С помощью генетического программирования в работе [12] был построен автомат, решающий задачу об «Умном муравье», который содержит 11 состояний. Его граф переходов показан на рис. 60 (впервые в настоящей работе он был приведен на рис. 14).

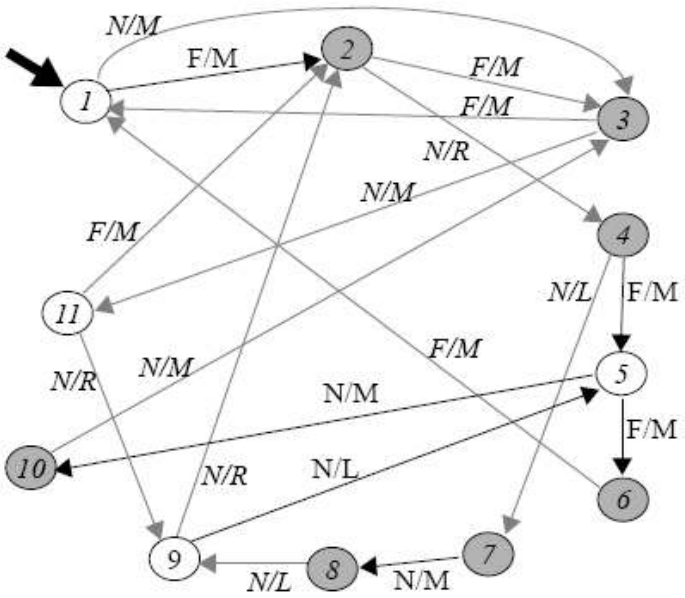


Рис. 60. Автомат из 11 состояний, решающий задачу об «Умном муравье»

На рис. 60 «замороженные» состояния и переходы показаны темно-серым цветом. Такими оказываются семь состояний и пять переходов. Этому автомату требуется 193 хода на решение задачи об «Умном муравье». Изучим более подробно его структуру. Отметим, что состояния с первого по шестое и переходы между ними хорошо защищены от мутаций «замораживанием». Также отметим, что с помощью этих состояний муравей проходит непрерывные отрезки с едой – эта часть автомата является очень важной для обеспечения оптимального поведения. Кроме того, можно проверить, что мутация какого-либо из этих переходов приводит к получению далекого от оптимального автомата. Таким образом, понятно, что с помощью «замораживания» выделяются наиболее важные с точки зрения обеспечения оптимального поведения состояния и переходы.

Автомат на рис. 11 изображен некорректно — из девятого состояния исходят две дуги, помеченные одинаковым входным воздействием. Однако, если пометку `N/L` на переходе 9–5 заменить на пометку `F/M`, то автомат становится корректным (http://is.ifmo.ru/works/_ant.pdf/).

Приведем более подробно результаты вычислительных экспериментов из работы [12]. Было проведено восемь запусков алгоритма, при каждом из которых размер поколения был равен 300-ам особям. Для того чтобы определить эффективность применения операций «сжатие» и «расширение» четыре запуска использовали эти операции в процессе генетического программирования автоматов, а другие четыре – не использовали. В табл. 15 показаны сводные результаты этих запусков алгоритма.

Таблица 15. Результаты построения автоматов со «сжатием» и без «сжатия»

Запуск алгоритма	Количество просмотренных автоматов (количество поколений)		Ускорение
	Без использования операции «сжатия»	С использованием операции «сжатия»	
1	187950 (1251)	63000 (418)	2,99
2	269850 (1797)	152100 (1012)	1,77
3	331200 (2206)	156600 (1042)	2,12
4	734850 (4897)	607950 (4051)	1,20

Для каждого из запусков алгоритма в табл. 15 приведено число автоматов, сгенерированных алгоритмом до получения автомата, который описывает поведение муравья, съедающего все 89 единиц еды за 200 ходов. Число поколений для каждого запуска алгоритма указано в скобках. Запуски алгоритма отсортированы по возрастанию скорости таким образом, что самый быстрый запуск алгоритма, использующего «сжатие», можно напрямую сравнивать с самым быстрым запуском алгоритма, который не использует «сжатие». Значения в столбце «Ускорение» были получены делением количества просмотренных автоматов для алгоритма, не использующего «сжатие», на количество просмотренных автоматов для алгоритма, использующего «сжатие».

Кроме указанных в таблице запусков, было проведено еще два (один использовал «сжатие», другой – не использовал), которые не нашли решения за 5000 поколений. Результаты этих двух запусков не включены в табл. 15.

Из изложенного можно сделать следующие выводы.

Во-первых, отметим, что построение оптимальных автоматов потребовало существенно меньше времени, чем в работе [34], – ускорение составило от 4.5 до 53 раз. Это, скорее всего, обусловлено переходом от применения генетического алгоритма с представлением хромосом в виде битовых строк к генетическому программированию с хранением хромосом в высокоуровневом представлении – в виде графов переходов конечных автоматов, задающих поведение муравья. При использовании генетического программирования удается избежать сложной функции преобразования генотипа (битовой строки) в фенотип (конечный автомат).

Во-вторых, отметим, что применение «сжатия» ускоряет процесс сходимости алгоритма в каждом из четырех случаев. При этом коэффициент ускорения принимает значения от 1,20 до 2,99. Объяснение этого эффекта заключается в том, что с помощью «сжатия» находятся и защищаются от нежелательных изменений компоненты, которые обеспечивают высокое значение функции приспособленности особи.

Однако, в части случаев (например, четвертый запуск алгоритма в табл. 15) алгоритм, использующий «сжатие», находит решение медленнее, чем три самых быстрых запуска алгоритма, которые не используют «сжатие». Это объясняется тем, что на ранних этапах эволюционного поиска решения «сжатием» могут случайно быть защищены от изменений те части автомата, которые позволяют ему достичь достаточно большого значения функции приспособленности, но без изменения которых не может быть найдено оптимальное решение – автомат, задающий поведение такого муравья, который съедает все 89 единиц еды за 200 ходов.

Из изложенного следует, что применение генетического программирования и операций «сжатие» и «расширение» позволяет строить автоматы с меньшим числом состояний для задачи об «Умном муравье» по сравнению с генетическим алгоритмом, использующим представление особей в виде битовых строк.

2.2.1.4. Строки в задаче о «флибах»

В работе [1] для решения задачи о «флибах» применяется генетический алгоритм, использующий кодирование хромосом в виде строк. Проиллюстрируем на примере способ кодирования автомата в виде строки. Рассмотрим автомат, граф переходов которого показан на рис. 61.

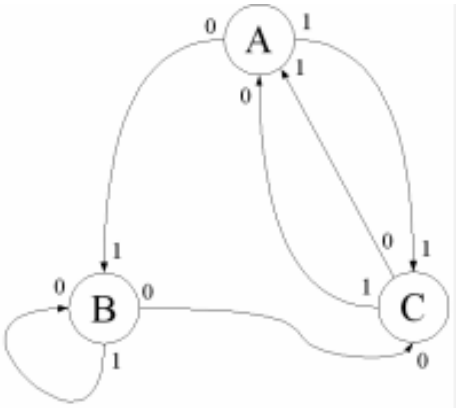


Рис. 61. Автомат из тремя состояниями для задачи о «флибах»

Табл. 16 – это таблица переходов и выходов этого автомата.

Таблица 16. Таблица переходов автомата для задачи о «флибах»

Старое состояние	0	Новое состояние	1	Новое состояние
A	1	B	1	C
B	0	C	0	B
C	1	A	0	A

В этой таблице для каждого состояния и входного символа указаны два элемента: выходной символ (второй и четвертый столбцы) и новое состояние (третий и пятый столбцы). Для представления таблицы переходов в виде строки столбцы со второго по пятый записываются в построчно. Например, для указанного автомата соответствующей строкой будет: `1B1C0C0B1A0A`.

Отметим, что число символов в такой строке в четыре раза превышает число состояний кодируемого ей автомата. Далее, если пронумеровать состояния автомата неотрицательными целыми числами, начиная с нуля, то если «флиб» находится в состоянии с номером `s`, а текущее входное воздействие равно `i`, то состояние, в которое перейдет «флиб» содержится в символе номер `4s+2i+1`, а значение выходного воздействия – в символе с номеров `4s+2i`.

Как показывают вычислительные эксперименты, проведенные в работе [4], описанная структура хромосомы позволяет строить «флибы», достаточно хорошо предсказывающие изменение среды. Однако точность предсказания все равно остается весьма далекой от 100%.

Например, при задающей среду битовой маске `1111010010111101001`, двадцати состояниях «флиба» и 400 поколениях наилучшая достигнутая точность предсказания равна 88%, а средняя – 78.3%. При битовой маске среды `10101111011000111101111110011001`, тридцати состояниях «флиба» и 1600 поколениях наилучшая достигнутая точность составила 87%, а средняя – 75.86%. Отметим, что существуют структуры хромосом, позволяющие при том же числе состояний и поколений строить «флибы», гораздо лучше предсказывающие среду.

2.2.1.5. Генетическое программирование в задаче о «флибах»

В работе [4] предлагается способ структурированного представления хромосом в задаче о «флибах», который не использует представление в виде битовых строк.

Этот способ основывается на кодировании «флиба» с помощью трех классов: `Flib`, `State` и `Branch`. Главным классом, представляющим «флиб», является класс `Flib`. Классы `State` и `Branch` реализуют его состояния и переходы, соответственно. Во всех этих классах имеется метод `Clone`, предназначенный для создания копии соответствующего объекта.

Опишем внутреннюю структуру указанных классов. Массив `_states` в классе `Flib` содержит состояния «флиба». Поле `_curStateIndex` класса `Flib` содержит номер текущего состояния «флиба». Число правильно предсказанных символов хранится в поле `_guessCount`. Метод `Step` переводит «флиб» в новое состояние и обновляет значение числа правильно предсказанных символов. Метод `Nulling` переводит «флиб» в начальное состояние.

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

Поля `_stateIndex` и `_output` класса `Branch` содержат номер состояния, в которое ведет переход, и значение выходной переменной, соответственно. Константа `TARGET_COUNT` определяет размер множества значений, которые может принимать выходная переменная, генерируемая «флибом».

Для описанного представления хромосомы в работе [4] были реализованы операции скрещивания и мутации.

Операция скрещивания. Операцию скрещивания для хромосом указанной структуры можно описать следующей последовательностью шагов:

1. Случайно выбирается одно из состояний нового «флиба».
2. В новый «флиб» добавляются состояния первого родителя с номерами, меньшими, чем у выбранного состояния, и состояния второго родителя с номерами, большими, чем у выбранного состояния.
3. Формируется и добавляется новое состояние, образованное из состояний обоих родителей с номерами, равными номеру выбранного состояния. Переходы из нового состояния формируются с помощью аналогичного алгоритма одноточечного скрещивания.

Операция мутации. Операцию мутации для указанной структуры хромомосмы можно описать следующей последовательностью операций:

1. Случайным образом выбирается состояние «флиба».
2. Случайным образом выбирается дуга перехода из выбранного состояния.
3. Случайным образом выбирается, что будет подвергнуто изменению – значение выходной переменной, соответствующее выбранному переходу, или номер состояния, в которое ведет выбранный переход.
    a. Если было определено, что изменяется значение выходной переменной, то ей присваивается выбранное случайным образом значение.

b. Если было определено, то изменяется номер состояния, в которое ведет переход, то переход перенаправляется в случайно выбранное состояние.

Отметим, что генерации нового поколения вследствие применения операторов скрещивания и мутации дуги переходов во «флибах» изменяются случайным образом. Из-за этого в некоторых «флибах» могут появиться состояния, в которые невозможно попасть из начального состояния ни при какой последовательности значений входной переменной. Такие состояния назовем *недостижимыми*. Наоборот, состояния, в которые можно попасть из начального состояния при подаче на вход «флиба» некоторой последовательности состояний среды, назовем *достижимыми*.

Таким образом, после применения операций мутации и скрещивания в некоторых «флибах» могут появиться недостижимые состояния. Так как недостижимые состояния не несут никакой полезной информации, то в работе [4] предлагается *алгоритм восстановления связей*, применение которого к «флибу» преобразует дуги его переходов так, чтобы в нем не было недостижимых состояний – все состояния были достижимыми.

Этот алгоритм можно записать в виде следующей последовательности шагов:

1. Формируется список доступных состояний – для этого можно применить любой метод обхода графов, например, обход в глубину или в ширину.
2. Выполняется цикл по всем состояниям. Если обрабатываемое на очередной итерации цикла состояние не входит в число достижимых, то для него выполняются следующие операции:
    a. Случайным образом выбирается состояние из списка достижимых состояний.
    b. Случайным образом выбирается одна их дуг выбранного состояния.
    c. В текущем состоянии одна из дуг заменяется дугой, которая ведет в то же состояние, что и дуга, выбранная в пункте b.
    d. Выбранная в пункте b, дуга заменяется дугой, которая ведет в текущее состояние.
    e. Обновляется список достижимых состояний – в него добавляется текущее состояние и все состояния, достижимые из него.

Отметим, что основным отличием алгоритма из работы [4] от алгоритма из работы [1] является то, что во всех «флибах» в каждом поколении все состояния являются достижимыми – никакой лишней неиспользуемой информации в хромосомах не содержится.

Приведем результаты вычислительных экспериментов, проведенных в работе [4]. Было проведено два вычислительных эксперимента, которые отличались битовой маской, задающей изменение среды, количеством состояний «флиба» и количеством поколений, в течение которых шла эволюция «флибов». Кроме этого, каждый раз алгоритм запускался как с применением восстановления связей между состояниями, так и без применения этого алгоритма.

При задающей среду битовой маске `1111010010111101001`, двадцати состояниях «флиба» и 400 поколениях наивысшая достигнутая точность предсказания равна 100% как при использовании восстановления связей между состояниями, так без использования этого алгоритма. Средняя точность предсказания составляет 92,26% без использования восстановления алгоритма восстановления связей, и 93,48% – при его использовании.

При битовой маске среды `10101111011000111101111100011001`, тридцати состояниях «флиба» и 1600 поколениях наивысшая достигнутая точность составила 97% как при использовании алгоритма восстановления связей между состояниями, так без его использования. Средняя точность предсказания при использовании восстановления связей между состояниями составила 92,72%, а без использования этого алгоритма – 90,44%.

Из изложенного следует, что предлагаемая в работе [4] структура хромосомы позволяет более эффективно строить «флибы», нежели чем при использовании хромосом, структура которых описана

в работе [1]. Более того, применение операции восстановления связей между состояниями позволяет получать более точные предсказатели, чем при применении генетического алгоритма, который не использует такой операции.

2.2.1.6. Генетический алгоритм в итерированной дилемме узника

В работе [26] для построения оптимальных стратегий, описываемых конечным автоматом, для итерированной дилеммы узника применяется генетический алгоритм. При этом строятся не обычные конечные автоматы, а вероятностные – их переходы снабжены вероятностями, с которыми выбирается этот переход. При этом сумма вероятностей по всем переходам из состояния равна единице. Основной целью работы [26] является не построение оптимального автомата, а описание алгебраической структуры, описывающей поведение в играх типа «итерированной дилеммы узника». Эта структура оказывается удобной для оптимизации с помощью генетических алгоритмов.

Опишем более подробно эту структуру и способ применения к ней генетических операторов мутации и скрещивания. Каждый автомат описывается несколькими матрицами. Их количество совпадает с размером входного алфавита, а размер каждой матрицы составляет `n×n`, где как `n` обозначено число состояний в автомате. Обозначим матрицу, соответствующую входному символу `c` как `T(c)`. В матрице `j`-ый элемент ее `i`-ой строки равен вероятности перехода из состояния `i` в состояние `j` по входному символу `c`. Все элементы матриц являются вещественными числами из отрезка `[0, 1]`. Сумма элементов каждой строки каждой матрицы равна единице.

При этом отметим, что выходной символ автомата зависит только от состояния – рассматриваемые в работе [26] автоматы являются автоматами Мура. Более того, соответствие состояний выходным символам задано заранее и не может быть изменено. Таким образом, это соответствие не должно быть закодировано в хромосоме.

Исходя из изложенного, в хромосому автомата были включены только матрицы переходов. Для получения хромосомы матрицы записываются построчно. Например, если размер входного алфавита равен двум, количество состояний автомата равно четырем, матрицы переходов приведены в табл. 17 и в табл. 18, то хромосома будет иметь следующий вид: `(0,1; 0,2; 0,5; 0,2); (0,2; 0.3; 0,5; 0,0); (0,3; 0,1; 0,5; 0,1); (0,4; 0,0; 0,5; 0,1;); (0,2; 0,1; 0,1; 0,6); (0,2; 0,3; 0,1; 0,4); (0,2; 0,3; 0,1; 0,4); (0,4; 0,1; 0,1; 0,4); (0,6; 0,1; 0,1; 0,2).`

Таблица 17. Матрица переходов, соответствующая входному символу «0»

0,1	0,2	0,5	0,2
0,2	0,3	0,5	0,0
0,3	0,1	0,5	0,1
0,4	0,0	0,5	0,1

Таблица 18. Матрица переходов, соответствующая входному символу «1»

0,2	0,1	0,1	0,6
0,2	0,3	0,1	0,4
0,4	0,1	0,1	0,4
0,6	0,1	0,1	0,2

Отметим, что при используемом методе представления матриц внутри хромосомы матрицы по-прежнему отделяются друг от друга некоторыми границами. Также отметим, что вещественные

числа не предлагается кодировать битовыми строками. Таким образом, все генетические операции оперируют с каждым числом, как с единым целым.

Приведем подробное описание генетических операций скрещивания и мутации для описанной структуры хромосомы.

Операция скрещивания. Рассмотрим две родительские хромосомы. Запишем их представления друг за другом в виде одной строки. После этого применим к полученной последовательности строк случайную перестановку. Первая половина полученной последовательности будет хромосомой одного из потомков, вторая половина – другого.

Операция мутации. Для выполнения операции мутации случайно выбирается одна из матриц, задающая автомат. У выбранной матрицы случайно выбранная строка заменяется на случайно сгенерированную.

Стратегии для итерированной дилеммы узника могут быть заданы невероятностным автоматом Мили. Примеры таких автоматов приведены на рис. 62 и рис. 63.

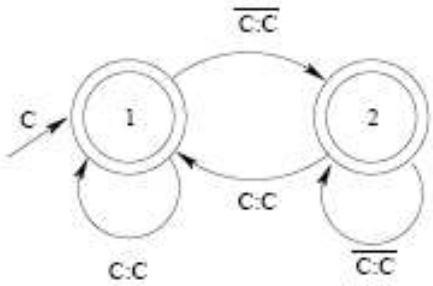


Рис. 62. Автомат, задающий стратегию «око за око»

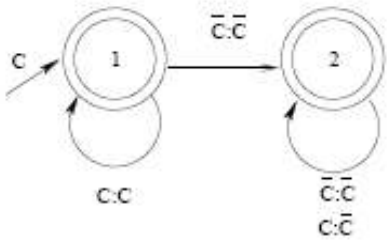


Рис. 63. Автомат, задающий «мстящую» стратегию

Поясним обозначения, используемые на этих рисунках. Метки на переходах имеют следующий формат: `входной символ : выходной символ`. Начальным состоянием в обоих автоматах является состояние 1. Символ c соответствует сотрудничеству (cooperate), а символ $\bar{c}$ – предательству (betray).

Первый из этих автоматов (рис. 62) задает стратегию «повторять действия противника на предыдущем ходе», а второй начинает выбирать действие «предать» как только так поступил его противник. Недостаток этих автоматов состоит в том, что они статически задают стратегии поведения и плохо подходят для эволюционной оптимизации.

Для устранения этого недостатка в работе [26] предлагается использовать вероятностный автомат с двумя состояниями, который приведен на рис. 64.

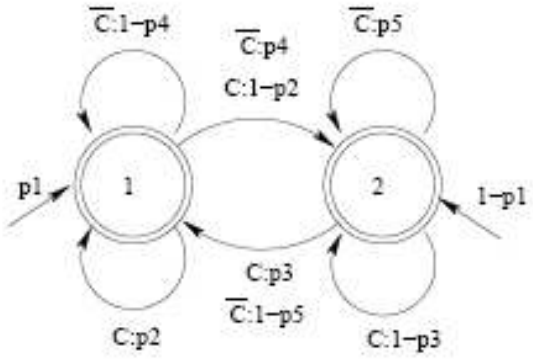


Рис. 64. Вероятностный автомат для итерированной дилеммы узника

Первое состояние этого автомата соответствует выбору действия «сотрудничать», второе – «предать». Матричное представление этого автомата приведено в табл. 19 и табл. 20.

Таблица 19. Матрица переходов, соответствующая входному символу «*C*»

p_2	$1 - p_2$
p_3	$1 - p_3$

Таблица 20. Матрица переходов, соответствующая входному символу « $\overline{C}$ »

p_4	$1 - p_4$
p_5	$1 - p_5$

Кроме двух указанных автоматов необходимо еще задать вероятность того, что первое состояние является начальным – обозначим ее как p_1. Тогда вероятность того, что начальным будет второе состояние, равна $1 - p_1$.

Применение генетического алгоритма с описанными выше операциями мутации и скрещивания позволило построить вероятностный автомат с двумя состояниями, который показал результаты, сравнимые с результатом детерминированного автомата из восьми состояний, рассмотренного следующем разделе.

2.2.1.7. Генетическое программирование в итерированной дилемме узника

В работе [22] для поиска оптимальной стратегии поведения в итерированной дилемме узника используется генетическое программирование – хромосома представляет автомат в виде графа переходов, а в процессе оптимизации используются только операции мутации, а операции скрещивания не используются.

Число состояний в исследуемых автоматах было ограничено восемью для того, чтобы построенные автоматы могли быть в дальнейшем проанализированы вручную. При этом начальное поколение состояло из 100 автоматов, в каждом из которых было от одного до пяти состояний. В каждом состоянии по каждой паре входных символов {(C, C), (C, D), (D, C), (D, D)} на выход подавался случайный символ из множества {C, D}.

В дальнейшем эти автоматы подвергались мутациям, в число которых входили:

1. Изменение начального состояния.
2. Добавление состояния.
3. Удаление состояния.
4. Изменение выходного символа.
5. Изменение перехода.

Каждый раз выбирался случайно и равновероятно одна из разновидностей мутации. После этого случайно выбирался один из способов выполнения такой мутации. Например, если была выбрана мутация «изменение выходного символа», то случайно выбиралось состояние автомата. Далее случайно выбирался переход из этого состояния, и изменялся символ, соответствующий этому переходу.

Для определения приспособленности особей проводился турнир по круговой системе. При этом каждая игра состояла из 151 раунда. После этого выбирались 100 лучших особей, которые переходили в следующее поколение. Этот процесс продолжался на протяжении 50 поколений.

С помощью этого алгоритма при одном из запусков был построен автомат из восьми состояний, граф переходов которого приведен на рис. 65.

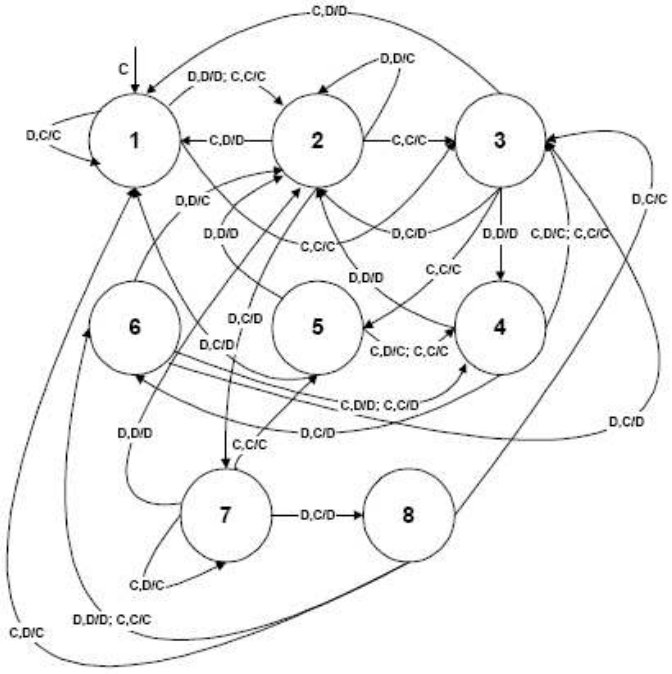


Рис. 65. Автомат из восьми состояний для итерированной дилеммы узника

2.2.2. Рекомендации по выбору структуры хромосомы для различных классов задач

В настоящем разделе сравниваются описанные ранее структуры хромосом и приводятся рекомендации по выбору структуры хромосомы для задач, сходных с рассмотренными ранее.

2.2.2.1. Задача об «Умном муравье»

Ранее (разд. 2.2.1) были рассмотрены различные структуры хромосом, которые применяются для построения оптимальных управляющих автоматов в задаче об «Умном муравье»: битовые строки (разд. 2.2.1.1), битовые строки с приведением автоматов перед кодированием к каноническому виду (разд. 4.2.1.2) и генетическое программирование с хранением автоматов в виде графов переходов (разд. 2.2.1.3). Далее будет проведено их сравнение, и приведены некоторые рекомендации по выбору структуры хромосомы, пригодной для решения сходных задач.

Табл. 21 содержит краткую сравнительную характеристику описанных ранее структур хромосом для задачи об «Умном муравье». В этой таблице приведены основные параметры структур хромосом (например, «Метод построения автоматов», «Используются ли операторы мутации и скрещивания?», «Какие используются дополнительные операции?», «Является ли фиксированным число состояний в автомате?»). Кроме этого, в ней приведены параметры, показывающие сложность программной реализации метода, использующего такую структуру хромосом (например, «Требуется ли преобразование хромосомы в автомат при вычислении функции приспособленности?»).

Рассмотрение этой таблицы позволяет сделать вывод о том, что первые две структуры хромосом во многом сходны, в то время как третья (которая используется в генетическом программировании) существенно от них отличается. Отсутствие необходимости декодирования хромосомы для вычисления функции приспособленности у хромосом, использующихся в генетическом программировании, позволяет утверждать, что этот подход менее трудоемок в реализации. Более развитый набор генетических операций, включающий, кроме операции мутации, операцию скрещивания, позволяет надеться на то, что построение оптимальных автоматов с помощью генетических алгоритмов может быть более эффективным.

Таблица 21. Сравнение структур хромосом в задаче об «Умном муравье»

Параметр, по которому выполняется сравнение	Битовые строки	Битовые строки + приведение к каноническому виду	Графы переходов + «модули»
Метод построения автоматов	Генетический алгоритм	Генетический алгоритм	Генетическое программирование
Объект, который представляется в виде хромосомы	Конечный автомат Мили	Конечный автомат Мили	Конечный автомат Мили
Требуется ли преобразование хромосомы в автомат при вычислении функции приспособленности?	Да, необходимо декодировать битовую строку	Да, необходимо декодировать битовую строку	Нет, хромосома представляет собой граф переходов автомата
Используется ли оператор мутации?	Да	Да	Да
Используется ли оператор скрещивания?	Да	Да	Нет
Какие используются дополнительные операции?	Не используются	Приведение к каноническому виду с помощью алгоритмов *SFS* и *MTF*	Объединение в «модули» – «сжатие» и «расширение»
Является ли фиксированным число состояний в автомате?	Да (32 состояния, некоторые из которых могут быть недостижимы)	Да (32 состояния, некоторые из которых могут быть недостижимы)	Нет (максимальный размер автомата – 32 состояния)
Действия, которые необходимо произвести при кодировании автомата в хромосому	Преобразование автомата в битовую строку по методу, описанному в разд. 2.2.1.1	Приведение автомата к каноническому виду и преобразование его в битовую строку	Отсутствуют – хромосома представляет собой граф переходов автомата

Сравнение указанных структур хромосом с точки зрения их эффективности для построения управляющих автоматов для задачи об «Умном муравье» приведено в табл. 22. В этой таблице систематизирована информация, приведенная в разд. 2.2.1.1 – 2.2.1.3.

Таблица 22. Результаты применения указанных структур хромосом в задаче об «Умном муравье»

	Битовые строки	Битовые строки + приведение к каноническому виду	Графы переходов + «модули»
Минимальное число состояний в автомате, решающем задачу об «Умном муравье»	13	8	11
Число особей в одном поколении	65536	65536	300
Число поколений, требуемое для построения простейшего автомата	52	51	От 418 до 4051
Число проанализированных автоматов	3 303 014	Порядка 3 000 000	От 63 000 до 607 950

Таким образом, для задачи об «Умном муравье» и аналогичных задач наиболее эффективным оказывается представление хромосом в виде битовых строк с использованием приведения автоматов к каноническому виду перед их кодированием в виде битовых строк. Это представление за счет некоторого усложнения реализации (необходимо реализовать алгоритмы *SFS* и *MTF*) позволяет достаточно быстро (анализ порядка трех миллионов автоматов) построить автомат, содержащий восемь состояний. Для сравнения отметим, что при использовании просто битовых строк при таком же количестве проанализированных автоматов был построен автомат с 13 состояниями.

2.2.2.2. Задача о «флибах»

Ранее (разд. 2.2.1) были рассмотрены различные структуры хромосом, которые применяются для построения оптимальных управляющих автоматов в задаче об «флибах»: строки (разд. 2.2.1.4) и структурированное представление (разд. 2.2.1.5). Далее будет проведено их сравнение и даны некоторые рекомендации по выбору структуры хромосом, пригодных для решения сходных задач.

В табл. 23 приведена краткая сравнительная характеристика описанных ранее структур хромосом для задачи об «флибах». В этой таблице приведены основные параметры структур хромосом (например, «Метод построения автоматов», «Используются ли операторы мутации и скрещивания?», «Какие используются дополнительные операции?», «Является ли фиксированным количество состояний в автомате?»). Кроме этого, в ней приведены параметры, показывающие сложность программной реализации метода, использующего такую структуру хромосом (например, «Требуется ли преобразование хромосомы в автомат при вычислении функции приспособленности?»).

Изучение этой таблицы позволяет ожидать, что второй способ (структурированное представление) позволяет добиться лучших результатов в этой задаче, так как при его использовании не требуется декодирования строки для получения графа переходов автомата. Еще одной причиной для подобных ожиданий является использование метода «Восстановление связей между состояниями», с помощью которого обеспечивается достижимость всех состояний. Отметим также, что оба метода обладают примерно одинаковой сложностью реализации, так как в одном требуется декодирование строки для получения графа переходов автомата, а во втором – реализация алгоритма восстановления связей между состояниями, который включает в себя обход графа переходов автомата с помощью поиска в глубину или в ширину.

Таблица 23. Сравнение структур хромосом в задаче о «флибах»

Параметр, по которому выполняется сравнение	Строки (разд. 2.2.1.4)	Структурированное представление (разд. 2.2.1.5)
Метод построения автоматов	Генетический алгоритм	Генетический алгоритм
Объект, который представляется в виде хромосомы	Конечный автомат Мили	Конечный автомат Мили
Требуется ли преобразование хромосомы в автомат при вычислении функции приспособленности?	Да, необходимо декодировать строку	Нет, автомат хранится в виде графа переходов
Используется ли оператор мутации?	Да	Да
Используется ли оператор скрещивания?	Да	Да
Какие используются дополнительные операции?	Никакие	Восстановление связей между состояниями
Является ли фиксированным число состояний в автомате?	Да, некоторые состояния могут быть недостижимы	Да, все состояния достижимы
Действия, которые необходимо произвести при кодировании автомата в хромосому	Преобразование автомата в строку по методу, описанному в разд. 2.2.1.4	Отсутствуют, хромосома хранит граф переходов автомата

Приведем результаты использования описанных структур для построения автоматов в задаче о «флибах» для сред, заданных масками `1111010010111101001` и `10101111011000111101111110011001` (табл. 24).

Таблица 24. Результаты применения структур хромосом в задаче о «флибах»

	Строки (разд. 2.2.1.4)	Структурированное представление (разд. 2.2.1.5)
Первая маска среды, 400 поколений, 20 состояний	Лучший результат: 88% Средний результат: 78,3%	Лучший результат: 100% Средний результат: 93,48%
Вторая маска среды, 1600 поколений, 30 состояний	Лучший результат: 87% Средний результат: 75,86%	Лучший результат: 97% Средний результат: 92,72%

Из табл. 18 следует, что использование структурированного представления хромосом с использованием операции восстановления связей между состояниями позволяет добиться лучших результатов в задаче о «флибах». При этом сложность реализации этого метода примерно такая же, как и у метода, использующего представление хромосом в виде строк. Таким образом, для задачи о «флибах» и аналогичных задач лучшим оказывается структурированное представление хромосом с использование восстановления связей между состояниями (разд. 2.2.1.5).

2.2.2.3. Итерированная дилемма узника

В разд. 2.2.1 рассмотрены различные структуры хромосом, которые применяются для построения оптимальных управляющих автоматов в итерированной дилемме узника: графы переходов и представление вероятностных автоматов в виде матриц (разд. 2.2.1.6). Далее будет проведено их сравнение и даны некоторые рекомендации по выбору структуры хромосом, пригодных для решения сходных задач.

В табл. 25 приведена краткая сравнительная характеристика описанных ранее структур хромосом для итерированной дилеммы узника. В этой таблице приведены основные параметры структур хромосом (например, «Метод построения автоматов», «Используются ли операторы мутации и скрещивания?», «Какие используются дополнительные операции?», «Является ли фиксированным количество состояний в автомате?»). Кроме этого, в ней приведены параметры, показывающие сложность программной реализации метода, использующего такую структуру хромосом (например, «Требуется ли преобразование хромосомы в автомат при вычислении функции приспособленности?»).

Информация, представленная в табл. 19, позволяет ожидать, что более эффективным окажется представление хромосом в виде вероятностных автоматов и их матриц переходов. Это связано с вероятностным характером задачи, включающей в себя несколько повторений. С другой стороны, детерминированный автомат позволяет добиться более стабильных результатов, к тому же его поведение может быть изучено вручную, в то время как поведение вероятностного автомата предсказать намного сложнее. Кроме того, для матричного представления используется оператор скрещивания, применение которого обычно позволяет добиться лучших результатов, нежели чем при использовании только мутаций.

Таким образом, для итерированной дилеммы узника и сходных с ней задач более эффективным оказывается представление вероятностных автоматов в виде матриц переходов – эта структура хромосом описана в разд. 2.2.1.6.

Таблица 25. Сравнение структур хромосом в итерированной дилемме узника

Параметр, по которому выполняется сравнение	Матрицы (разд. 2.2.1.6)	Графы переходов (разд. 2.2.1.7)
Метод построения автоматов	Генетический алгоритм	Генетическое программирование
Объект, который представляется в виде хромосомы	Вероятностный автомат с действиями в состояниях	Конечный автомат Мили
Требуется ли преобразование хромосомы в автомат при вычислении функции приспособленности?	Да, необходимо декодировать матричное представление автомата	Нет, автомат хранится в виде графа переходов
Используется ли оператор мутации?	Да	Да
Используется ли оператор скрещивания?	Да	Нет
Какие используются дополнительные операции?	Никакие	Никакие
Является ли фиксированным число состояний в автомате?	Да, количество состояний равно двум	Нет, некоторые состояния могут быть недостижимы
Действия, которые необходимо произвести при кодировании автомата в хромосому	Преобразование автомата в матрицы переходов по методу, описанному в разд. 2.2.1.6	Отсутствуют, хромосома хранит граф переходов автомата

2.3. Методы построения оценочных функций, применяемых при генерации автоматов управления системами со сложным поведением

Генетические алгоритмы – хорошо распараллеливающиеся математические алгоритмы, которые преобразуют популяцию отдельных математических объектов (особей) в новую популяцию, используя операции, моделирующие естественные эволюционные процессы. В основе эволюции лежит тот факт, что чем более эффективна (лучше приспособлена) особь, тем выше ее шансы выжить и оставить потомство. Для определения эффективности (приспособленности) особей используются, так называемые, оценочные функции. При разработке генетического алгоритма для конкретной задачи важно правильно построить оценочную функцию.

2.3.1. Анализ существующих методов построения оценочных функций

Существует множество способов задание оценочных функций. Например, оценочная функция может быть явно заданной – возвращать вещественное значение, зависящее только от определенной особи и постоянное для этой особи. В качестве примера для такой функции может служить функция, получающая на вход битовую строку, задающую некоторый алгоритм, и возвращающая число, соответствующее эффективности этого алгоритма. В этом случае приспособленность особи определяется ее способностью решать поставленную задачу. Например, отслеживать путь [34].

Оценочная функция также может основываться на ранге битовых строк в популяции [50]. К сожалению, в том случае, когда для кодирования особей используется способ, отличный от битовых строк, применение этой функции становится затруднительным.

Можно построить оценочную функцию, в основе которой будет лежать на один из методов отбора. Например, турнирный отбор [28].

В тех случае, когда структуры, использующиеся для кодирования особей, имеют переменную длину, может потребоваться найти такое решение поставленной задачи, длина которого будет минимальной или хотя бы ограниченной разумными пределами. Это не сложно сделать, если включить в оценочную функцию размер структуры, использующейся для кодирования особи. Пусть f_i определяет на сколько хорошо i-я особь решает поставленную задачу, а $length_i$ – размер структуры, использующейся для кодирования i-ой особи. В качестве оценочной функции можно использовать такую функцию F_i, что

1. $F_i < F_j$ для любых i и j, при которых $f_i < f_j$;
2. $F_i < F_j$ для любых i и j, при которых $f_i = f_j$ и $length_i < length_j$;
3. $F_i = F_j$ для любых i и j, при которых $f_i = f_j$ и $length_i = length_j$.

Построенная таким образом оценочная функция, в первую очередь, отдает предпочтение тем особям, которые лучше решают поставленную задачу. При этом она также учитывает и размер структуры, использующейся для кодирования особи. Примером задачи, в которой была бы полезна такая оценочная функция, может служить любая задача, решаемой с помощью представления в виде конечного автомата. При этом, как правило, желательно построить автомат с минимальным числом состояний. Число состояний автомата можно использоваться вместо длины структуры, задающей особь.

Оценочная функция может быть определена через взаимодействие особей с окружающей средой. В этом случае значение оценочной функции для конкретной особи изменяется в зависимости от окружающей среды. Один из путей построения такой оценочной функции состоит в проведении соревнований между особями популяции. В работе [31] Дж. Холланд комбинирует совместную эволюцию и генетические алгоритмы в ECHO-системе. Цель его работы – исследование совместной эволюцию организмов, заданных строками фиксированной длины, в «миниатюрном мире». В ECHO всего одна популяция искусственных организмов. Окружающая среда для каждого организма состоит из всех остальных организмов.

Еще один пример оценочной функции зависящей от всех особей в текущем поколении, приведен в работе [48]. В этой работе битовые строки, задающие особей, интерпретируются, как стратегии для игры в пятнашки. В указанной игре в пятнашки принимает участие два или более игроков, один из которых назначается преследователем. Преследователь пытается догнать других игроков, которые, в свою очередь, пытаются убежать от него. Такое поведение часто встречается в природе. Большинство взаимодействий типа «хищник – жертва» включают в себя преследование и бегство. Игра в пятнашки включает в себя смену ролей. Для успешной игры требуется, как умение догонять, так и умение убегать. Цель преследователя догнать другого игрока. Когда преследователь приближается на достаточно близкое расстояние к другому игроку, роли меняются и беглец становиться преследователем. Целью работы [48] было автоматически построить контроллер для игры в пятнашки, используя эволюционный процесс, оценочная функция которого основана только на результатах соревнований между контроллерами.

В традиционной игре в пятнашки нет формального метода для подсчета очков. Интуитивно понятно, что цель игры – не стать преследователем. Выигравшей в соревновании между двумя котроллерами, считается тот, для которой остается беглецом больший процент времени (число шагов) от общего времени игры. При таком подходе используется нормализованная мера определения приспособленности для одного соревнования: ноль – для худшего контроллера, единица – для лучшего. Для определения эффективности контроллера он сравнивается с другими контроллерами.

Для сравнения контроллеров выполняются две серии из четырех игр между ними. В первой серии первый контроллер начинает игру преследователем, во второй серии – второй. Перед каждой игрой начальные позиции и направления игроков задаются случайным образом. Из-за этого игры между одними и теми же игроками могут приводить к разным результатам. Каждая игра состоит из 25 шагов. Очки, которые получает контроллер за игру – число шагов, в течение которых он не был преследователем, деленное на 25.

Для определения приспособленности контроллера он участвует в сериях игр против шести других игроков. Оппоненты выбираются с помощью равномерного случайного отбора. Очки, полученные во всех играх, усредняются для определения окончательного значения оценочной функции. Значение оценочной функции, равное 50%, означает, что способности контроллера сопоставимы со средними способностями особей в популяции. Значение оценочной функции, превышающее 50%, означает, что способности контроллера выше среднего уровня способностей особей в популяции. Так как противники выбираются случайным образом, а начальные параметры игр рандомизированные, то значение оценочной функции может существенно варьироваться и позволяет получить только грубую оценку фактической эффективности особи.

Несколько исследователей (например, в работах [15, 22, 42]) экспериментировали с итерированной дилеммой узника. В этой задаче оценочная функция основывается на таблице баллов для следующих ситуаций: взаимное сотрудничество, взаимное противостояние, противостояние-сотрудничество. Особи с различными стратегиями из эволюционирующей популяции играют друг против друга. Необходимо отметить, что в работе [15] эволюционирующие стратегии играли против девяти лучших компьютерных программ, предложенных на международном компьютерном чемпионате. Лучшая стратегия для одного игрока (стратегия задавалась строкой из 70 бит с возможностью заглядывать на три хода назад) была получена в результате соревнований с девятью лучшими стратегиями, построенными вручную. Эти стратегии и задавали окружающую среду, к которой пыталась приспособиться популяция. В работах [22, 42], оценочная функция определялась через взаимодействие особей с изменяющимся окружением, которое представлено в виде эволюционирующей популяции. В работе [42] популяция состояла из автоматов с 16 состояниями, которые задавалась с помощью битовых строк длиной в 148 символов. Каждая строка в популяции представляла собой полную стратегию, руководствуясь которой игрок принимал решения. Автомат определял выбор хода игроком для любой последовательности ходов противника.

Как отмечалось выше, другой путь для создания изменяющегося окружения – использование конкурирующих популяций. В работе [29] эволюционировали две популяции: популяция сортирующих сетей и популяция перестановок чисел. Каждая перестановка состояла из 16 элементов. Чем в большем числе перестановок из конкурирующей популяции может правильно упорядочить элементы сортирующая сеть, тем выше ее относительная эффективность (приспособленность к среде). И наоборот, относительная приспособленность перестановки чисел определяется числом сортирующих сетей, которые не могут упорядочить ее элементы. Каждая популяция создает окружающую среду, к которой пытается приспособиться конкурирующая популяция. Каждая особь в каждой популяции участвует в определенном числе состязаний с особями конкурирующей популяции. Значение оценочной функции для каждой сортирующей сети равно числу тестовых перестановок, элементы которых она которых она правильно упорядочила в процессе соревнований. Значение оценочной функции для каждой перестановки равно числу сортирующих сетей, которые не смогли правильно упорядочить ее элементы в процессе соревнований. В работе [29] отмечается, что каждая популяция эволюционируя, училась эксплуатировать слабости и избегать сильных сторон противоборствующей популяции. Такой подход, напоминающий природную модель «хищник – жертва», позволяет избежать необходимости тестировать каждую сортирующую сеть на всех перестановках 16 элементов для того, чтобы определить абсолютную эффективность каждой особи.

Еще одна оценочная функция предлагается в работе [47]. В ней генетический алгоритм используется для создания самовоспроизводящихся программ. В данном подходе особи располагаются в *Tierra's Reaper* -очередях. Особи в начале очереди, как правило, наиболее старые или не способные к самовоспроизведению. Они удаляются, когда становится мало памяти. В такой системе оценочная функция определяется через *Reaper*. Автор данного подхода (тропический эколог) моделировал с помощью компьютера экологические системы – повторяющие системы, найденные в природе.

Одна из основных целей при разработке нового или улучшении существующего генетического алгоритма – увеличение скорости, с которой генетический алгоритм сходится к лучшему решению задачи. В связи с тем, что вычисление оценочной функции, как правило, является наиболее трудоемкой частью алгоритма, эффективность алгоритма часто измеряется в виде числа вычислений оценочной функции, которое было сделано до получения лучшего решения. Для любого генетического алгоритма производительность может сильно зависеть как от структуры пространства поиска, оценочной функции, так и от других его параметров. Например, от размера популяции и вероятности применения оператора мутации.

В работе [32] демонстрируется эффективность инкрементальной эволюции, при которой избирательность оценочной функции увеличивается поэтапно.

В работе [33] рассматриваются оценочные функции и способы их адаптации для увеличения эффективности генетических алгоритмов. В работе [39] показано, что если оценочная функция может быть представлена в качестве линейной комбинации отдельных оценочных функций, то можно настроить коэффициенты в этой линейной комбинации в процессе работы генетического алгоритма таким образом, что скорость, с которой сходится алгоритм, существенно увеличится.

Оценочная функция определят направление поиска для генетического алгоритма. Если операторы скрещивания и мутации являются движущей силой эволюции, то оценочная функция определят цель эволюции и пути достижения этой цели. Для того чтобы генетический алгоритм смог найти «идеальное» решение при построении оценочной функции необходимо учесть все характеристики решения, важные для исследователя.

В том случае, когда целью исследователя является изучение эволюционных процессов (получение идеального решения отходит на второй план) используются оценочные функции, основанные на конкуренции особей. Использование совместной эволюции позволяет изучать такие особенности, как возникновение кооперации между особями популяции и взаимное влияние популяций на эволюцию друг друга. Оценочные функции, основанные на изменяющемся окружении также, полезны, если использование явно заданной оценочной функции невозможно (например [29]).

В результате исследования установлено влияние оценочных функций на результаты применения генетических алгоритмов. Для повышения качества решений предлагается использовать методы, учитывающие специфику задачи при построении таких функций. Таким образом, работы по разработке рекомендаций по выбору оценочных функций являются актуальными.

2.3.2. Предлагаемые методы построения оценочных функций

2.3.2.1. Пример простейшей оценочной функции

В классическом генетическом алгоритме оценочная функция определяется как $f_i \big/ \overline{f}$, где f_i – приспособленность особи i, а $\overline{f}$ – средняя приспособленность всех особей в популяции. Значение оценочной функции используется для формирования промежуточной популяции особей (промежуточная популяция используется для выбора родительских особей для скрещивания). Для

каждой особи i, где $f_i/\overline{f}$ больше единицы, целая часть числа обозначает, сколько копий этой особи гарантированно добавляется в промежуточную популяцию. Для всех особей (включая те, у которых $f_i/\overline{f}$ меньше 1.0) в промежуточную популяцию добавляются их копии с вероятностью равной дробной части $f_i/\overline{f}$. Например, одна копия особи для которой $f_i/\overline{f} = 1.36$, сразу добавляется в промежуточную популяцию и еще одна копия добавляется с вероятностью 0.36. Копия особи, значение оценочной функции для которой $f_i/\overline{f} = 0.54$, будет добавлена в промежуточную популяцию с вероятностью 0.54.

2.3.2.2. Построение оценочной функции для совместной эволюции популяций

Эволюционные процессы в природе не редко описывают так, как если бы популяция особей пыталась адаптироваться к неизменной окружающей среде. Такой подход является только первым уровнем приближения к реальной ситуации. Окружающая среда фактически состоит как из физической среды (которая обычно относительно постоянна), так и из независимо действующих биологических популяций особей, которые одновременно пытаются адаптироваться к их окружающей среде. Действия каждой из этих независимо существующих биологических популяций (видов) обычно оказывает эффект на другие виды. Иными словами, окружающая среда отдельно взятого вида включает в себя все остальные биологические виды, которые обитают в той же физической среде, что и выбранный вид, в течение промежутка времени, когда он пытается адаптироваться к среде. В биологии термин совместная эволюция зачастую используется для того, чтобы подчеркнуть тот факт, что все виды, одновременно существующие в некоторой окружающей среде, совместно эволюционируют.

Биологический пример совместной эволюции представлен в работе [31]. Определенные виды растений могут столкнуться с окружающей средой, содержащей насекомых, которые питаются этими растениями. Защищаясь от хищников (увеличивая тем самым вероятность своего выживания), растения через определенное время, могут приобрести крепкую кожуру, которая будет мешать насекомым их поедать. Спустя некоторый период времени, насекомые могут приобрести сильные челюсти, которые позволят популяции насекомых продолжить поедать растения и увеличат вероятность их выживания в окружающей среде. Через некоторое временя растения могут научиться вырабатывать яд, который будет защищать их от насекомых. Однако насекомые снова через некоторое время, могут научиться вырабатывать пищеварительный фермент, нейтрализующий эффект яда. Благодаря нему популяция насекомых сможет продолжить питаться растениями.

В иерархическом алгоритме совместной эволюции используется две (иногда более) популяции особей. Окружающей средой для первой популяции является вторая популяция. И наоборот, окружающей средой для второй популяция является первая популяция.

Процесс совместной эволюции обычно начинается с того, что обе популяции состоят из плохо приспособленных особей (если производить измерение абсолютной приспособленности). Затем первая популяция пытается приспособиться к окружающей среде, в качестве которой для нее выступает вторая популяция. В то же время вторая популяции пытается адаптироваться к окружающей среде, созданной первой популяцией.

Этот процесс осуществляется с помощью тестирования каждой особи из первой популяции с каждой особью из второй популяции (или из подмножества особей второй популяции). В результате

такого тестирования определятся относительная (конкурентная) приспособленность особи. Относительная приспособленность одной особи из первой популяции отражает ее эффективность по отношению к окружающей среде, роль которой играет вторая популяция. Затем каждая особь из второй популяции тестируется с каждой особью из первой популяции (или из подмножества особей первой популяции).

Необходимо отметить, что измерение относительной приспособленности особи в процессе совместной эволюции не является измерением абсолютной приспособленности особи по сравнению с идеальным противником. Это просто относительное измерение, определяющее, на сколько хорошо особь приспособилась к текущей популяции особей противников. Допустим, существует популяция боксеров, которые умеют наносить удары только левой рукой и существует особь возможности защиты которой позволяют ей хорошо защищаться, но только от ударов левой рукой. Относительная приспособленность такой особи будет очень высока. Однако абсолютная приспособленность будет низкой, когда тестирование будет проводиться с участием противника, который умеет наносить удары как левой, так и правой рукой (идеальные противник).

Даже на начальном этапе, когда особи обоих популяций плохо приспособлены (и абсолютно, и относительно), некоторые особи будут иметь хотя бы немного лучшую относительную приспособленность, чем другие. Это означает, что некоторые особи в каждой популяции более эффективны при взаимодействии с особями из популяции противников по сравнению с большинством особей из собственной популяции.

Операция «воспроизведение потомства пропорционально приспособленности» (основанная на принципе Дарвина, суть которой в том, что выживают и оставляют потомство наиболее приспособленные особи) применяется ко всем популяциям. В качестве значения оценочной функции используется текущая относительная приспособленность особи. После этого оператор скрещивания может применяться к такой паре родительских особей, в которой, по крайней мере, одна особь выбрана на основе ее относительной приспособленности.

В течение всего периода времени обе популяции стремятся совместно эволюционировать и достичь высшего уровня эффективности, определяющегося абсолютной приспособленностью. Обе популяции эволюционируют, не ставя перед собой цель, достичь абсолютной приспособленности к некой внешней окружающей среде. В некоторых случаях, обе популяции особей могут эволюционировать до того уровня эффективности, который равен максимальной возможной абсолютной эффективности. Таким образом, алгоритм иерархической совместной эволюции представляет собой самоорганизующийся обоюдно-развивающий процесс, движущей силой которого является относительная, а не абсолютная эффективность особей.

2.3.2.3. Адаптивные инкрементальные оценочные функции

Предложенный в работе [33] подход является результатом объединения идей из работ [32, 39]. Использование адаптивной инкрементальной оценочной функции подразумевает тот факт, что популяция организмов эволюционирует поэтапно. На каждом этапе эволюции к оценочной функции добавляется новый компонент. Перед началом каждого этапа эвристическим методом определяется эффективность применения для каждого неиспользуемого компонента оценочной функции. Наиболее эффективный компонент добавляется в текущую оценочную функцию и используется на данном этапе эволюции. В результате эволюция сначала идет с использованием наиболее эффективных компонентов оценочной функции, оставляя менее эффективные компоненты на более поздние этапы работы генетического алгоритма.

Для использования адаптивной инкрементальной оценочной функции необходимо переопределить оценочную функцию, по сравнению с тем определением, которое дано в работе [39]:

- пусть оценочная функция F имеет следующий вид $F = \sum_k F_i$, где $\{F_i\}$ – множество оценочных функций, связанных соответственно с множеством свойств $\{P_i\}$, которые совместно определяют задачу;
- пусть каждая оценочная функция F_i нормализована так, что особь c будет решением, тогда и только тогда, если $F_i(c) = 1$, для всех i;
- введем коэффициенты $\{\alpha_i\}$, где $\alpha_i > 0$ для любого i и $\sum_k \alpha_i = 1$;
- определим $F' = \sum_k \alpha_i F_i$.

В дополнение для упрощения описания инкрементальной природы предложенного процесса введем следующие множества и переопределим F и F' через них:

- Φ – множество, состоящее из всех оценочных функций, пронумерованных с помощью индекса i;
- ϕ – множество индексов оценочных функций, использующихся на текущем этапе эволюционного процесса (оно в начале $\varnothing$);
- $F(c) = \sum_{i\in\Phi} F_i$;
- $F'(c) = \sum_{i\in\phi} \alpha_i F_i$.

Приведем описание генетического алгоритма, использующего адаптивную инкрементальную оценочную функцию:

1. Инициализируется популяция особей p.
2. Для всех особе $c \in p$ вычисляется $F(c)$.
3. Выбирается лучшая особь $hero$ таким образом, что $F(hero) = \max_{c\in p} F(c)$.
4. $\phi \Leftarrow \varnothing$.
5. До тех пор, пока $F(hero) < 1$, выполняется следующий набор действий:
   - Для всех $i \in \Phi$, вычисляется $\overline{F_i}(p)$ (здесь и далее под значением $\overline{F_i}(p)$ понимается среднее значение, которое принимает оценочная функция на текущей популяции особей p).
   - Если $\phi \neq \Phi$, то выбираем j такое, что $\overline{F_j}(p) = \min_{i\in\Phi-\phi} \overline{F_i}(p)$ и добавляем j в ϕ ($\phi \Leftarrow \phi \cup j$).
   - Для того чтобы настроить множество коэффициентов $\{\alpha_i\}$ согласно эффективности компонент оценочных функций для каждого $i \in \phi$ определяем следующим образом:

$$\alpha_i \Leftarrow \frac{\frac{1}{\overline{F_i}(p)}}{\sum_{i\in\phi} \frac{1}{\overline{F_i}(p)}}.$$

   - Для всех особе $c \in p$ вычисляется $F'(c)$.
   - Выбирается лучшая особь $subhero$ таким образом, что $F'(subhero) = \max_{c\in p} F'(c)$.
   - До тех пор пока $F'(subhero) < 1$, популяцию p эволюционирует, используя в качестве оценочной функции F'.
   - Если $F(subhero) > F(hero)$, то $hero \Leftarrow subhero$.

В качестве эвристической оценки значимости компонента оценочной функции используется величина обратная среднему значению компонента оценочной функции на текущей популяции решений $\frac{1}{\overline{F_i(p)}}$. Другими словами, чем меньше среднее значение компонента оценочной функции, тем хуже особи в популяции ей удовлетворяют. На каждом этапе эволюции производится перенастройка множество коэффициентов $\{\alpha_i\}$, так чтобы увеличить вклад наиболее значимых компонентов оценочных функций.

К сожалению, подход, основанный на адаптивных инкрементальных оценочных функциях, может быть применен только к узкому классу задач.

2.4. Методы выбора оценочных функций

2.4.1. Выбор оценочной функции для задачи о флибах

В книге [24] рассматривается подход к проблеме автоматического построения конечных автоматов с помощью эволюционных методов. Первая задача, рассматриваемая в [24] – задача предсказания. На вход конечного автомата (флиба [4]) подается символ, соответствующий текущему состоянию окружающей среды. Задача флиба предсказать состояние среды в следующий момент времени. Для того, что бы определить, на сколько корректно автомат предсказывает изменения среды, ему на вход подается последовательность входных символов, соответствующих состояниям окружающей среды. Каждый выходной символ, генерируемый флибом, сравнивается со следующим входным символом, поступающим из среды. Процент правильных предсказаний есть мера способности автомата предсказывать поведение среды на основе предшествующих символов.

Рассмотрим, как работает оценочная функция на примере автомата M_0, диаграмма переходов для которого изображена на рис. 66.

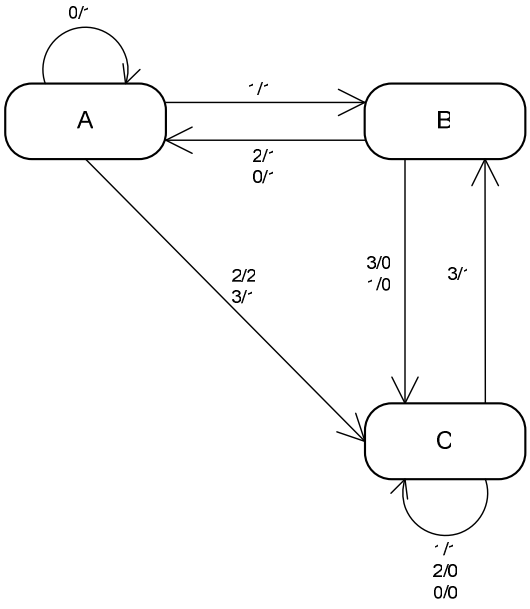


Рис. 66. Диаграмма переходов автомата

Пример обработки автоматом последовательности символов приведен в табл. 26.

Таблица 26. Результат работы автомата

Текущее состояние	B	A	C	C	C	C	B	C	C	B	A
Входной символ	2	2	1	0	1	3	3	0	3	0	1
Выходной символ		1	2	1	0	1	1	0	0	1	1
Стоимость ошибки		1	1	1	1	1	1	0	1	1	0

В первой строке этой таблицы показана последовательность текущих состояний флиба в процессе его реагирования на последовательность входных символов, представленную во второй строке таблицы. Состояние *B* было произвольно выбрано в качестве начального состояния. В третьей строке приведена последовательность символов, полученных в результате работы флиба. Необходимо отметить, что эти символы сдвинуты на одну позицию вправо для облегчения сравнения. Значения выходных символов в этой строке таблицы соответствуют предсказанным состояниям среды.

Последняя строка образована путем сопоставления фактического и предсказанного символов при учете «матрицы стоимостей» [24] (или «матрицы штрафов») $A = \{a_{ij}\}$. В данном примере $a_{ij} = 0$, если $i = j$, и $a_{ij} = 1$, если $i \neq j$. Стоимость каждого преобразования находится по матрице стоимостей, где индекс i относится к предсказанному символу (значению выходной переменной, генерируемой флибом), а j – к очередному символу входной последовательности. Оценочная функция F_1 (мера предсказательной способности машины) определяется как отношение суммы стоимостей всех ошибок к числу символов входной последовательности. Так средняя стоимость ошибок автомата M_0 находится по формуле $F_1 = \dfrac{Сумма\_стоимостей\_ошибок}{Длина\_входной\_последовательности} = \dfrac{8}{10} = 0.8$.

Как уже было отмечено выше, оценочная функция строится на основе матрицы стоимостей, приписанных каждому возможному правильному и ошибочному предсказанию следующего состояния среды. Например, если целью является минимизация величины ошибки предсказания каждого следующего символа, элементы матрицы стоимостей будут принимать значения $a_{ij} = |i - j|$. Если требуется минимизировать среднеквадратичную ошибку предсказания, элементы матрицы стоимостей будут иметь вид $a_{ij} = (i - j)^2$. Если требуется «точное попадание», все диагональные элементы в матрице ошибок будут равны нулю, а не диагональные элементы будут одинаковы. Такой же подход используется и в рассматриваемом выше примере. Если правильное предсказание некоторого символа имеет большую значимость, это может быть выражено в матрице стоимостей, выражающей цель предсказания. Так матрица стоимостей в табл. 27 указывает, что более важно правильное предсказание символа 0, менее важно правильное предсказание символа 1 и, наконец, еще менее важно правильное предсказание символа 2. В то же время из нее следует, что предпочтительнее ошибиться, приняв нуль за единицу, чем наоборот. Такой способ задания цели отражает относительный вес каждого возможного исхода. В рассмотренном примере не взимается дополнительный штраф за столь сложную цель. По мере сравнения символов на выход автомата с очередными входными символами, фактически появившимися в окружающей среде флиба, по матрице цели определяется стоимость соответствующей ошибки. Сумма соответствующих стоимостей характеризует ценность автомат с точки зрения его способности предсказывать только что предъявленную последовательность символов.

Таблица 27. Пример матрицы стоимостей

		Предсказанные символы		
		0	1	2
Фактические символы	0	0	3	4
	1	4	1	4
	2	4	4	2

В общем случае последовательность возникших в процессе эволюции автоматов будет зависеть от цели, заданной с помощью оценочной функции. Нет необходимости ограничивать эволюцию одной лишь неизменной целью. Если цель будет изменяться по мере протекания процесса в реальном времени, то новые поколения автоматов будут постепенно отражать эти изменения. Конечно, все сказанное выше об оценочных функциях построенных на основе матрицы штрафов, в равной степени применимо и к матрице платежей.

Оценочная функция может быть еще более гибкой. Например, если требуется предсказывать каждый второй символ, то для этого достаточно сравнивать значения выходной переменной флиба с символами среды со сдвигом на два такта. Небольшое изменение оценочной функции позволяет использовать этот подход для предсказания многомерных сред. Символы, описывающие изменяемые параметры среды, могут быть скомбинированы в новый алфавит символов, описывающий среду однозначным образом. Результаты предсказаний получаются в том же алфавите, так что они могут быть интерпретированы в виде индивидуальных предсказаний по каждому символу в отдельности.

До сих пор предполагалось, что каждый автомат следует оценивать по всей его доступной предыстории. Очевидно, что такая стратегия не всегда является наилучшей. Например, если среда изменяет свои статистические свойства, может оказаться более целесообразным ограничиться использованием, некоторых последовательностей, для того чтобы повысить вероятность нахождения наилучшей логики для настоящего времени. В случае полного отсутствия информации о характере среды выбор конкретной длины последовательности может оказаться весьма затруднительным. Можно использовать эвристическую процедуру, которая будет последовательно изменять длину последовательности, учитываемую оценочной функцией. В каждый момент времени две (или более) популяции автоматов эволюционируют на различной длине предыстории. Затем их показатели сравниваются. Это дает возможность определить, какие следующие длины предысторий подлежат рассмотрению. Полную длину последовательности следует запоминать как можно дольше, для того чтобы длина предыстории при желании могла быть расширена. Трудность этой задачи возрастает вместе с ограничением, налагаемым на объем памяти автомата вплоть до того момента, когда единственной альтернативой становится метод скользящего интервала – сохранение фиксированной длины предыстории.

В книги [24] описывается эксперимент, связанный с влиянием длины начального запоминания на качество предсказания. На протяжении начальной опытной последовательности поведение предсказывающей системы почти случайное. Ошибки, допущенные в этой последовательности, неизбежно уменьшают величину показателя качества. Этот эффект может быть устранен, если первое предсказание делается после испытания достаточно большого числа символов. На рис. 67 приведены сравнительные характеристики предсказания для автоматов с начальным запоминанием двух и 20-и символов.

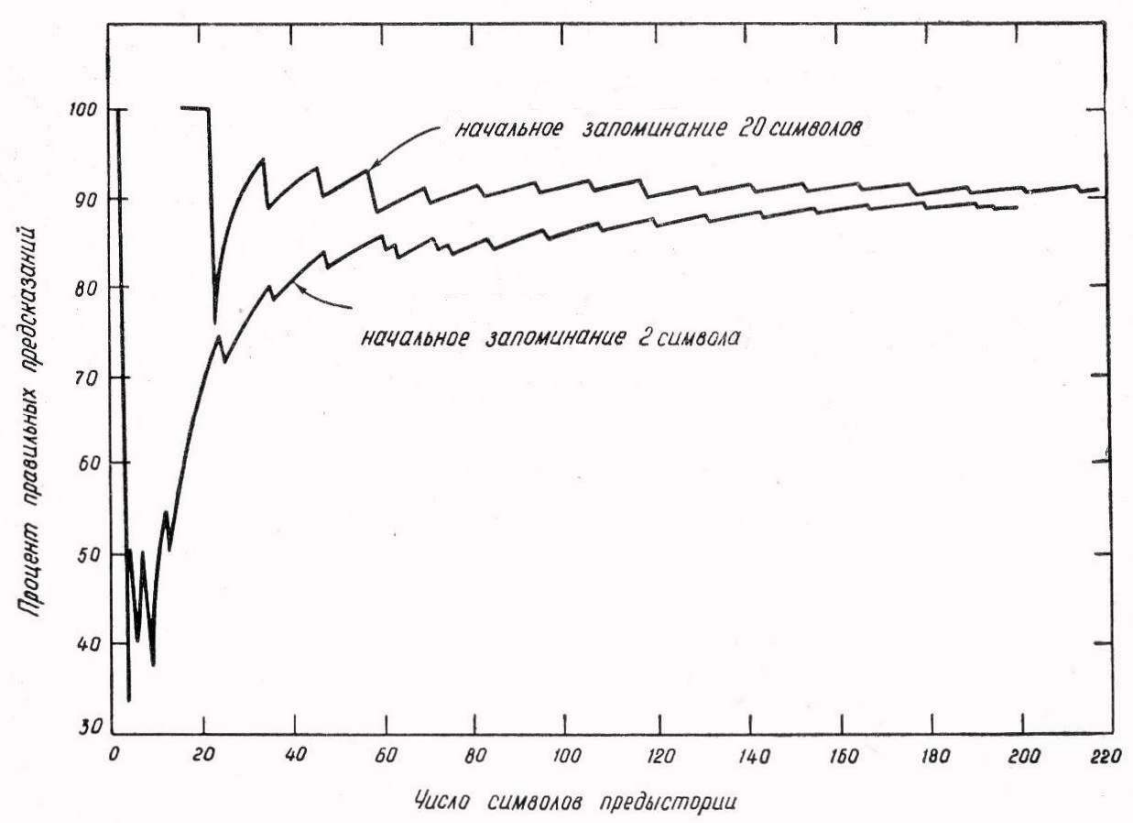


Рис. 67. Сравнение результатов предсказаний при различной длине предыстории

Эксперимент проводились при следующих условиях: циклическая среда вида 101110011101; три автомата на поколение; однократная мутация; случайный ряд единиц; пять поколений на предсказание.

В работе [24] показано как строить различные оценочные функция для задачи предсказания изменений характеристик окружающей среды. Оценочная функция изменятся с учетом целей, которые ставит перед собой исследователь. Использование специальных модификаций оценочных функций позволяет ускорить построение искомого автомата.

2.4.2. Использование совместной эволюции для построения минимаксных стратегий для пары игроков

В работе [36] совместная эволюция применяется для построения минимаксных стратегий для пары игроков, играющих в игру на двоих, правила которой заданы с помощью дерева игры, изображенного на рис. 68.

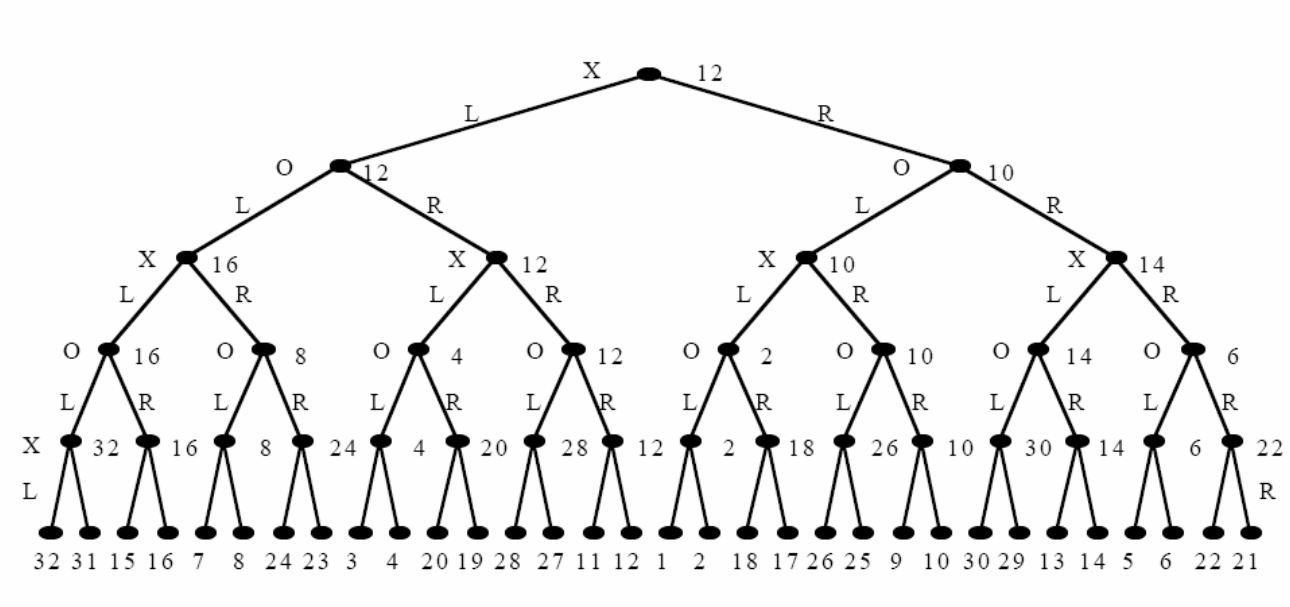


Рис. 68. Дерево игры с выплатами

В случае алгоритма совместной эволюции нет идеального противника, на котором можно было бы тренировать популяцию особей. Вместо него используются две совместно эволюционирующих популяции. Перед началом эволюции обе популяции заполняются особями, сгенерированными с помощью случайных композиций элементов из набора допустимых функций и терминалов.

Рассмотрим простую дискретную игру, дерево игры для которой изображено на рис. 68. Рядом с каждым узлом дерева нанесена метка, соответствующая игроку, который должен сделать ход. Каждая ветвь дерева помечена вариантом выбора (`L` или `R`), который может сделать игрок. Каждому листу дерева присвоена некоторая сумма вознаграждения (для игрока *X*).

Это игра соревнование для двоих, в которой игроки при каждом ходе принимают одно из двух возможных альтернативных решений. При каждом ходе игрок выбирает куда идти: `L` – налево или `R` – направо. После того как игрок *X* делает три хода и игрок *O* делает два хода, игрок *X* получает (а игрок *O* отдает) вознаграждение, соответствующее текущему листу дерева игры.

Каждый игрок имеет доступ к полной информации о предыдущих ходах его противника и о своих собственных ходах. История игры содержится в пяти переменных: `XM1` – первый ход игрока *X*, `OM1` – первый ход игрока *O*, `XM2` – второй ход игрока *X*, `OM2` – второй ход игрока *O* и `XM3` – третий ход игрока X. Каждая из этих пяти переменных может принимать одно из трех возможных значений: `L` – налево, `R` – направо или `U` – не определенно. Значения переменной перестает быть неопределенным, после того как один из игроков делает ход, связанный с ней. В начале значения всех пяти переменных не определены. Список переменных, значения которых не определены, задает состояние, в котором находится на текущий момент процесс игры. Например, если оба игрока сделали по одному ходу, переменные `XM1` и `OM1` определены (принимают значение или `L` или `R`), но значения трех оставшихся переменных (`XM2`, `OM2` и `XM3`) не определены (принимают значение `U`).

Стратегия для конкретного игрока определяет, какой выбор он сделает в каждой из возможных для него ситуаций. В частности, стратегия для игрока X должна определять его первый ход, если он находится в начале игры. Стратегия для игрока X также должна определять его второй ход, если игрок *O* сделал один ход. Она должна определять его третий ход, если игрок *O* уже сделал два хода. Так как игрок X ходит первым, его первый ход не зависит от предыдущих ходов. Однако второй ход игрока *X* будет зависеть от первого хода игрока *O* (от переменной `OM1`) и от выбора который игрок *X*

сделал при своем первом ходе (от переменной XM1). Таким же образом третий ход игрока X будет зависеть от первых двух ходов игрока *O* и от его собственных двух первых ходов. Аналогично сказанному выше, стратегия для игрока *O* должна определять выбор для игрока *O* во всех возможных ситуациях, в которые он может попасть. Стратегии в работе [36] заданы в виде компьютерных программ (S-выражений на языке *LISP*) на вход которых подаются значения переменных, содержащих историю игры. Значение выходной переменной, генерируемое программой, определяющей стратегию игрока, соответствует выбору (L или R), который игрок делает при соответствующем ходе. Таким образом, множество терминалов состоит из следующих элементов T = {L, R}.

Четыре проверочные функции CXM1, COM1, CXM2 и CMO2 позволяют делать выбор в зависимости от того, какие значения принимают переменные, содержащие историю игры. Каждая из этих функций определенный вариант *CASE*-функции языка *LISP*. Например, функция CMX1 принимает на вход три аргумента. Первый аргумент вычисляется, если переменная XM1 принимает значение U (не определено). Второй аргумент вычисляется, если переменная XM1 принимает значение L (налево) и третий аргумент, если переменная XM1 принимает значение R (направо). Функции COM1, CXM2 и CMO2 построены аналогично. Таким образом, множество функций состоит из следующих элементов F = {CXM1, COM1, CXM2, CMO2}. Каждая из этих функций принимает на вход три аргумента.

Задача состоит в построении стратегий для обоих игроков этой игры с помощью генетического алгоритма, основанного на совместной эволюции.

Оценочная функция, определяющая эффективность стратегии игрока, базируется на среднем вознаграждении, которое получает игрок в результате состязаний со всеми игроками из популяции соперников.

Абсолютная эффективность конкретной стратегии для конкретного игрока определяется, когда он играет против минимаксной стратегии противника. Заметим, что для вычисления абсолютной эффективности стратегии для игрока *X*, необходимо проверить ее для четырех возможных комбинаций ходов игрока *O* (игрок *O* делает выбор L или R на первом и втором ходах). Для вычисления абсолютной эффективности стратегии для игрока *O*, необходимо проверить ее для восьми возможных комбинаций ходов игрока *X* (этот игрок делает выбор L или R на первом, втором и третьем ходах). Необходимо отметить, что проверка четырех или восьми комбинаций ходов противников не делается при вычислении относительной эффективности стратегии. Для двух минимаксных стратегии противники играют друг против друга. При этом размер вознаграждения равен 12. Минимаксные стратегии имеют преимущество перед не минимаксными стратегиями других игроков.

Как упоминалось выше, в алгоритме совместной эволюции никак не используется минимаксная стратегия противника. Алгоритм совместной эволюции использует только относительную эффективность стратегий.

Эксперимент проводился при 300 особях в популяции. Программа, соответствующая лучшей стратегии для игрока *X* в первом поколении, имела следующий вид:

```
(COM1 L
      (COM2  (CXM1  (CXM2   R
                            (CXM2  R R R)
                            (CXM2  R L R))
                  L
                  (CXM2  L R  (COM2  R R R)))
```

```
(COM1 R
      (COM2 (CXM2 L R L) (COM2 R L L) R)
      (COM2 (COM1 R R L)
            (CXM1 R L R)
            (CXM1 R L L)))
      (CXM1 (COM2 (CXM1 R L L) (CXM2 R R L) R)
            R
            (COM2 L R (CXM1 L L L))))
R).
```

После упрощения эта программа принимает вид

```
(COM1 L (COM2 L L R) R).
```

Значение оценочной функции для этой особи равно 10.08.

Программа, соответствующая лучшей стратегии для игрока О, так же была сложной. После упрощения она приняла следующей вид

```
(CXM2 R (CXM1 # L R) (CXM1 # R L)).
```

Следует отметить, что при упрощении программы был использован символ # для обозначения тех ситуаций, которые никогда не могли возникнуть. Относительная эффективность этой стратегии 7.57.

Ни лучшая стратегия для игрока *X*, ни лучшая стратегия для игрока О из первого поколения не достигли максимальной эффективности.

Необходимо отметить, что значения оценочной функции для лучшей стратегии для игрока *X* и для лучшей стратегии для игрока *O* из первого поколения (10.08 и 7.57, соответственно) вычислялись, как усредненное вознаграждение, полученное в результате состязаний с трехстами особями из популяции противников.

Во втором поколении значение оценочной функции для лучшей стратегии для игрока X стало равным 11.28. Эта стратегия не являлась минимаксной стратегией. Максимальное значение абсолютной эффективности не было достигнуто.

Во втором поколении значение оценочной функции для лучшей стратегии для игрока О стало равным 7.18. Программа соответствующая этой стратегии представлена ниже:

```
(CXM2 (CXM1 R R L) (CXM2 L L (CXM2 R L R)) R).
```

Эта лучшая стратегия для игрока *O* во втором поколении и есть минимаксная стратегия для игрока *O*. Если бы эта стратегия состязалась с минимаксной стратегией для игрока X, ее вознаграждение было бы равным 12.

Лучшую стратегию для игрока О можно упростить до вида

```
(CXM2 (CXM1 # R L) L R).
```

В поколениях между вторым и пятнадцатым число особей в *O*-популяции, достигших абсолютной эффективности было 2, 7, 17, 28, 35, 40, 50, 64, 73, 83, 93, 98 и 107, соответственно. Так программы, реализующие минимаксную стратегию для игрока *O*, начали доминировать в популяции.

В пятнадцатом поколении значение оценочной функции для лучшей стратегии для игрока *X* стало равным 18.11. Программа, соответствующая этой стратегии, приведена ниже:

```
(COM2 (COM1 L L (CXM1 R R R))
      L
      (CXM1 (COM1 L L (CXM1 R R R))
            (CXM2 L R R)
            R)).
```

Эта лучшая стратегия для игрока *X* в тринадцатом поколении и есть минимаксная стратегия для игрока *X*. Если бы эта стратегия состязалась с минимаксной стратегией для игрока *O*, то ее вознаграждение было бы равным 12.

Лучшую стратегию для игрока *X* можно упростить до вида

```
(COM2 (COM1 L L R) L R).
```

В поколениях между 16 и 31, число особей в *X* популяции, достигших абсолютной эффективности, было 3, 4, 8, 11, 10, 9, 13, 21, 24, 29, 43, 32, 52, 48 и 50, соответственно. Так программы, реализующие минимаксную стратегию для игрока *X*, начали доминировать в популяции. Тем временем программы, реализующие минимаксную стратегию для игрока *O*, начали еще сильнее доминировать в *O*-популяции.

В 39-ом поколении число особей в *O*-популяции, достигших абсолютной эффективности стало равным 188 (почти две трети популяции), а особей в *O*-популяции, достигших абсолютной эффективности, стало равным 74 (примерно четверть). В 39-ом поколении минимаксные стратегии в поколения X стратегий для обоих игроков стали доминирующими.

Интересно, что значение оценочной функции для 74 стратегий для игрока X было равным 19.11, а для 188 стратегий для игрока O – 10.47. Ни одно из этих значений не равно 12, так как популяции не полностью заполнены минимаксными стратегиями.

Необходимо отметить, что генетический алгоритм, основанный на совместной эволюции, смог построить минимаксные стратегии для обоих игроков, несмотря на то, что изначально у него не было информации о минимаксной стратегии ни для одного из противников.

2.4.3. Использование инкрементальной оценочной функции для генерации конечных автоматов, распознающих регулярные выражения

Рассмотрим пример использования инкрементальных оценочных функция для построения конечных автоматов, распознающих регулярные выражения [32].

Конечный автомат читает входную строку из регулярного выражения и принимает или отвергает ее. Регулярные выражения создаются по следующим правилам:

1. Конечный алфавит.
2. Объединение, пересечение или конкатенации двух регулярных выражение – регулярное выражение.
3. Дополнение регулярного выражения так же является регулярным выражением.
4. Повторяемость регулярных выражений, так же называемая замыканием Клини.
5. Пустая строка тоже регулярное выражение.

Благодаря конечному алфавиту претендентов на решения легко проверить. Так как результатом пересечения, объединения или конкатенации двух регулярных выражений является регулярное выражение, то выведение новых регулярных выражений не вызывает сложностей.

Стандартный конечный автомат, распознающий регулярные выражения имеет только два возможных значений выходной переменной: «принять» и «не принять». Поэтому возникают сложности при сравнении эффективности двух конечных автоматов. Можно только определить корректно работает конечный автомат или нет, но невозможно понять, на сколько близко его

поведение к желаемому. Для решения этой проблемы в работе [32] предлагается два способа для решения этой проблемы. Первый способ заключается в увеличении числа моментов времени, в которых происходит считывания значения выходной переменой, генерируемой эволюционирующим конечным автоматом. Выходная переменная проверяется после чтения автоматом каждого входного символа. Предполагается, что каждый входной символ может быть последним символом строки. Второй способ заключается в разбиения значения выходной переменной «не принять» на два отдельных значения: «отклонить» и «не известно». Выходная переменная может принимать значение «не известно», пока процесс разбора не завершится. Если входной символ приводит к ошибке в разборе, то выходная переменная принимает значение «отклонить». Устройство остается в состоянии отклонить, так как поступающие на вход символы не могут изменить того факта, что строка не удовлетворяет регулярному выражению.

Пример разбора строки помощью такого метода показан на рис. 69.

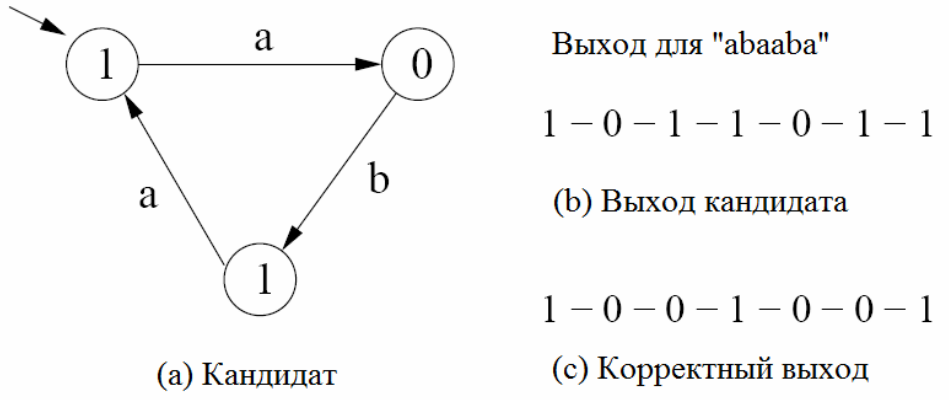


Рис. 69. Выход конечного автомата при разборе строки “abaaba”

На рис. 69 изображен один из кандидатов на распознавания строк, соответствующих регулярному выражению “(aba)*”. Все значения выходной переменной, равные «отклонить», удалены для упрощения примера. Заметим, что кандидат выдал корректное значение выходной переменной ‘1’ («принять») в первом состоянии (пустая строка удовлетворяет рассматриваемому в примере регулярному выражению). На вход кандидату подавалась допустимая строка “abaaba”. На рис. 69 приведен список выходных значений, выданных кандидатом, и корректный список значений, для автомата, умеющего корректно распознавать строки, соответствующие регулярному выражению.

Предложенный в работе [32] алгоритм использует пять независимых операторов мутации: добавить состояние, удалить стояние, изменить начальное состояние, изменить переход между состояниями, изменить значение выходной переменной. Оператор скрещивания, используемый в работе [32], создает новый конечный автомат, комбинируя части, вырезанные из двух родительских автоматов. Благодаря оператору скрещивания новый автомат наследует часть характеристик от каждого из своих родителей.

В работе [32] описывается ряд экспериментов, проводимых для сравнения эффективности инкрементального генетического алгоритма с обычным генетическим алгоритмом. Для каждого набора начальных данных производится серия испытаний. В большинстве испытаний удается построить автомат, удовлетворяющий условию задачи. Случай “aab(ba)*bba(abjba)bb” слишком сложен для обычного генетического алгоритма, в то время как, итеративный алгоритм справляется с поиском правильного решения. Размер популяции во все экспериментах равен 300 особям. При формировании нового поколения треть особей выбирается из текущего поколения без изменений, треть особей формируется с помощью оператора скрещивания, а еще треть формируется с

помощью оператора мутации. Проверка эффективности генетического алгоритма требует большого числа тестов. В связи с этим число тестов выбрано равным 500. Максимальное число поколений ограничено десятью тысячами. После того как алгоритм находит корректное решение, эволюция продолжается в течение еще 50 поколений, для того чтобы уменьшить размер полученного конечного автомат.

Первая группа экспериментов связана с построением автоматов для регулярных выражений, заданных строками различной длины. В целях сравнения далее приводятся усредненные результаты из всех тестовых групп. В первом случае требовалось построить автомат для регулярного выражения заданного строкой "abba". Эксперименты проводились с различной длиной приращения символов, добавляемых на каждом этапе работы генетического алгоритма. Использовались длина приращения символов – ноль (обычный генетический алгоритм), длина приращения символов – единица ("a", "ab", "abb", "abba") и длина приращения символов – два ("ab", "abba"). Результаты этих экспериментов приведены на рис. 70. На графике по вертикальной оси отложено среднее число поколений, необходимое для построения искомого автомата в каждом случае. Каждый прямоугольник на графике соответствует группе экспериментов с фиксированной длиной приращения символов. Прямоугольник "0" соответствует стандартному генетическому алгоритму. Остальные прямоугольники соответствуют инкрементальным генетическим алгоритмам. Использование инкрементальной оценочной функции уменьшило число поколений, требующееся для построения автомата на 16%.

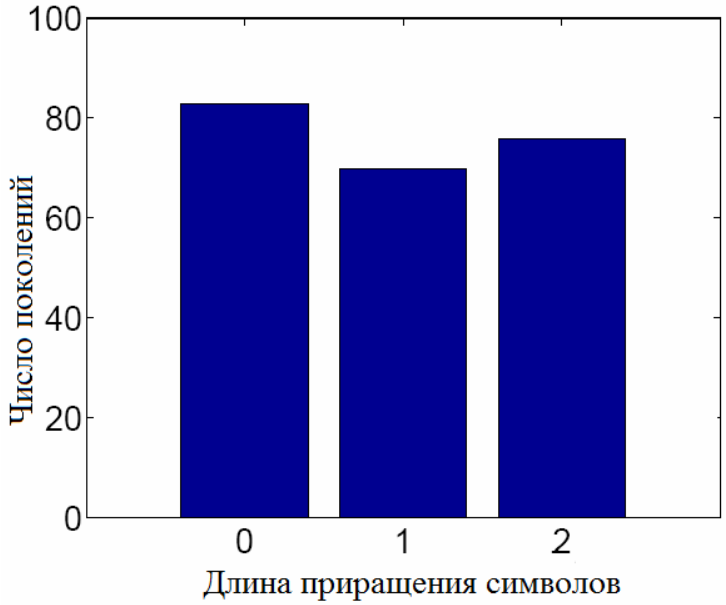


Рис. 70. Построение автомата для строки из четырех символов

Эффект от использования инкрементальных оценочных функций особенно хорошо заметен при построении автоматов для более длинных строк. На рис. 71 и рис. 72 приведены результаты экспериментов для строк длиной 6 и 12 символов соответственно. Использование инкрементальной оценочной функции с длиной приращения символов равной трем в эксперименте по построению автомата, соответствующего регулярному выражению, заданному строкой из шести символов, позволило уменьшить число поколений по сравнению со стандартным генетическим алгоритмом на 80%.

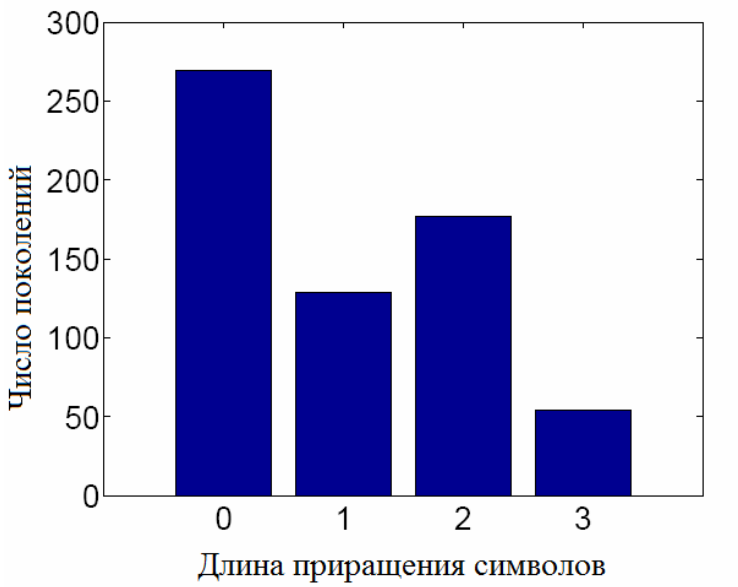


Рис. 71. Построение автомата для строки из шести символов

Использование инкрементального генетического алгоритма с длиной приращения символов, равной единице, в эксперименте по построению автомата, соответствующего регулярному выражению, заданному строкой из двенадцати символов, позволило сократить число поколений на 50%. Увеличение скорости сходимости при длине приращения символов равной трем, можно объяснить при рассмотрении строк, с помощью которых задавались регулярные выражения. В этих экспериментах строились строки для регулярных выражений, заданных строками "abbabb" и "abbabbbababb". Можно заметить, что символы "abb" повторяются в этих строках. Как только автомат для регулярного выражения "abb" построен, он может использоваться как основа для дальнейшей эволюции. При наличии в поколении автомата для регулярного выражения заданного строкой "abb" генетический алгоритм быстро строит автомат для регулярного выражения, заданного строкой "abbabb".

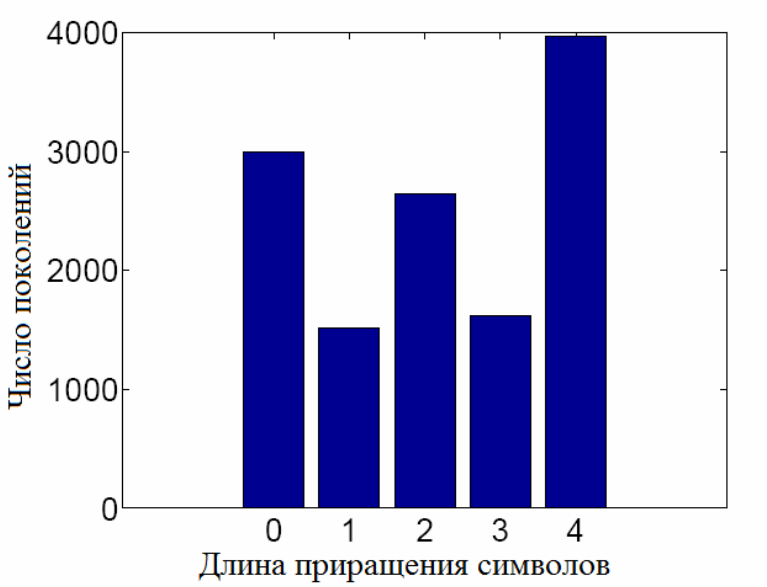


Рис. 72. Построение автомата для строки из двенадцати символов

В работе [32] также описываются две серии экспериментов для регулярных выражений заданных с помощью объединения строк, состоящих из трех символов. Первая серия экспериментов производилась для регулярного выражения заданного через объединение шести строк "aaa", "aab",..., "bab". Вторая серия экспериментов проводилась над объединением строк из первой серии экспериментов, к которому была добавлена строка "bba". На рис. 73 показаны результаты первой серии экспериментов. Отметим, что инкрементальный генетический алгоритм смог построить автомат при числе добавленных строк равном двум и трем. Инкрементальный эволюционный подход плохо работает при добавлении строк в объединение. Например, при построении автомата, принимающего только строки вида "aaa", на него накладывается ограничение отклонять строки вида "aab". После перехода к этапу, на котором требуются построить автомат для регулярного выражения "aaa|aab" из уже найденного решения необходимо удалить часть, отвечающую за отклонение строки вида "aab" и добавить новую функциональность.

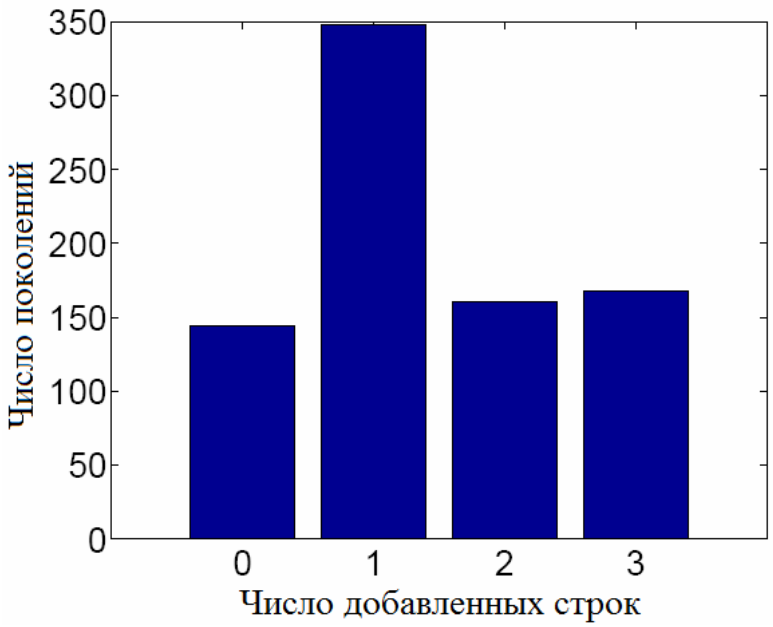


Рис. 73. Построение автомата для объединения из шести строк

Для получения автомата, который решает задачу во второй группе экспериментов, потребовалось значительно больше поколений. Отклонение только строк вида "bbb" требует значительно более сложного конечного автомата. Усложнение задачи существенно замедляет работу стандартного генетического алгоритма. На рис. 74 показаны усредненные результаты этой серии экспериментов. При числе добавленных строк на этап равном, двум, число поколений, необходимое для построения корректно работающего по сравнению со стандартным генетическим алгоритмом уменьшается на 88%.

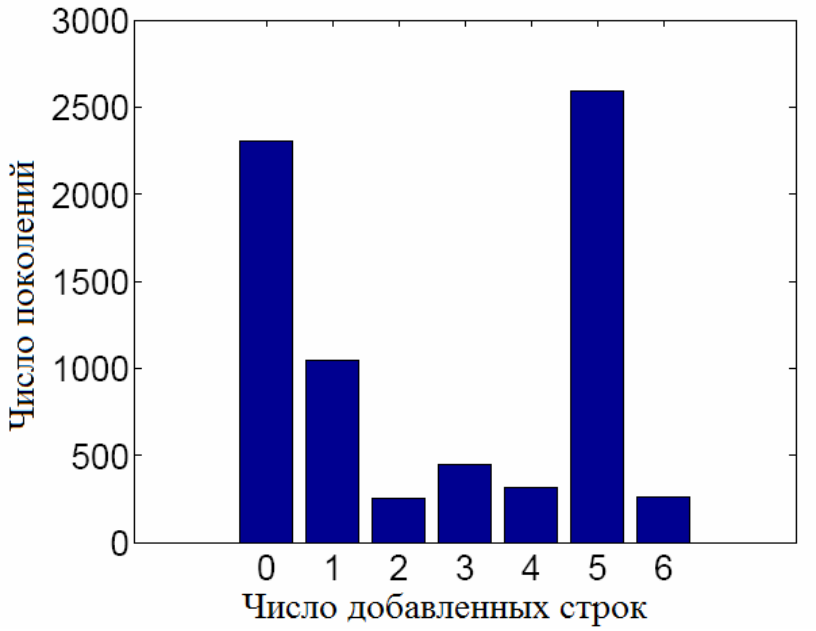


Рис. 74. Построение автомата для объединения из семи строк

Необходимо отметить, что в большинстве экспериментов генетические алгоритмы, в которых использовалась инкрементальные оценочные функции, показали существенно лучшие результаты по сравнению со стандартными алгоритмами.

2.5. Определение функциональных особенностей построения оценочных функций, применяемых при генерации автоматов

Эволюционные вычисления (включая генетические алгоритмы) применяются для нескольких типов исследований, например:

- изучение изменяющихся свойств популяции;
- решение вариационных задач (поиск оптимума).

Необходимо разделять эти типы исследований, так как оценочные функции используются в них для различных целей.

В работе [47] исследуются репродуктивные свойства популяции во время эволюции. В работе [28] изучают процессы коммуникации между особями. Эти работы относятся к первому типу исследований.

Приведем примеры работ, относящихся ко второму типу исследований. Цель работы [15] состоит в нахождении оптимальной стратегии поведения игрока, а в работе [25] генетический алгоритм непосредственно используется для решения вариационной задачи.

На особенности построения оценочных функций влияют также типы эволюции.

Совместная эволюция (коэволюция) эффективно применяется для решения нескольких типов задач, например, задач из теории игр. В них игроки почти никогда не имеют априорного доступа к минимаксной стратегии друг друга. Как правило, при попытке построить минимаксную стратегию для лучшего игрока, знание минимаксных стратегий более слабых игроков не дает каких-либо преимуществ. Например, в шахматах или шашках, новому игроку сложно научиться играть хорошо, если у него не было опыта игры с противником высокого уровня. Таким образом, совместная эволюция популяций и оценочная функция, основанная на относительной приспособленности особей, позволяют обойтись без знания стратегии лучшего игрока.

Другим примером успешного применения коэволюции является работа [35]. В ряде задач отсутствует возможность точного вычисления функции приспособленности. Поэтому, например, в задаче классификации плотности [35] требуется создать некоторый набор репрезентативных тестов таких, что значение приближенной функции приспособленности, вычисляемой на этих тестах, как можно более точно соответствовало бы ее реальному значению. Решения задачи в процессе эволюции постепенно адаптируются к выбранным тестам, и они перестают быть репрезентативными – значение приближенной функция становится бо́льшим, чем значение реальной.

Для того чтобы избежать указанного эффекта, автор идеи коэволюции (*D. Hillis*) в 80-х годах прошлого века в работе [29] предложил наряду с популяцией решений создать эволюционирующую популяцию тестов. Функция приспособленности, определенная на множестве решений, прямо пропорциональна числу тестов, которые "проходит" это решение, а функция приспособленности на множестве тестов прямо пропорционально числу решений, которые не проходят данный тест. Это позволит тестам постоянно совершенствоваться, становясь все сложнее.

Рассмотрим также способ определения оценочной функции, использованный в работе [13]. Его особенностью является то, что построенная оценочная функция отличается от целевой функции задачи, предназначенной для оценки окончательного решения. Однако данное различие “мало” – особи, имеющие высокую приспособленность, являются также и хорошими решениями задачи в смысле целевой функции. При этом построенная функция приспособленности значительно проще и быстрее вычисляется, по сравнению с целевой функцией задачи. В работе [13] для построения оценочной функции используется ряд “хороших” эвристик, обеспечивающих указанное свойство

схожести оптимальных решений в смысле оценочной функции генетического алгоритма и целевой функции задачи.

Рассмотрим еще один тип эволюции. В работе [32] *инкрементальная (поэтапная) эволюция* используется для построения конечного автомата распознающего определенные регулярные выражения. Для каждого автомата из популяции значение оценочной функции вычисляется с помощью его тестирования на наборе строк (автомат отвергает или принимает строки) и сравнения полученных и правильных результатов. Процесс начинается с набора строк единичной длины. На каждом этапе эволюции в тестовый набор добавляются строки, длина которых на единицу больше. Интуитивно понятно, что такая формулировка алгоритма может быть легко изменена для использования адаптивных итерационных оценочных функций.

Для того чтобы понять, почему инкрементальный (поэтапный) эволюционный процесс эффективен, необходимо рассмотреть две важные стороны эволюционного процесса.

Рассмотрим свойства операторов скрещивания и мутации. В то время как оператор скрещивания вносит значительные изменения в структуру конечного автомата, изменения, вносимые в его структуру, оператором мутации достаточно невелики. Это означает, что оператор скрещивания позволяет значительно изменять параметры кандидатов в решения, в то время как оператор мутации отвечает за небольшие шаги в пространстве параметров. Оператор скрещивания позволяет перемещаться на большие расстояния в пространстве параметров и быстро находить решение задачи. Оператор мутации улучшает структуру решений с помощью небольших шагов в пространстве решений.

Популяция решений в процессе эволюции должна перемещаться в пространстве параметров по направлению к решению задачи. Возможно существование нескольких путей в пространстве параметров, по которым идет эволюционный процесс. Каждый путь определяется оценочной функцией. Особи популяции перемещаются в пространстве параметров по направлению к точке, в которой значение этой функции максимально. Если есть несколько путей по направлению к цели, то некоторые особи из популяции могут двигаться к ней разными путями. До тех пор пока значения оценочной функции на каждом из путей близки, эволюционный процесс будет идти по каждому из них.

Покажем, почему инкрементальная эволюция обладает указанными достоинствами. Особи популяция выстраиваются вдоль нескольких путей в пространстве решений. Во время применения оператора скрещивания новый конечный автомат формируется из родительских особей, выбранных случайным образом. Если оба родителя находятся на отличных (и удаленных друг от друга в пространстве параметров) путях, тогда дочерний автомат имеет высокие шансы не попасть на правильный путь. Значение оценочной функции для него может быть существенно ниже, чем у остальных особей популяции, и его шансы оставить потомство будут крайне низкими. Из-за этого эффективность использования оператора скрещивания снижается, и эволюция продолжается, в основном, за счет применения более медленного оператора мутации.

Метод, предложенный в работе [32], позволяет ускорить эволюционный процесс за счет использования небольших этапов (на каждом из этих этапов к оценочной функции добавляется дополнительная часть). При инкрементальной эволюции расстояния, на которые должна переместиться популяция в пространстве параметров значительно сокращаются. При этом на каждом этапе добавляется небольшая функциональность. Также понятно, что, так как только небольшая функциональность добавляется на каждом этапе, то новая цель в пространстве параметров не должна находиться далеко от цели, достигнутой на предыдущем этапе. Благодаря этому уменьшается вероятность получения дочерней особи, которая при использовании оператора скрещивания будет сильно хуже родительской, что, в свою очередь, увеличивает скорость эволюционного процесса.

Например, вместо того, чтобы строить сразу автомат для регулярного выражения, заданного строкой "aababa" в работе [32], можно сначала построить автомат, корректно обрабатывающий строку "aa". Когда в популяции появляется особь, которая корректно обрабатывает строку "aa", задача изменяется, и после этого генетический алгоритм пытается вывести особь, которая сможет корректно обрабатывать строку "aaba". Эволюция продолжается до тех пор, пока алгоритм не найдет решение. После этого цель изменяется еще раз, и теперь требуется вывести особь, которая сможет корректно обработать строку "aababa".

В заключение раздела отметим, что использование адаптивных итерационных оценочных функций [33] существенно сокращает класс задач, к которому могут быть применены генетические алгоритмы, построенные на их основе.

Из изложенного следует, что функциональные особенности построения оценочных функций определяются характером решаемой задачи, необходимостью поддержки совместной эволюции, а также возможностью аппроксимации целевой функции задачи простой оценочной функцией и целесообразностью применения инкрементальной эволюции.

ЗАКЛЮЧЕНИЕ

В результате исследований, выполненных на первом этапе работ по контракту, был проведен анализ предметной области и существующих методов генерации автоматов. При этом были сформированы требования, которым должны удовлетворять такие методы.

Рассмотрены базовые компоненты генетического программирования, предназначенные для генерации автоматов рассматриваемого класса. При этом были рассмотрены специфичные для предметной области структуры кодирования генетической информации (хромосомы) и даны рекомендации по их применению в зависимости от решаемой задачи. Также были рассмотрены методы построения оценочных функций, применяемых в рассматриваемых генераторах и методы их выбора.

По результатам первой главы были определены основные задачи исследования первого этапа:

- сравнение различных методов построения автоматов, анализ их достоинств и недостатков на ряде известных задач, для которых ранее подобного анализа не проводилось;
- разработка новых высокоуровневых способов представления автоматов в виде особей генетического алгоритма, обладающих б*о*льшей эффективностью, чем известные способы;
- рассмотрение генетических операторов, соответствующих разрабатываемым представлениям автоматов.

На основе исследований, выполненных во второй главе, установлено, что скорость и качество генерации автоматов на основе генетического программирования могут быть увеличены за счет соответствующего выбора структуры хромосом, методов построения оценочных функций, методов выбора оценочных функций, так же учета функциональных особенностей методов их построения.

При выполнении работы по первому этапу авторами были проанализированы все доступные источники в области применения генетического программирования для генерации автоматов, и на их основе были получены решения всех задач, поставленных в техническом задании на проведение этого этапа работы.

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

ИСТОЧНИКИ

1. *Воронин О., Дьюдни А.* Дарвинизм в программировании //Мой компьютер. 2004. № 35. http://www.mycomp.kiev.ua/text/7458
2. *Гач П., Курдюмов Г. Л., Левин Л. А.* Одномерные однородные среды, размывающие конечные острова // Проблемы передачи информации. 1976. 14, 3.
3. *Гладков Л. А., Курейчик В. В., Курейчик В. М.* Генетические алгоритмы. М.: Физматлит. 2006.
4. *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
5. *Непейвода Н. Н.* Стили и методы программирования. М.: Интернет-Университет Информационных технологий, 2005.
6. *Николенко С. И.* Лекции по генетическим алгоритмам. http://logic.pdmi.ras.ru/~sergey/teaching/ml/
7. *Полимеразная цепная реакция.* http://en.wikipedia.org/wiki/Polymerase_chain_reaction
8. *Туккель Н. И., Шалыто А. А.* Система управления дизель-генератором (фрагмент). Программирование с явным выделением состояний. Проектная документация. http://is.ifmo.ru/projects/dg/
9. *Хопкрофт Дж., Мотвани Р., Ульман Дж.* Введение в теорию автоматов, языков и вычислений. М.: Вильямс, 2002.
10. *Шалыто А. А.* Switch-технология. Алгоритмизация и программирование задач логического управления СПб.: Наука, 1998.
11. *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
12. *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
13. *Armstrong, D.* A programmed algorithm for assigning internal codes to sequential machines // IRETransactions on Electronic Computers, EC.11. 1962. I.4, pp.466–472.
14. *Axelrod R.* The Evolution of Cooperation. NY: Basic Books, 1984.
15. *Axelrod R.* The Evolution of Strategies in the Iterated Prisoner's Dilemma / In Genetic Algorithms and Simulated Annealing. London: Pitman, 1987. pp. 32−41. http://www-personal.umich.edu/~axe/research_papers.html
16. *Belz A., Eskikaya B.* A Genetic Algorithm for Finite State Automata Induction with Application to Phonotactics / Proceedings of the ESSLLI-98 Workshop on Automated Acquisition of Syntax and Parsing, Saarbruecken, 1998, pp. 9–17. http://www.itri.brighton.ac.uk/~Anja.Belz/Publications/A_GA_for_FSA_induction_with_an_application_to_phonotactics.ps
17. *Benson K.* Evolving Finite State Machines with Embedded Genetic Programming for Automatic Target Detection. http://ieeexplore.ieee.org/iel5/6997/18853/00870838.pdf
18. *Brave S.* Evolving Deterministic Finite Automata Using Cellular Encoding / Proceeding of First Annual Conference (Genetic Programming-1996), pp. 39−44. http://citeseer.ist.psu.edu/131538.html
19. *Chambers L.* Practical Handbook of Genetic Algorithms. Complex Coding Systems. Volumes I. II, III. CRC Press, 1999.

20. *Das R., Mitchell M., Crutchfield J.P.* A genetic algorithm discovers particle-based computation in cellular automata // Lecture Notes in Computer Science. 1994. www.santafe.edu/research/publications/workingpapers/94-03-015.pdf
21. *Ferreira C.* Discovery of the Boolean Functions to the Best Density Classification Rules Using Gene Expression Programming // Lecture Notes in Computer Science. 2002. Vol. 2278, pp. 51–60. www.gene-expression-programming.com/webpapers/ferreira-EuroGP02.pdf
22. *Fogel D.* The Evolution of Intelligent Decision Making in Gaming // Cybernetics and Systems. 2001. 22, pp. 223−236.
23. *Fogel L.* Autonomous Automata // Industrial Research. 1962. V.4, pp. 14–19.
24. *Fogel L., Owens A., Walsh M.* Artificial Intelligence through Simulated Evolution. NY: Wiley. 1966.
25. *Frey C., Leugering G.* Evolving Strategies for Global Optimization – A Finite State Machine Approach /Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001). Morgan Kaufmann. 2001, pp. 27–33. http://citeseer.ist.psu.edu/456373.html
26. *Ghnemat R., Khatatneh K., Oqeili S., Bertelle C., Duchamp G.* Automata-based adaptive behavior for economic modeling using game theory. 2005. http://arxiv.org/abs/cs/0510089
27. *Gold E. M.* Complexity of automaton identification from given data. // Information and Control, 1978, № 37, pp. 302−320.
28. *Goldberg D.* A Note on Boltzmann Tournament Selection for Genetic Algorithms and Population-Oriented Simulated Annealing // Complex Systems. 1990. V. 4, I. 4, pp. 445−460.
29. *Hillis W.* Co-Evolving Parasites Improve Simulated Evolution as an Optimization Procedure / In Artificial Life II. MA: Addison-Wesley, 1992. pp. 313−324.
30. *Holland J.* Adaptation in Natural and Artificial Systems. University of Michigan Press, 1975.
31. *Holland J.* ECHO: Explorations of Evolution in a Miniature World / Proceedings of the Second Conference on Artificial Life. Redwood City. CA: Addison-Wesley, 1990.
32. *Horihan J., Yung-Hsiang Lu.* Improving FSM evolution with progressive fitness functions / In GLSVLSI '04: Proceedings of the 14th ACM Great Lakes symposium on VLSI. NY: ACM Press. 2004. pp. 123−126. http://www.google.com/url?sa=t&ct=res&cd=1&url=https%3A%2F%2Fengineering.purdue.edu%2FResearchGroups%2FHELPS%2Fpapers%2Fjason_vlsi&ei=-0HZRsKVHZn8wwHr1oGkCQ&usg=AFQjCNFn2-moavfhbnw2eMgIuTWipFeOSg&sig2=1VtS0rjoRQITb74XYz_Img
33. *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
34. *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
35. *Juillie H., Pollack J.* Coevolving the "Ideal" Trainer: Application to the Discovery of Cellular Automata Rules. 1998. http://citeseer.ist.psu.edu/16712.html
36. *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
37. *Koza J.* Genetic programming. On the Programming of Computers by Means of Natural Selection. MA: The MIT Press, 1998.
38. *Levy S.* Artificial Life: The Quest for a New Creation. New York: Pantheon, 1992. 380 p.

39. *Linton R.* Adapting binary fitness functions in genetic algorithms / Proceedings of the 42nd annual Southeast regional conference. NY: ACM Press. 2004. pp. 391−395.
40. *Lucas M., Reynolds T.J.* Learning DFA: Evolution versus Evidence Driven State Merging.
41. *MacLennan B.* Synthetic Ethology: An Approach to the Study of Communication / Proceedings of Artificial Life II: The Second Workshop on the Synthesis and Simulation of Living. CA: Addison Wesley. 1992. V. X, pp. 631–658. http://www.cs.utk.edu/~mclennan/papers/SEASC.pdf
42. *Miller J.* The Coevolution of Automata in the Repeated Prisoner's Dilemma. Working Paper. Santa Fe Institute. 1989 // Journal of Economic Behavior & Organization. 1996. V. 29, I. 1, pp. 87–112.
43. *Mitchell M.* An Introduction to Genetic Algorithms. MA: The MIT Press, 1996.
44. *Mitchell M., Crutchfield J. P., Hraber P. T.* Evolving cellular automata to perform computations. Physica D. 75, pp. 361–391, 1993. http://web.cecs.pdx.edu/~mm/mech-imped.pdf
45. *Naidoo A., Pillay N.* The Induction of Finite Transducers Using Genetic Programming / Proceedings of Euro GP. Springer. 2007. http://saturn.cs.unp.ac.za/~nelishiap/papers/eurogp07.pdf
46. *Nedjah N., Mourelle L.* Mealy Finite State Machines: An Evolutionary Approach // International Journal of Innovative Computing, Information and Control. 2006. V.2, I. 4.
47. *Ray T.* An Approach to the Synthesis of Life / In Artificial Life II. MA: Addison-Wesley, 1992. pp. 371−408.
48. *Reynolds C. W.* Competition, Coevolution and the Game of Tag / Proceedings of Artificial Life IV. Cambridge. MA: MIT Press, 1994. pp. 59−69. http://www.red3d.com/cwr/papers/1994/alife4.html
49. *Tu M., Wolff E., Lamersdorf W.* Genetic Algorithms for Automated Negotiations: A FSM-based Application Approach /Proceedings of the 11-th International Conference on Database and Expert Systems. 2000. http://ieeexplore.ieee.org/iel5/7035/18943/00875153.pdf
50. *Whitley D.* The GENITOR Algorithm and Selective Pressure / Proceedings of the 3rd International Conference on Genetic Algorithms. Colorado State: Morgan Kaufmann, 1989. pp. 116−121. http://www.genalgo.com/index.php?option=com_content&task=view&id=80&Itemid=30
51. *Wolff K., Nordin P.* An Evolutionary Based Approach for Control Programming of Humanoids / Proceedings of the 3rd International Conference on Humanoid Robots (Humanoids'03). Karlsruhe: VDI/VDE-GMA. 2003. http://citeseer.ist.psu.edu/641298.html
52. *Zomorodian A.* Context-free Language Induction by Evolution of Deterministic Push-Down Automata Using Genetic Programming. http://www.cs.dartmouth.edu/~afra/papers/aaai96/aaai96.pdf