Федеральное агентство по образованию

УДК 004.4’242
ГРНТИ 20.01.01, 28.23.29
Инв. № 390149-1

ПРИНЯТО:	УТВЕРЖДЕНО:
Приемочная комиссия Государственного заказчика:	**Государственный заказчик Федеральное агентство по образованию**
От имени Приемочной комиссии _____/Попова Е.П. /	**От имени Государственного заказчика _____/Бутко Е.Я./**

НАУЧНО-ТЕХНИЧЕСКИЙ ОТЧЕТ

о выполнении 1 этапа Государственного контракта
№ П1188 от 27 августа 2009 г.
с Дополнением №__ от 22 октября 2009 г.

Исполнитель: Государственное образовательное учреждение высшего профессионального образования "Санкт-Петербургский государственный университет информационных технологий, механики и оптики"
Программа (мероприятие): Федеральная целевая программ «Научные и научно-педагогические кадры инновационной России» на 2009-2013 гг., в рамках реализации мероприятия № 1.3.1 Проведение научных исследований молодыми учеными – кандидатами наук.
Проект: Разработка методов совместного применения генетического и автоматного программирования для построения систем управления беспилотными летательными аппаратами
Руководитель организации: Васильев Владимир Николаевич
Руководитель проекта: Гуров Вадим Сергеевич

Согласовано: Управление научных исследований и инновационных программ От имени Заказчика _____/Кошкин В.И./

Санкт-Петербург
2009 г.

СПИСОК ОСНОВНЫХ ИСПОЛНИТЕЛЕЙ

по Государственному контракту № П1188 от 27 августа 2009 г.
с Дополнением №__ от 22 октября 2009 г.
на выполнение поисковых научно-исследовательских работ для государственных нужд

Организация-Исполнитель: Государственное образовательное учреждение высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики»

Руководитель темы:

кандидат технических наук, доцент — Гуров В. С.
_____
подпись, дата

Исполнители темы:

кандидат технических наук, доцент — Корнеев Г. А.
_____
подпись, дата

без ученой степени, без ученого звания — Царев Ф. Н.
_____
подпись, дата

без ученой степени, без ученого звания — Данилов В. Р.
_____
подпись, дата

без ученой степени, без ученого звания — Клебан В. О.
_____
подпись, дата

без ученой степени, без ученого звания — Мандриков Е. А.
_____
подпись, дата

без ученой степени, без ученого звания — Кулев В. А.

_____
подпись, дата

без ученой степени, без ученого звания — Давыдов А. А.

_____
подпись, дата

без ученой степени, без ученого звания — Соколов Д. О.

_____
подпись, дата

без ученой степени, без ученого звания — Царев М. Н.

_____
подпись, дата

без ученой степени, без ученого звания — Чеботарева Ю. К.

_____
подпись, дата

РЕФЕРАТ

Отчет 111 с., 5 гл., 60 рис., 10 табл., 76 источников.

Ключевые слова: генетическое программирование; автоматное программирование; линейные бинарные графы; симуляторы летательных аппаратов.

В настоящем отчете излагаются результаты выполнения *первого этапа поисковых научно-исследовательских работ по направлению «Информатика» по проблеме «Разработка методов совместного применения генетического и автоматного программирования для построения систем управления беспилотными летательными аппаратами»*, выполняемых в рамках государственного контракта, заключенного между Федеральным агентством по образованию и государственным образовательным учреждением высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» в соответствии с решением *Единой комиссии (протокол от 07 августа 2009 г. № 3/НК-178П) по конкурсу № НК-178П* «Проведение поисковых научно-исследовательских работ по направлению «Информатика» в рамках мероприятия 1.3.1 Программы», выполняемому в рамках мероприятия 1.3.1 «Проведение научных исследований молодыми учеными – кандидатами наук» мероприятия 1.3 «Проведение научных исследований молодыми учеными – кандидатами наук и целевыми аспирантами в научно-образовательных центрах» направления 1 «Стимулирование закрепления молодежи в сфере науки, образования и высоких технологий» федеральной целевой программы «Научные и научно-педагогические кадры инновационной России» на 2009-2013 годы, утвержденной постановлением Правительства Российской Федерации от 28 июля 2008 года № 568 «О федеральной целевой программе «Научные и научно-педагогические кадры инновационной России» на 2009-2013 годы».

Целями настоящего этапа являются:

1. Выполнение аналитического обзора.
2. Выбор и обоснование оптимального варианта направления исследований.
3. Подготовка плана проведения теоретических и экспериментальных исследований.
4. Разработка и программная реализация метода генетического программирования, основанного на использовании линейных бинарных графов для представления функции переходов автоматов.
5. Экспериментальное исследование метода генетического программирования, основанного на использовании линейных бинарных графов для представления функции переходов автоматов, на примере построения автомата управления системой со сложным поведением в задаче «Умный муравей–3».

При выполнении первого этапа работ использовался следующий инструментарий:

1. Статьи в ведущих зарубежных и российских журналах, монографии и патенты за период 1998–2008 гг.
2. Результаты автоматического поиска в сети Интернет по ключевым словам и шаблонам.
3. Работы членов коллектива, опубликованные в рамках НИР по схожей тематике.
4. Аналитический обзор, составленный в рамках НИР.
5. Постановление Правительства Российской Федерации от 4 мая 2005 г. № 284 «О государственном учете результатов научно-исследовательских, опытно-конструкторских и технологических работ гражданского назначения».
6. Язык программирования *Java*.
7. Интегрированная среда разработки *Eclipse*.

8. Персональные компьютеры.
9. ГОСТ 7.32-2001 «Отчет о научно-исследовательской работе. Структура и правила оформления».

Излагаются результаты выполнения аналитического обзора по следующим направлениям: системы со сложным поведением и автоматное программирование, методы построения конечных автоматов с помощью генетических алгоритмов и генетического программирования, программы-симуляторы беспилотных летательных аппаратов, методы построения автопилотов верхнего уровня для летательных аппаратов.

Приводится обоснование выбора оптимального варианта направления исследований, а также план проведения теоретических и экспериментальных исследований.

Описывается метод генетического программирования, основанный на применении линейных бинарных графов для представления функции переходов конечных автоматов. Приводится его программная реализация.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящем отчете излагаются результаты выполнения *первого этапа поисковых научно-исследовательских работ по направлению «Информатика» по проблеме «Разработка методов совместного применения генетического и автоматного программирования для построения систем управления беспилотными летательными аппаратами»*, выполняемых в рамках государственного контракта, заключенного между Федеральным агентством по образованию и государственным образовательным учреждением высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» в соответствии с решением *Единой комиссии (протокол от 07 августа 2009 г. № 3/НК-178П) по конкурсу № НК-178П* «Проведение поисковых научно-исследовательских работ по направлению «Информатика» в рамках мероприятия 1.3.1 Программы», выполняемому в рамках мероприятия 1.3.1 «Проведение научных исследований молодыми учеными – кандидатами наук» мероприятия 1.3 «Проведение научных исследований молодыми учеными – кандидатами наук и целевыми аспирантами в научно-образовательных центрах» направления 1 «Стимулирование закрепления молодежи в сфере науки, образования и высоких технологий» федеральной целевой программы «Научные и научно-педагогические кадры инновационной России» на 2009-2013 годы, утвержденной постановлением Правительства Российской Федерации от 28 июля 2008 года № 568 «О федеральной целевой программе «Научные и научно-педагогические кадры инновационной России» на 2009-2013 годы».

Целями настоящего этапа являются:

1. Выполнение аналитического обзора.
2. Выбор и обоснование оптимального варианта направления исследований.
3. Подготовка плана проведения теоретических и экспериментальных исследований.
4. Разработка и программная реализация метода генетического программирования, основанного на использовании линейных бинарных графов для представления функции переходов автоматов.
5. Экспериментальное исследование метода генетического программирования, основанного на использовании линейных бинарных графов для представления функции переходов автоматов, на примере построения автомата управления системой со сложным поведением в задаче «Умный муравей–3».

Отчет имеет следующую структуру. В первой главе приводятся результаты выполнения аналитического обзора по четырем направлениям:

- системы со сложным поведением и автоматное программирование;
- методы построения конечных автоматов с помощью генетических алгоритмов и генетического программирования;
- программы-симуляторы летательных аппаратов.

Во второй главе изложен план проведения теоретических и экспериментальных исследований, а также обосновывается выбор оптимального направления исследований.

В третьей главе приводится описание метода генетического программирования, основанного на использовании линейных бинарных графов для представления функции переходов автоматов, а также его программная реализация на языке программирования *Java*.

Каждая из глав снабжена выводами, кратко резюмирующими содержание главы. В заключении дается общая оценка работ по этапу.

Исследования, проводимые, например, в университете Стэнфорда, показывают, что для управления беспилотным летательным аппаратом целесообразно применять модели, основанные на состояниях. Автомат управления беспилотным летательным аппаратом на верхнем уровне построить сравнительно просто – он будет содержать состояния «Посадка», «Взлет», «Набор высоты»,

состояния, соответствующие различным режимам полета и т.д. В то же время построение автоматов, соответствующих различным режимам полета, вручную является достаточно трудоемкой и сложной задачей.

Возникает естественное желание – автоматизировать процесс проектирования автоматов, поручив основную работу компьютеру – это позволяет уменьшить затраты времени на разработку систем управления, а также сократить влияние человеческого фактора.

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 управления беспилотными летательными аппаратами.

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 скрещивания 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.

Изложенное позволяет утверждать, что результаты выполнения научно-исследовательской работы будут превышать мировой уровень разработок в рассматриваемой области.

1. АНАЛИТИЧЕСКИЙ ОБЗОР

В настоящей главе приводятся результаты аналитического обзора. Аналитический обзор проводился по следующим направлениям:

- сущности со сложным поведением и автоматное программирование;
- методы построения конечных автоматов с помощью генетических алгоритмов и генетического программирования;
- программы-симуляторы летательных аппаратов.

1.1. Сущности со сложным поведением и автоматное программирование

В настоящем разделе приводится аналитический обзор работ, посвященных автоматному программированию и сущностям со сложным поведением.

1.1.1. Системы со сложным поведением

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

Будем считать, что сущность обладает сложным поведением, если в ответ на одно и то же входное воздействие она может сгенерировать в зависимости от предыстории различные выходные воздействия. Сущность с простым поведением в ответ на одно и то же входное воздействие всегда генерирует одно и то же выходное воздействие (рис. 1).

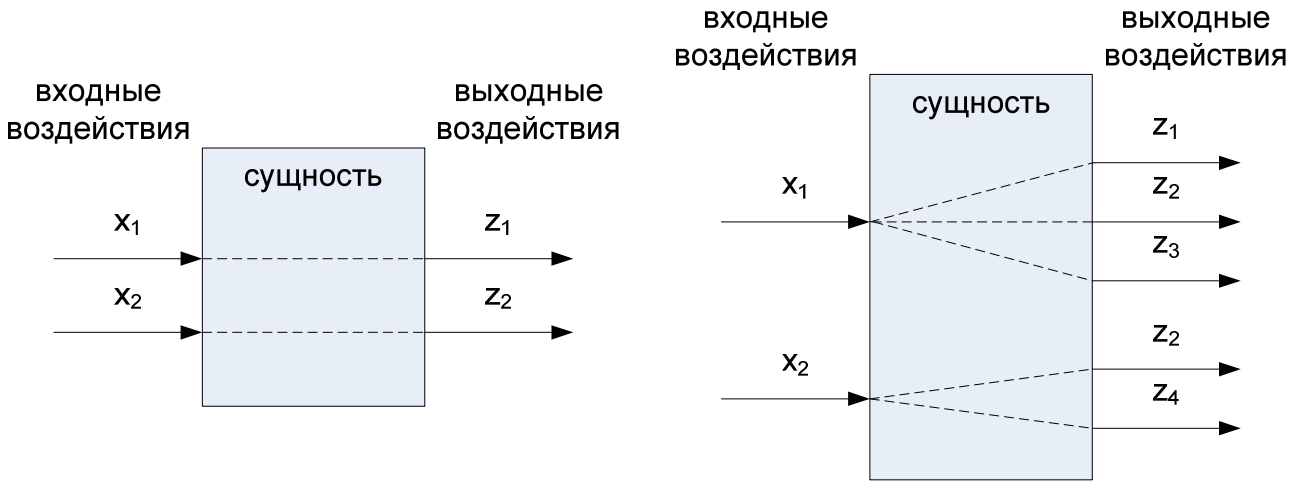

Рис. 1. Сущность с простым поведением и сущность со сложным поведением

При традиционной программной реализации сущностей 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 [17]. При этом все поведение системы оказывается сосредоточенной в автомате или системе взаимодействующих автоматов. Такой подход к описанию поведения называют *автоматным* [11].

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

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 [19] 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 [15]. На рис. 2 изображена схема взаимодействия автоматов, которые образуют эту систему.

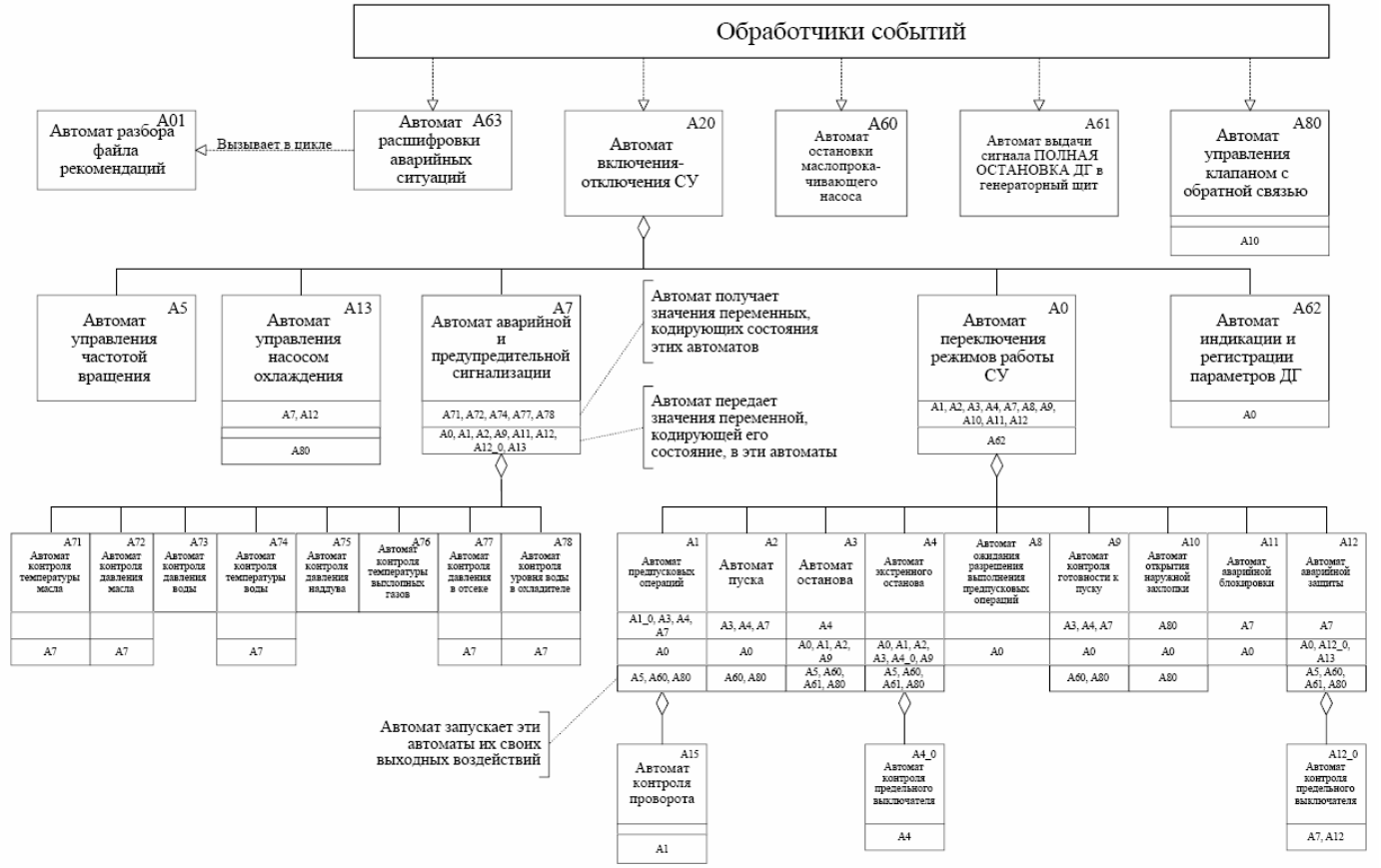

Рис. 2. Схема взаимодействия автоматов, управляющих системой со сложным поведением

1.1.2. Автоматное программирование

Автоматное программирование, предложенное в работе [13], – это парадигма программирования, которая состоит в представлении сущностей со сложным поведением в виде автоматизированных объектов управления.

Одна из центральных идей автоматного программирования 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.

Базовым понятием автоматного программирования является «состояние». Это понятие в том смысле, как оно используется в описываемой парадигме, было введено А. Тьюрингом и с успехом применяется во многих развитых областях науки, например, в теории управления и теории формальных языков.

Основное свойство состояния системы в момент времени t_0 заключается в «отделении» будущего ($t > t_0$) от прошлого ($t < t_0$) в том смысле, что текущее состояние несет в себе всю информацию о прошлом системы, необходимую для определения ее реакции на любое входное воздействие, формируемое в момент времени t_0.

При описании понятия «сложное поведение» предполагается, что реакция сущности со сложным поведением на входное воздействие может зависеть, в том числе, и от предыстории. При использовании понятия «состояние» знание предыстории более не требуется. Состояние можно

рассматривать как особую характеристику, которая в неявной форме объединяет все входные воздействия прошлого, влияющие на реакцию сущности в настоящий момент времени. Реакция зависит теперь только от входного воздействия и текущего состояния.

Состояния системы могут быть разделены на управляющие и вычислительные. Неформально основные различия между ними сформулированы в табл. 1.

Таблица 1. Управляющие и вычислительные состояния

Управляющие состояния	Вычислительные состояния
Их число не очень велико	Их число либо бесконечно, либо конечно, но очень велико
Каждое из них имеет вполне определенный смысл и качественно отличается от других	Большинство из них не имеет смысла и отличается от остальных лишь количественно
Они определяют действия, которые совершает сущность	Они непосредственно определяют лишь результаты действий

Понятие входное воздействие также является одним из базовых для автоматного программирования. Чаще всего, входное воздействие – это вектор. Его компоненты подразделяются на события и входные переменные в зависимости от смысла и механизма формирования. Совокупность конечного множества состояний и конечного множества входных воздействий образует (конечный) автомат без выходов. Такой автомат реагирует на входные воздействия, определенным образом изменяя текущее состояние. Правила, по которым происходит смена состояний, называют функцией переходов автомата.

То, что в автоматном программировании собственно и называется (конечным) автоматом (рис. 3), получается, если соединить понятие «автомат без выходов» с понятием «выходное воздействие». Такой автомат реагирует на входное воздействие не только сменой состояния, но и формированием определенных значений на выходах. Правила формирования выходных воздействий называют функцией выходов автомата.

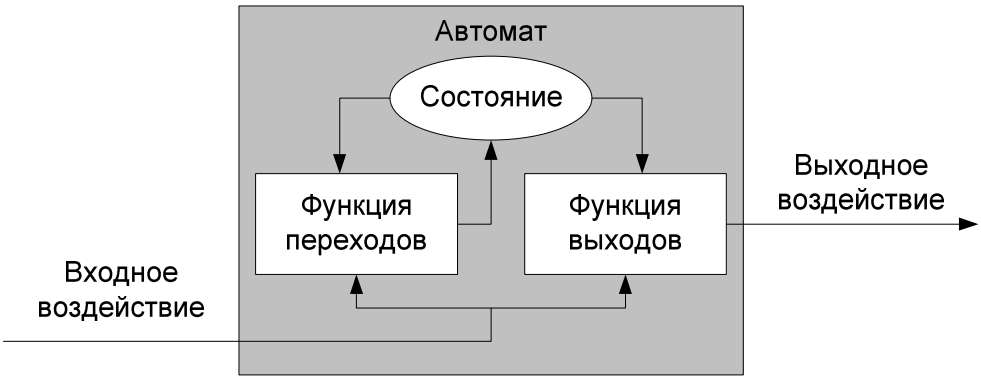

Рис. 3. Конечный автомат

При построении систем автоматизации технических процессов обычно выделяют управляющие устройства и управляемые объекты. Следуя этой концепции, сущность со сложным поведением естественно разделить на две части:

- *управляющую часть*, ответственную за логику поведения (выбор выполняемых действий, зависящий от текущего состояния и входного воздействия), а также за переход в новое состояние;
- *управляемую часть*, ответственную за выполнение действий, выбранных для выполнения управляющей частью, и, возможно, за формирование некоторых компонентов входных воздействий для управляющей части – обратных связей.

В соответствии с традицией теории управления, управляемая часть здесь и далее называется «объект управления», а управляющая часть – «система управления». Поскольку для реализации управляющей части используются автоматы, то она часто называется «управляющий автомат» или просто «автомат».

После разделения сущности со сложным поведением на объект управления и автомат реализовать ее уже несложно, а главное, ее реализация становится понятной и удобной для модификации. Вся логика поведения сущности сосредоточена в управляющем автомате. Объект управления, в свою очередь, обладает простым поведением (а, следовательно, может быть легко реализован традиционными «неавтоматными» методами). Он не обрабатывает непосредственно входные воздействия от внешней среды, а только получает от автомата команды совершить те или иные действия. При этом каждая команда всегда вызывает одно и то же действие (это и есть определение простого поведения).

Таким образом, в соответствии с автоматным подходом, сущности со сложным поведением следует представлять в виде *автоматизированных объектов управления* – так в теории управления называют объект управления, интегрированный с системой управления в одно устройство.

При построении модели автоматизированного объекта предполагается, что управляющий автомат взаимодействует и с объектом управления, и с внешней средой (рис. 4).

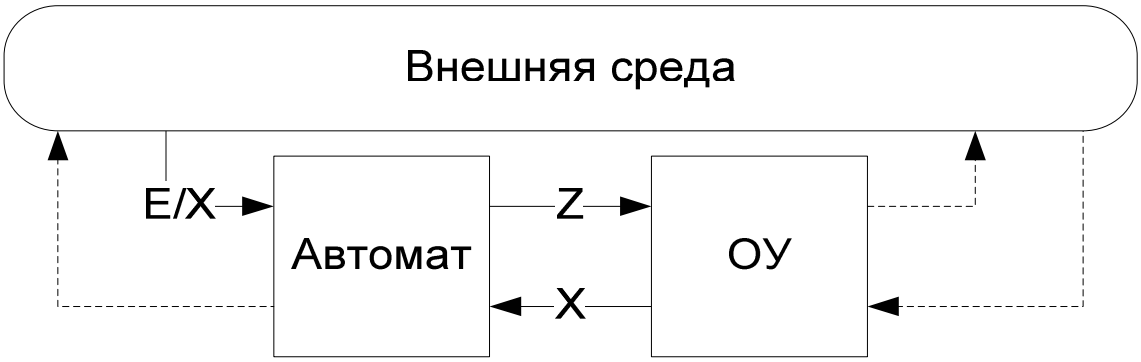

Рис. 4. Взаимодействие компонентов модели автоматизированного объекта

На этом рисунке сплошными стрелками обозначены традиционные и наиболее типичные для программных реализаций виды взаимодействия между автоматом, объектом управления и внешней средой. Автомат получает входные воздействия, как со стороны среды, так и от объекта управления. В событийных системах часть или все компоненты входного воздействия со стороны среды могут быть событиями (множество событий обозначено на рисунке буквой *E*). Входное воздействие со стороны объекта управления формирует в модели обратную связь (от управляемого объекта к управляющему). Это воздействие может отсутствовать, тогда модель является разомкнутой – так в теории управления называются системы управления без обратной связи [5]. В противном случае модель называется замкнутой. Автомат, в свою очередь, воздействует на объект управления.

Пунктирными стрелками обозначены менее распространенные, хотя и возможные, варианты взаимодействия. Так, автомат может оказывать выходное воздействие и на внешнюю среду. Однако таких связей обычно можно избежать, включив все управляемые автоматом сущности в состав его

объекта управления. Отметим, что в программировании, в общем случае, различие между объектом управления и внешней средой носит скорее концептуальный, а не формальный характер. Создавая модель системы со сложным поведением, разработчик производит ее декомпозицию на *автоматизированные объекты*, определяя тем самым объект управления каждого автомата. В целях минимизации связей между модулями программной системы целесообразно проводить декомпозицию таким образом, чтобы автомат оказывал выходные воздействия только на собственный объект управления.

Кроме того, объект управления может взаимодействовать с внешней средой напрямую.

Напомним, что в абстрактных автоматных моделях входные и выходные воздействия обычно представляют собой символы некоторого конечного алфавита или цепочки таких символов, а в структурных моделях – битовые строки заданной длины. В программировании на вид входных и выходных воздействий не накладывается ограничений: это могут быть символы, числа, строки, множества, последовательности, произвольные объекты – все зависит от специфики поставленной задачи и инструментов, используемых для ее решения. Кроме того, могут различаться способы передачи входных воздействий автомату и интерпретации выходных воздействий в объекте управления.

Если по назначению сущность близка к традиционной системе управления, то представление входных и выходных воздействий битовыми строками будет удобным по тем же причинам, что и для структурных автоматных моделей. Однако интерпретация этого представления может быть различной. В программировании обычно используется такая интерпретация: каждой выходной переменной сопоставляется определенное *изменение* вычислительного состояния (действие, команда). При этом единица обозначает наличие действия, а ноль – его отсутствие. В этом случае вектору из нулей соответствует отсутствие каких-либо команд. Такой вид выходного воздействия может привести к недетерминизму в том случае, если результат зависит от последовательности выполнения команд. Поэтому в качестве выходного воздействия вместо множества команд часто используется последовательность команд.

При рассмотрении всевозможных деталей использования автоматных моделей в программировании становится ясно, что выбрать одну конкретную модель, подходящую для всех задач, невозможно. При программной реализации сущностей со сложным поведением применение могут найти активные и пассивные, разомкнутые и замкнутые модели, различные формы представления и интерпретации входных и выходных воздействий. Модель автоматизированного объекта управления должна быть применима для любой сущности со сложным поведением, и поэтому целесообразно сформулировать ее довольно абстрактно. Примеры программной реализации сущностей со сложным поведением, которые будут приведены в последующих главах, являются конкретными воплощениями этой модели.

Приведем формальное определение автоматизированного объекта управления.

Пара <*A, O*>, состоящая из управляющего автомата и объекта управления, называется автоматизированным объектом управления.

Управляющий автомат представляет собой шестерку $\langle X, Y, Z, y_0, \varphi, \delta \rangle$, где $X = X_E \times X_O$ – конечное множество входных воздействий, причем каждое входное воздействие x состоит из компоненты x_E, порождаемой внешней средой, и компоненты x_O, порождаемой объектом управления; Y – конечное множество управляющих состояний; Z – конечное множество выходных воздействий; $y_0 \in Y$ – начальное состояние; $\varphi = (\varphi', \varphi'')$ – функция выходов (выходных воздействий), состоящая, в общем случае, из двух компонент: функции выходных воздействий в состояниях $\varphi' : Y \to Z$ и функции выходных воздействий на переходах $\varphi'' : X \times Y \to Z$; $\delta : X \times Y \to Y$ – функция переходов.

Объект управления – это тройка $\langle V, f_q, f_c \rangle$, где V – потенциально бесконечное множество вычислительных состояний (или значений), $f_q : V \to X_O$ – функция, сопоставляющая входное воздействие вычислительному состоянию, $f_c : Z \times V \to V$ – функция, изменяющая вычислительное состояние в зависимости от выходного воздействия.

Функции f_q и f_c являются математическими эквивалентами набора запросов и набора команд соответственно.

Графическое представление описанной модели приведено на рис. 5.

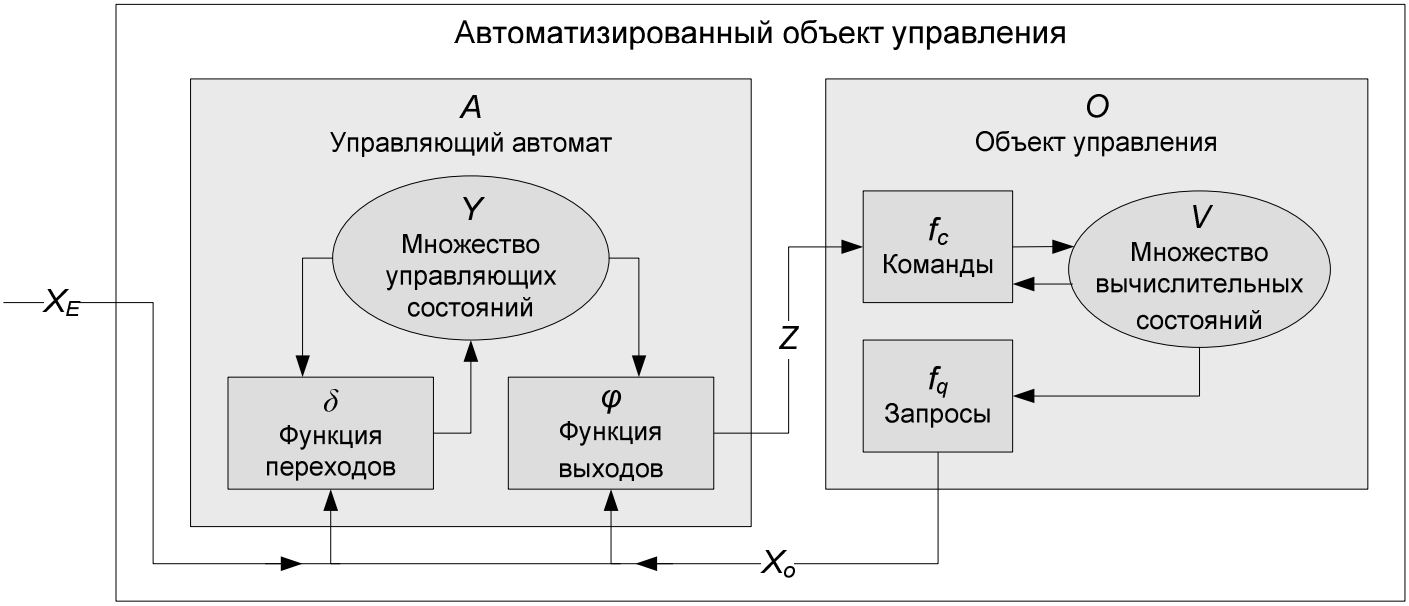

Рис. 5. Автоматизированный объект управления

Таким образом, с позиций теории формальных языков автоматизированный объект – это автомат с произвольной памятью. По вычислительной мощности он, в общем случае, эквивалентен машине Тьюринга. Формально, для обеспечения эквивалентности необходимо потребовать, чтобы запросы и команды объекта управления являлись *вычислимыми* функциями (их можно было вычислить с помощью машины Тьюринга). В определении это свойство подразумевается.

В разомкнутой модели автоматизированного объекта входные воздействия поступают только от внешней среды ($X = X_E$), а запросы объекта управления отсутствуют (рис. 6). Такой автоматизированный объект по вычислительной мощности эквивалентен детерминированному конечному автомату: его объект управления уже не является дополнительной памятью, а скорее представляет собой разновидность выходной ленты.

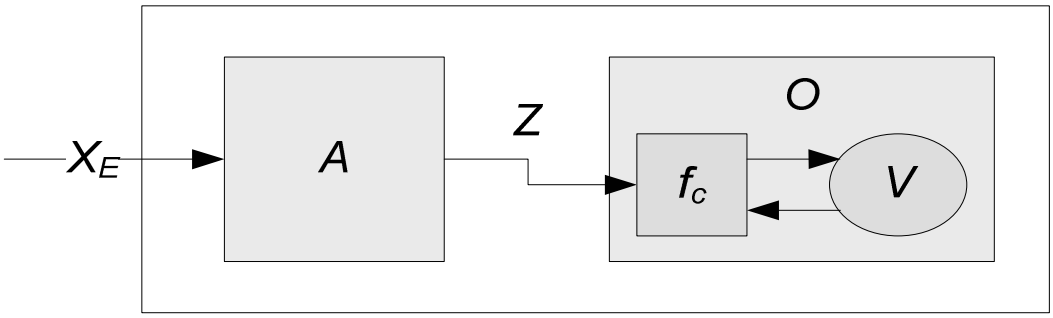

Рис. 6. Разомкнутый автоматизированный объект управления

В терминах теории логического управления автоматизированный объект – это, в общем случае, замкнутая система, управляемая автоматом первого рода с выходным преобразователем, в качестве которого может использоваться автомат Мура, Мили или смешанный автомат.

Как было упомянуто выше, автоматизированный объект эквивалентен по вычислительной мощности машине Тьюринга. Иначе говоря, для любого автоматизированного объекта можно построить машину Тьюринга, которая решает ту же задачу, и наоборот. С одной стороны, из этого свойства следует важное достоинство автоматного подхода: с помощью автоматизированного объекта можно описать любой алгоритм, любую программу, которая только может быть выполнена компьютером. С другой стороны, возникает вопрос: зачем было изобретать автоматизированный объект вместо того, чтобы выбрать на роль модели сущности со сложным поведением более простую и столь же мощную машину Тьюринга?

Программирование на машине Тьюринга (или *тьюрингово программирование* [16]) чрезвычайно сложно и непрактично по той причине, что набор элементарных операций этой машины с дополнительной памятью очень ограничен. Поэтому автомату приходится не только управлять, но и *выполнять не свойственную ему функцию вычисления*. Переход к модели автоматизированного объекта управления позволяет использовать в качестве элементарных операций произвольные запросы и команды. При этом на управляющий автомат ложится лишь часть ответственности по реализации алгоритма: та, что связана с логикой (управлением) – то, для чего автоматы, собственно, и предназначены.

Можно сказать, что при программировании на машине Тьюринга любое поведение (кроме алгоритма, состоящего из единственного шага, на котором необходимо записать символ и сдвинуть головку) является сложным. В автоматном программировании грань между простым и сложным поведением определяется разработчиком. Умножение двух чисел может быть сложным, если вы собираетесь эмулировать и визуализировать счеты. Построение сбалансированного дерева поиска может быть элементарной операцией, если в вашем распоряжении есть соответствующая библиотека, предоставляющая готовую функцию. Таким образом, переход от тьюрингова программирования к автоматному состоит в повышении уровня абстракции операций с памятью, причем этот уровень при автоматном подходе не фиксирован, а зависит от решаемой задачи.

Более того, низкоуровневый автоматизированный объект может быть инкапсулирован в объекте управления, существующем на более высоком уровне абстракции. Благодаря такому вложению автоматизированных объектов автоматное программирование поддерживает концепцию выделения уровней абстракции, распространенную в современной методологии разработки ПО.

Выбор уровня абстракции элементарных операций при моделировании сущности со сложным поведением определяет разделение поведения на логику и семантику, а состояний – на управляющие и вычислительные. Это и есть наиболее творческий и нетривиальный шаг в автоматном подходе к

разработке ПО. Выбор чересчур простых элементарных операций приводит к разрастанию и усложнению автомата, логика становится менее понятной и перегруженной деталями, которые эффективнее было бы реализовать в объекте управления. Машина Тьюринга – крайний случай такого «злоупотребления логикой».

Напротив, выбор слишком абстрактных запросов и команд ведет к усложнению их реализации, перегруженности флагами и условными конструкциями. Вырожденный случай такого «злоупотребления семантикой» – использование традиционных (неавтоматных) стилей программирования, где любое поведение считается простым.

1.2. Методы построения конечных автоматов с помощью генетических алгоритмов и генетического программирования

В настоящем разделе приводится аналитический обзор работ, посвященных методам построения конечных автоматов с помощью генетических алгоритмов и генетического программирования.

1.2.1. 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 [32]. 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 [33]. 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 [39], 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* [46], употребляется термин «традиционное генетическое программирование». Для обозначения генетических алгоритмов, оперирующих с битовыми строками фиксированной длины (описанных, например, в работе [52]), используется термин «традиционные генетические алгоритмы».

1.2.2. Преимущества автоматизированного построения автоматов

В случае, когда автоматы описывают системы со сложным поведением, их проектирование является нетривиальной и трудоемкой 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*.

В настоящем разделе рассматривается 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.

1.2.2.1. Распознавание изображений

Конечные автоматы успешно применялись в работе [26] для распознавания изображений. Задача была поставлена следующим образом: необходимо выделить корабли на изображении, полученном с радара. Пример изображения приведен на рис. 7 (белыми квадратами выделены корабли).

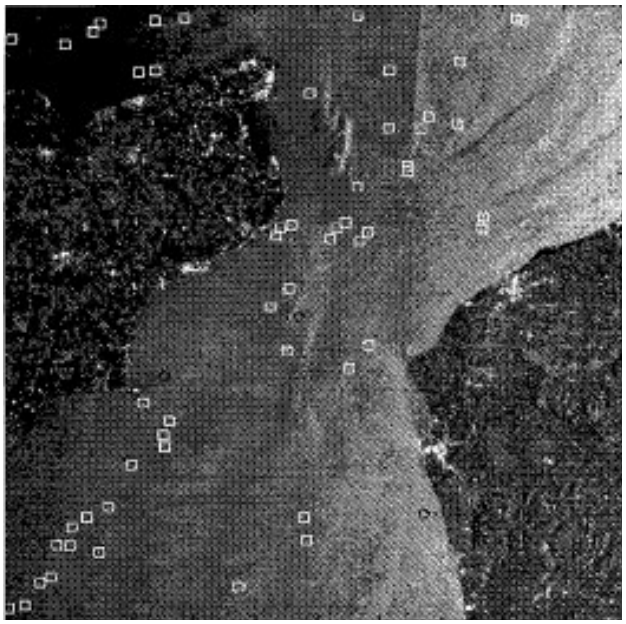

Рис. 7. Пример изображения с радара

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 начального 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.

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

В работе [40] авторы предлагают новой подход к решению известной в молекулярной биологии задачи выбора праймера для полимеразной цепной реакции (ПЦР) [14]. 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 [14], благодаря которым могут быть построены различные эвристические методы проверки праймера.

В указанной работе 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 указанной работы, 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. 8 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 число хороших и плохих состояний, в которые переходит автомат по мере увеличения принимаемого префикса. Если число плохих состояний превосходит число хороших, то считается, что праймер не подходит. Если число плохих состояний меньше числа хороших, то считается, что праймер подходит для ПЦР. В случае же равенства – ответ не дается.

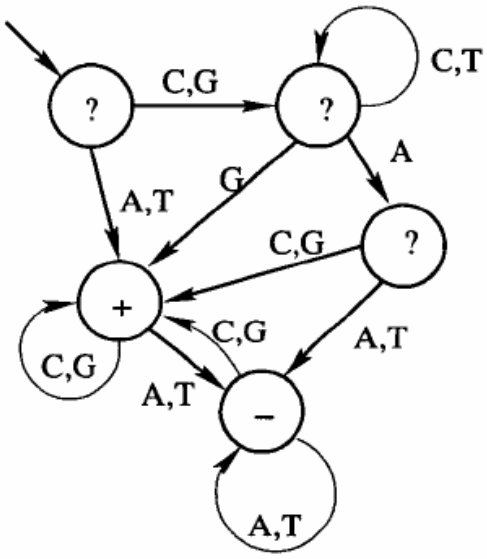

Рис. 8. Структура автомата, классифицирующего праймеры

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 соответствует приросту 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 классификации, 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.

1.2.2.3. 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 числом состояний. Однако, в работе [36] показано, что модифицированная задача *NP*-трудна.

В работе [25] 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 – число верно распознанных тестовых примеров, число 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 выполняется 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 указанной работы удалось построить автомат из семи состояний, который распознает по множеству из ста примеров русские двухсложные слова.

В работе [27] для генерации автомата был применен следующий вариант генетического программирования: выводилось выражение, результатом вычисления которого являлся автомат. Выражение описывало последовательность действий, преобразующих исходный автомат (рис. 9) в результирующий. Для вывода выражений, кодирующих развитие автомата, было применено традиционное генетическое программирование, оперирующее над деревьями выражений.

Рис. 9. 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 [27] введен единственный терминал `DONE`, означающий конец редактирования. Пример выражения, использующего эти функции, приведен на рис. 10. 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`.

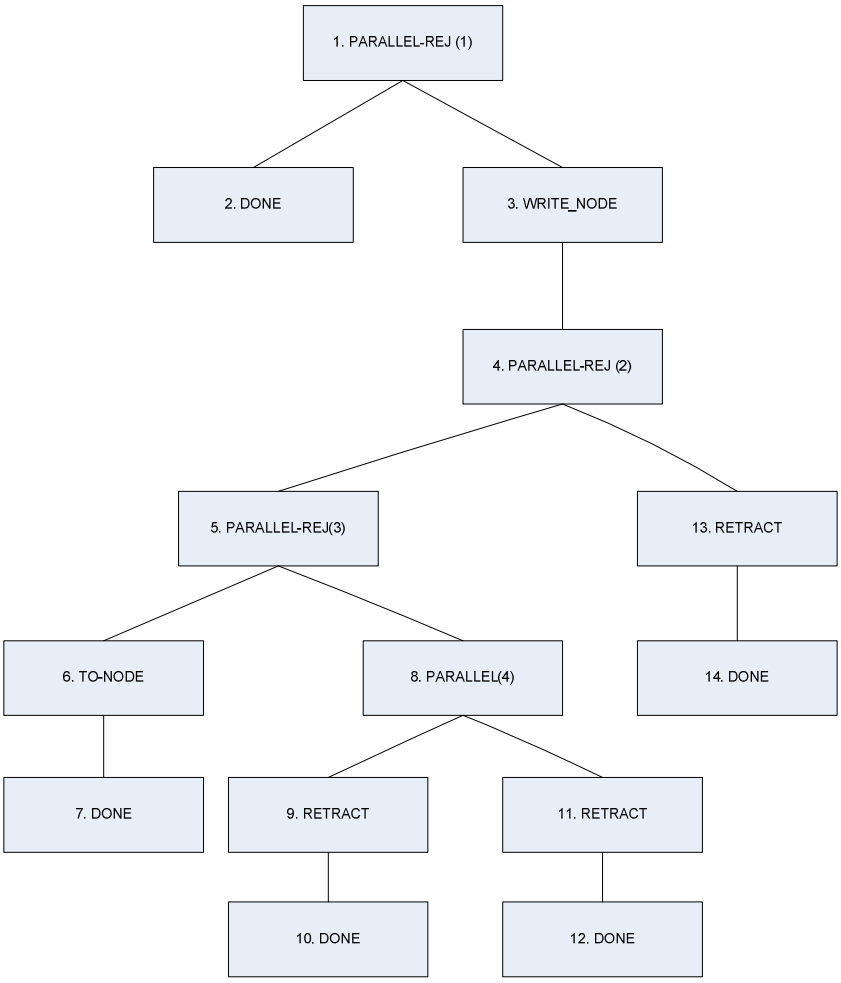

Рис. 10. Выражение, описывающее развитие автомата

На рис. 11 показан процесс развития автомата из исходного по приведенному выражению.

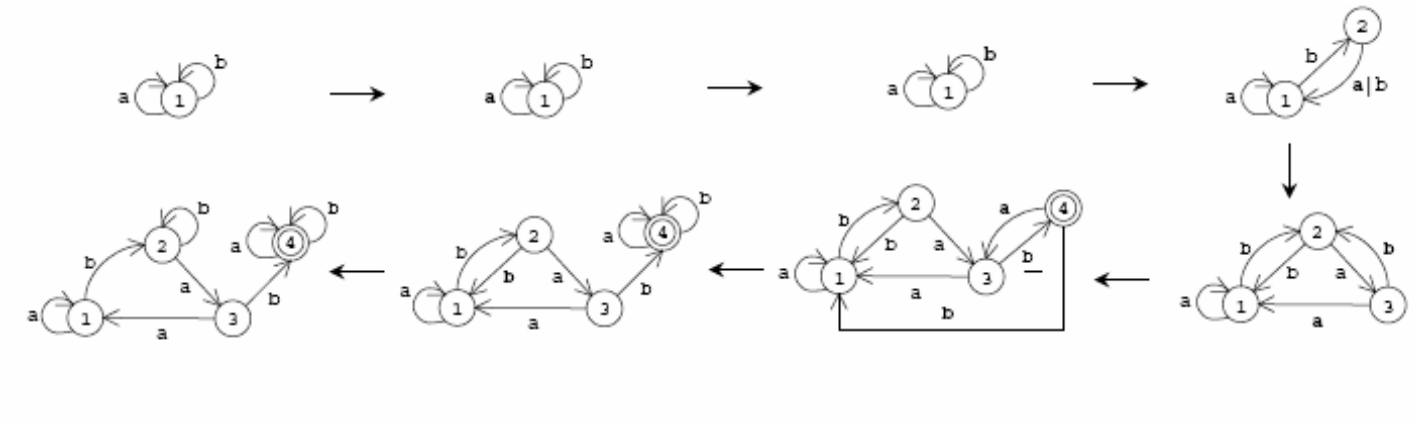

Рис. 11. Процесс построения автомата при вычислении выражения

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 [27] 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. 12.

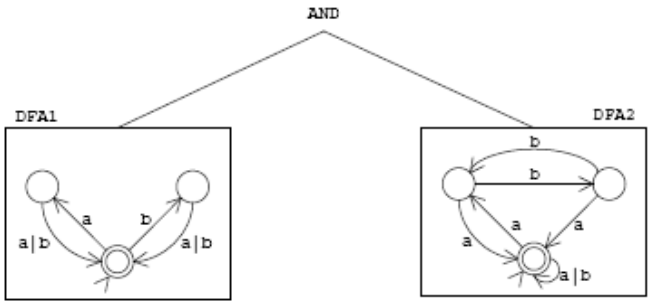

Рис. 12. Декомпозиция распознавателя

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 [49] 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.

1.2.2.4. Искусственная этология

Этология – это дисциплина зоологии, занимающаяся изучением поведения особи в естественной для нее 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 числом 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 [37] с помощью искусственной этологии изучаются процессы коммуникации между особями в ходе эволюции их популяции. 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 авторами указанной работы, 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. 13.

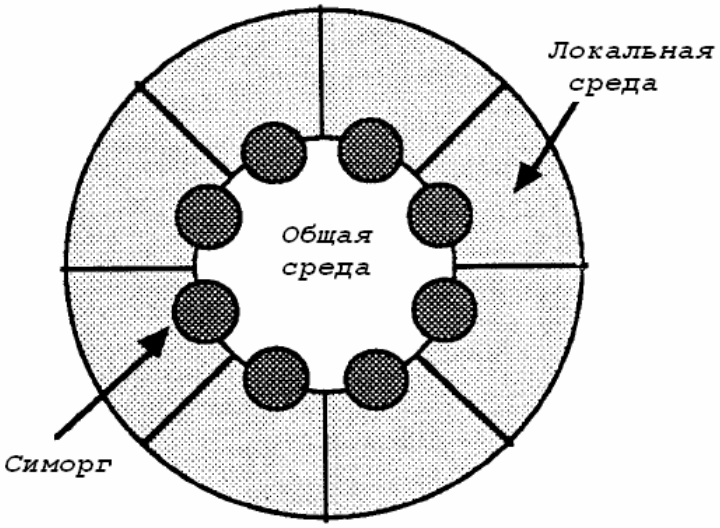

Рис. 13. Топология среды обитания симоргов

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, последним отправивший 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. 14 приведены зависимости функции приспособленности популяции (α) от числа итераций эволюционного алгоритма (t) для случая, когда возможность коммуникации отсутствовала (а) и когда коммуникация была разрешена (б).

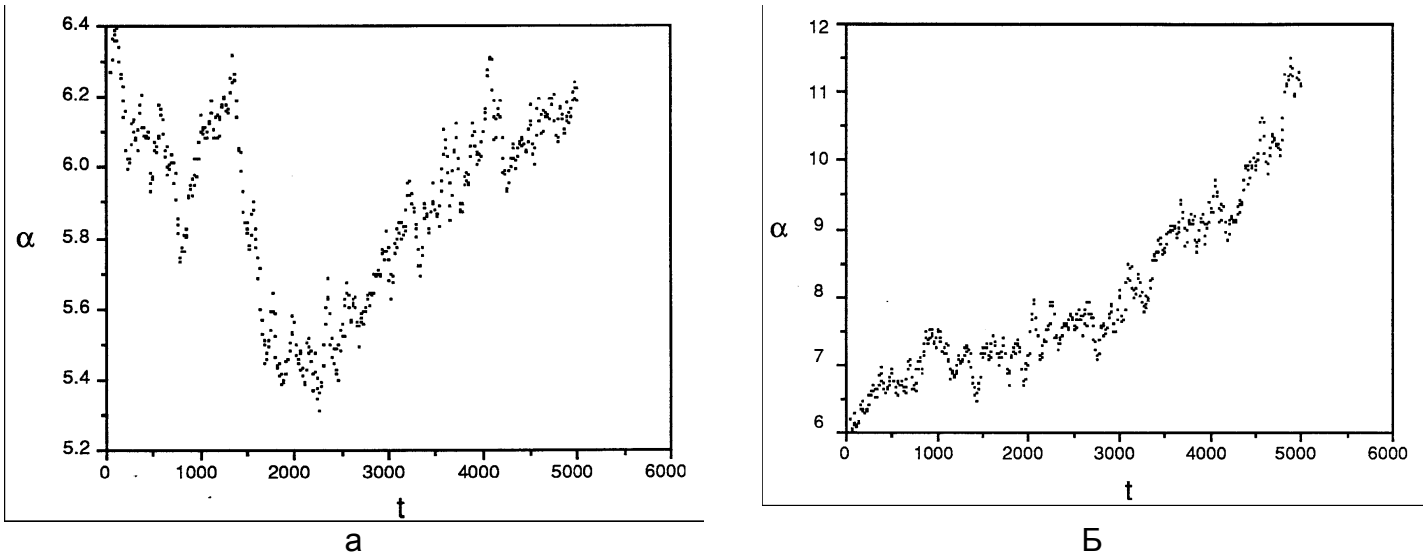

Рис. 14. Результаты экспериментов

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

1.2.2.5. Сравнение автоматных программ с деревьями

В работе [74] анализируется эффективность применения автоматных программ в генетическом программировании по сравнению с деревьями. Анализируются структуры и свойства представления программ, адекватные решаемой задаче. По словам автора этой работы, в предыдущих его работах для управления роботами автоматы использовались по следующей причине: автоматы схожи по структуре с задачей контроллера робота – робот в процессе выполнения некоторой деятельности всегда находится в некотором состоянии и отвечает на внешние воздействия немедленными действиями или переходом в другое состояние. Таким образом, деятельность робота может быть точно промоделирована диаграммой переходов. Кроме того, по мнению этого автора, конечные автоматы более просты в понимании, анализе и верификации, чем другие представления, такие как, например, нейронные сети. Автоматы удобнее для инкрементальной эволюции контроллеров, исследование которой продолжается в указанной работе. При этом расширение функциональности состоит в добавлении состояний и переходов, при относительно небольших изменениях в ранее существовавших состояниях и переходах, в то время как архитектуру нейронных сетей в таких случаях часто приходится изменять очень значительно, за исключением случаев применения какого-нибудь модульного подхода. Однако, исследование модульных подходов к проектированию нейронных сетей еще находится на очень ранней стадии своего развития.

В рассматриваемой работе в качестве условия перехода используется сравнение значений двух регистров, или значения регистра с константой (напомним, что ранее автор использовал сообщения, а

в исследовании, проводимом по настоящему контракту, анализируются многоместные булевы формулы). Кроме того, упоминаются взвешенные автоматы и модели Маркова. Сделан вывод о том, что большая эффективность деревьев или автоматов зависит от решаемой задачи. Деревья более удобны в случае линейных программ, а автоматы – в случае повторяющихся и возможно нерегулярных шаблонов поведения, с возможностью изменения режима функционирования. Для новой модели автомата характерны следующие особенности:

- две анализируемые при выборе перехода переменные;
- не обеспечивается полнота условий на переходах. В случае отсутствия перехода, условие которого выполняется, программа завершается;
- непротиворечивость переходов обеспечивается через их упорядочивание.

1.2.2.6. Задача о «Флибах» (задача прогнозирования)

В работе [4] рассматривается решение задачи о «Флибах» при помощи генетических алгоритмов.

Рассматриваемая задача – это задача моделирования простейшего живого существа, которое способно предсказывать имеющие периодичность изменения простейшей окружающей среды. При этом живое существо моделируется конечным автоматом, а генетические алгоритмы позволяют автоматически построить автомат, который предсказывает изменения среды с достаточно высокой точностью. Таким образом, при решении данной задачи требуется построить «устройство», которое предсказывает изменения среды с наибольшей вероятностью.

Одна из важнейших способностей живых существ – предсказание изменений окружающей среды. В качестве простейшей модели такого существа можно использовать конечный автомат. Такие конечноавтоматные модели названы ***флибами***.

На вход флиба подается переменная, которая принимает одно из двух значений – ноль или единица. Эта переменная соответствует состоянию окружающей среды в текущий момент времени. Среда настолько проста, что имеет только два состояния. Флиб изменяет свое состояние и формирует значение выходной переменной, принимающей одно из двух указанных значений. Это значение соответствует возможному состоянию среды в следующий момент времени. Задача флиба – предсказать какое на самом деле состояние окружающей среды наступит в следующий момент времени. Это можно выполнить благодаря периодичности изменений состояний среды. При этом считается, что чем выше вероятность предсказания изменений среды, тем больше вероятность у флиба выжить и оставить потомство.

В описываемой работе предложено применять метод генерации нового поколения, используя турнирный отбор и принцип элитизма (в новое поколение добавляется одна или несколько лучших особей из предыдущего поколения).

Алгоритм имеет следующий вид.

1. Создается текущее поколение случайных флибов.
2. Подсчитывается сколько изменений состояний окружающей среды правильно предскажет каждый из этих флибов.
3. Строится новое поколение флибов:
   a. Создается пустое новое поколение и в него добавляется лучший предсказатель из текущего поколения.
   b. Случайным образом из текущего поколения выбираются две пары флибов.
   c. Из каждой пары выбирается лучший предсказатель.
   d. Лучшие предсказатели из указанных пар *скрещиваются*.

e. Случайным образом определяется необходимость применения к полученному флибу *оператора мутации*. Если это необходимо, то указанный оператор применяется к флибу.
f. К флибу применяется новая операция – «Восстановление связей между состояниями автомата».
g. Флиб добавляется в новое поколение.
h. Переходим к пункту b, если число флибов в новом поколении меньше числа флибов в текущем поколении.

4. Текущее поколение флибов заменяется новым.
5. Если число поколений меньше заданного пользователем, то переходим к пункту 2.

В качестве входного сигнала для флибов используется повторяющаяся последовательность битов, названная битовой маской задающей среду. Эта маска в программе зациклена.

Лучшим предсказателем является тот флиб, который наиболее точно предсказывает входной сигнал. В качестве оценочной функции используется число правильно угаданных символов. После формирования нового поколения все флибы в нем устанавливаются в начальное состояние. Затем для определения приспособленности флибов, на вход каждого из них подается несколько входных символов. После этого можно строить новое поколение решений, используя в качестве приспособленности флиба значение поля, содержащее число правильно предсказанных символов.

В результате применения данного алгоритма вероятность успешного прогнозирования была увеличена с 0.72 до 0.92.

1.2.2.7. Сравнение рассмотренных работ

В табл. 4 приведено сравнение указанных работ.

Таблица 4. Сравнение работ, использующих автоматизированное построение автоматов

Название работы	Решаемая задача	Существование решений, без использования генетических алгоритмов	Применение высокоуровневого кодирования	Использование коэволюционного метода
Evolving Finite State Machines with Embedded Genetic Programming for Automatic Target Detection.	Распознавание изображений	Да	Нет	Да
ECHO: Explorations of Evolution in a Miniature World	Выбор праймера для полимеразной цепной реакции	Да	Нет	Нет
A Genetic Algorithm for Finite State Automata Induction with Application to Phonotactics	Распознавание языков	Да	Нет	Да

Evolving Deterministic Finite Automata Using Cellular Encoding	Распознавание языков	Да	да	нет
A Note on Boltzmann Tournament Selection for Genetic Algorithms and Population-Oriented Simulated Annealing	Искусственная этология	нет	нет	нет
Использование генетических алгоритмов для автоматического построения конечных автоматов в задаче о «Флибах».	Задача о «Флибах»	нет	да	нет

1.2.3. Преимущества и сравнение методов высокоуровневого кодирования 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.

1.2.3.1. Умный муравей (задача управления)

Приведем описание задачи «Умный муравей» на основе работ [21, 43]. Используется двумерный тор размером 32 на 32 клетки (рис. 15). На некоторых клетках поля расположены яблоки – черные клетки на рис. 15. 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.

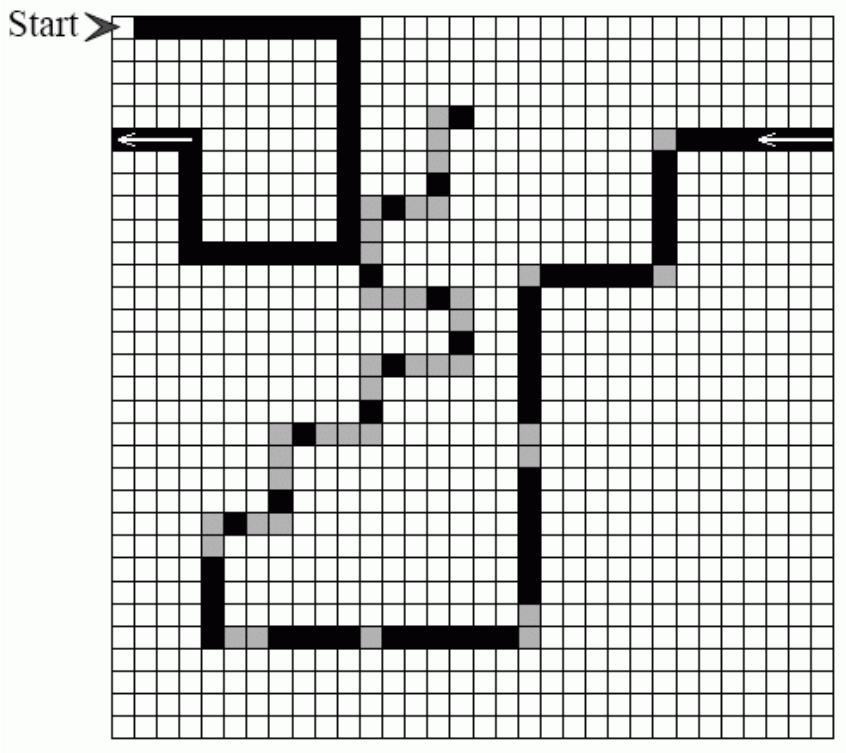

Рис. 15. Поле с яблоками

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 число яблок, съеденных муравьем. Это значение – результат игры.

Цель игры – создать муравья, который за 200 ходов съест как можно больше яблок. Муравьи, съевшие одинаковое число 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 числе 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 число яблок при заданном числе состояний.

Конечный автомат, изображенный на рис. 16, содержит всего пять состояний.

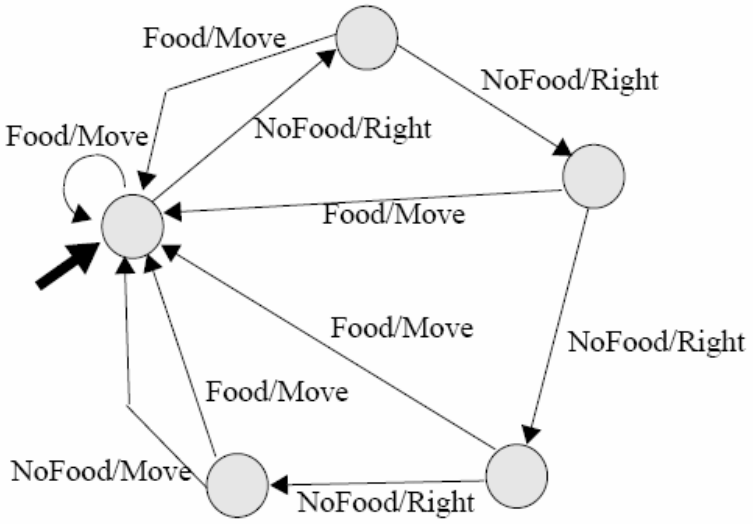

Рис. 16. Конечный автомат, задающий муравья

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. 17), который построен в работе [43] с помощью генетических алгоритмов и решает поставленную задачу.

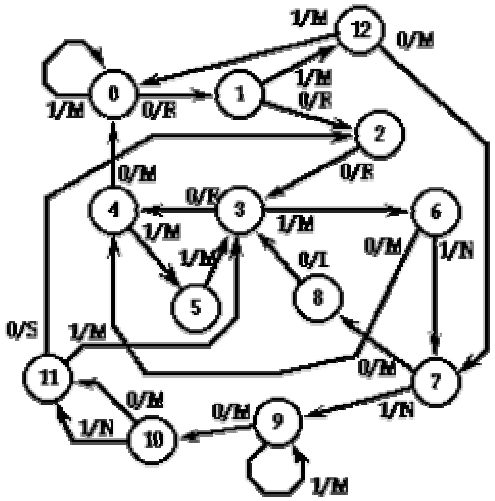

Рис. 17. Конечный автомат с 13 состояниями, задающий муравья, съедающего все яблоки за 200 ходов

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 [21], приведенный на рис. 18, содержит 11 состояний. По утверждению авторов этой работы, он съедает все яблоки за 193 хода. Этот граф некорректен. Как устранить некорректность указано в разд. 4.2.1.3.

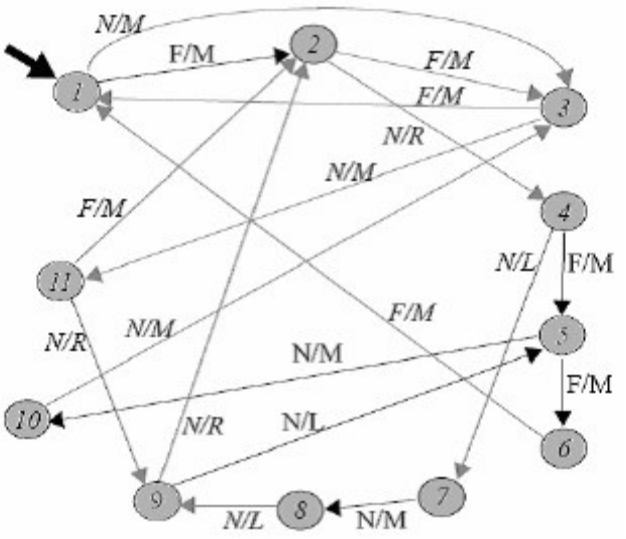

Рис. 18. Конечный автомат с 11 состояниями, задающий муравья, съедающего все яблоки за 193 хода

Для генерации конечных автоматов, задающего муравья, в работах [21, 43] были применены генетические алгоритмы. В работе [43] приспособленность особи (*fitness*) определяется как число 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. 19 приведен пример графа переходов автомата, описывающего поведение некоторого муравья.

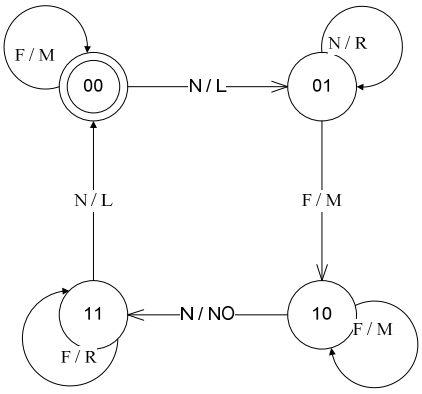

Рис. 19. Пример автомата, описывающего поведение муравья

Кодирование этого графа переходов приведено в табл. 5.

Таблица 5. 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 [46] 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 вычисляем выражение `A`, иначе – выражение `B`.
- `PROGN2 A B` – безусловный переход: вычисляем выражение `A`, а затем выражение `B`.
- `PROGN3` – аналогично `PROGN2`, с той лишь разницей что выполняются последовательно три аргумента данной функции.

В работе [46] 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. 20. В результате экспериментов, описанных в работе [46], был сгенерирован муравей, съедающий все яблоки на поле. На рис. 21 приведено задающее его дерево.

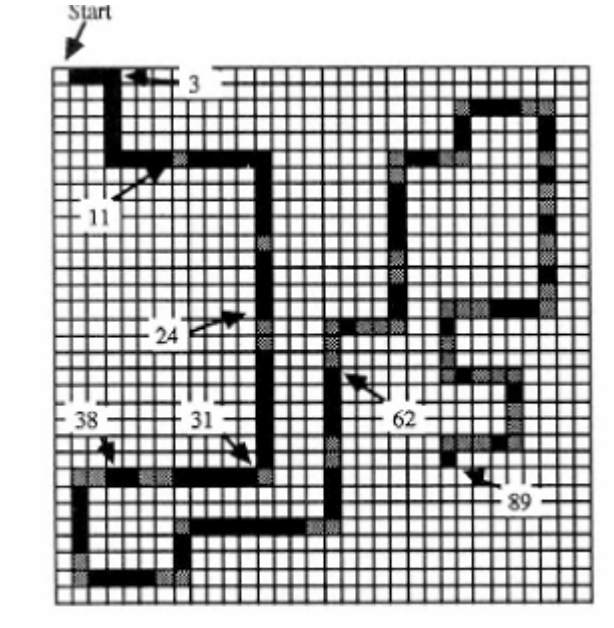

Рис. 20. Santa Fe Trail

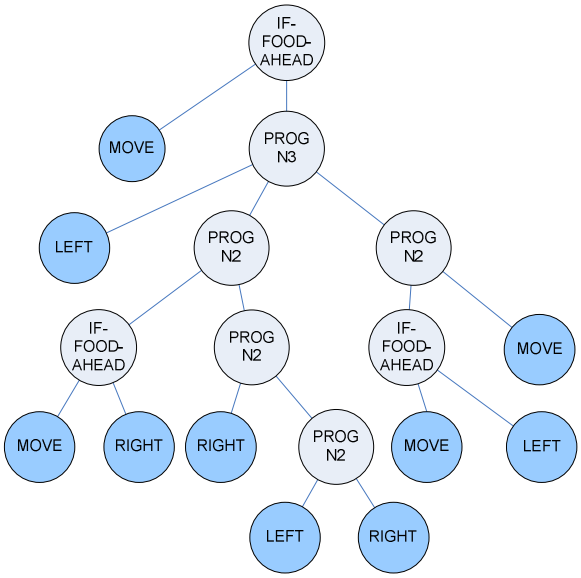

Рис. 21. Дерево выражений, задающее муравья

Автомат, соответствующий данному дереву, показан на рис. 22.

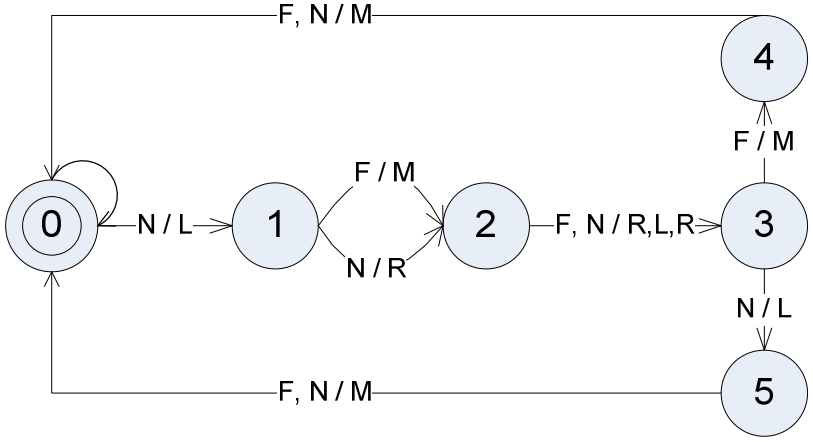

Рис. 22. Автомат, соответствующий дереву выражений

Отметим, что данный муравей был выведен на 21-ой итерации (поколении) алгоритма, а число особей в популяции равнялось 500. Таким образом, суммарное число вычислений функции приспособленности составило $21*500=10500$, в то время как в работе [43] на построение муравья, решающего задачу, потребовалось 94 поколения, при том, что размер популяции составлял 65536 автоматов. При этом число вычислений функции приспособленности равно $94*65536=6160384$.

1.2.3.2. Решение задачи «Умный муравей» при помощи автоматов Мили

В работе [18] находится оптимальное с точки зрения числа состояний решение задачи об «Умном муравье» при помощи автомата Мили.

Каждая особь генетического алгоритма представляет собой некоторый автомат, описывающий поведение муравья. Хромосома особи состоит из номера начального состояния и описаний всех состояний автомата. Описание одного состояния содержит описания двух переходов, соответствующих тому, что перед муравьем либо есть еда, либо ее нет. Описание перехода состоит из номера состояния, в которое автомат переходит, и действия, выполняемого на этом переходе.

Начальное поколение состоит из фиксированного числа случайно сгенерированных автоматов. Все автоматы в поколении имеют одинаковое наперед заданное число состояний.

При мутации случайно выбирается один из четырех равновероятных вариантов изменения:

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

Оператор скрещивания получает на вход две особи и выдает также две особи. Процесс скрещивания происходит следующим образом. Обозначим родительские особи *P1* и *P2*, а потомков – *S1* и *S2*.

Обозначим начальное состояние автомата *A* как *A.is*. Тогда для потомков *S1* и *S2* будет верно одно из двух: либо *S1.is* = *P1.is* и *S2.is* = *P2.is*, либо *S1.is* = *P2.is* и *S2.is* = *P1.is*, причем оба варианта равновероятны.

Опишем, как «устроены» переходы автоматов-потомков *S1* и *S2*. «Скрещивание переходов» может быть реализовано одним из двух способов. Выбор одного из них осуществляется при каждом скрещивании автоматов с помощью генератора случайных чисел.

Переход из состояния номер *i* в автомате *P1* по значению входной переменной «Перед муравьем есть еда» как *P1(i, 1)*, а по значению «Перед муравьем нет еды» – как *P1(i, 0)*. Аналогичный

смысл придадим обозначениям *P2(i, 0)* и *P2(i, 1)*. Тогда для переходов из состояния с номером *i* в автоматах-потомках *S1* и *S2* справедливо:

- либо *S1(i, 0) = P1(i, 0)*, *S1(i, 1) = P2(i, 1)* и *S2(i, 0) = P2(i, 0)*, *S2(i, 1) = P1(i, 1)*;
- либо *S1(i, 0) = P2(i, 0)*, *S1(i, 1) = P1(i, 1)* и *S2(i, 0) = P1(i, 0)*, *S2(i, 1) = P2(i, 1)*;
- либо *S1(i, 0) = P1(i, 0)*, *S1(i, 1) = P1(i, 1)* и *S2(i, 0) = P2(i, 0)*, *S2(i, 1) = P2(i, 1)*;
- либо *S1(i, 0) = P2(i, 0)*, *S1(i, 1) = P2(i, 1)* и *S2(i, 0) = P1(i, 0)*, *S2(i, 1) = P1(i, 1)*.

Все четыре варианта равновероятны.

В качестве основной стратегии формирования следующего поколения используется элитизм. При обработке текущего поколения отбрасываются все особи, кроме нескольких наиболее приспособленных. Доля выживающих особей постоянна для каждого поколения и является одним из параметров алгоритма.

Эти особи переходят в следующее поколение. После этого оно дополняется до требуемого размера следующим образом: пока оно не заполнено, выбираются две особи из текущего поколения, и они с некоторой вероятностью скрещиваются или мутируют. Обе особи, полученные в результате мутации или скрещивания, добавляются в новое поколение.

Кроме этого, если на протяжении достаточно большого числа поколений не происходит увеличения приспособленности, то применяются «малая» и «большая» мутации поколения. При «малой» мутации поколения ко всем особям, кроме 10% лучших, применяется оператор мутации. При «большой» мутации каждая особь либо мутирует, либо заменяется на случайно сгенерированную.

Число поколений до «малой» и «большой» мутации постоянно во время работы алгоритма.

Функция приспособленности особи (автомата) равна $F + \frac{200 - T}{200}$, где F – количество еды, которое съедает за 200 ходов муравей, поведение которого задается этим автоматом, а T – номер хода, на котором муравей съедает последнюю единицу еды.

На рис. 23 изображен граф переходов, построенный описанным алгоритмом после генерации 160 млн. автоматов. Этот граф переходов описывает поведение автомата с **семью** состояниями, который позволяет муравью съесть всю еду за 190 ходов.

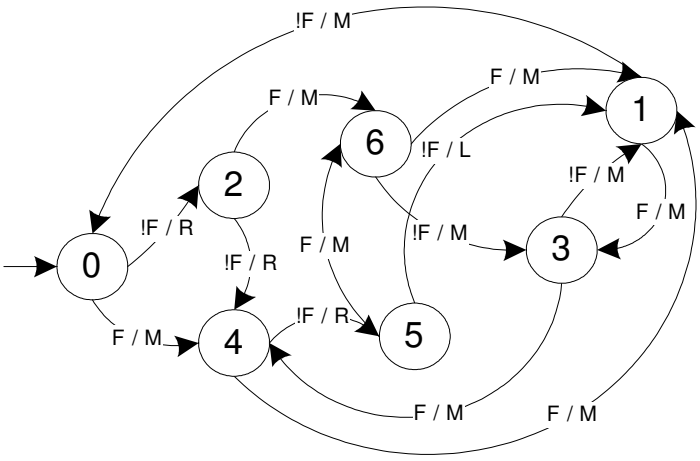

Рис. 23. Автомат, позволяющий муравью съесть всю еду

Отметим, что при другом запуске алгоритма после генерации 250 млн. автоматов был построен еще один автомат с семью состояниями.

С помощью описанного алгоритма генетического программирования также были построены автоматы из пяти (рис. 24) и шести состояний, позволяющие съесть 83 и 85 единиц еды соответственно.

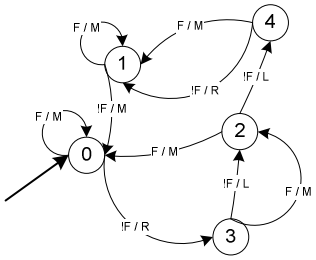

Рис. 24. Автомат из пяти состояний, построенный алгоритмом генетического программирования

Данная работа закрывает вопрос о нахождении автомата Мили с наименьшим числом состояний, решающего задачу об «Умном муравье».

1.2.3.3. Решение задачи «Умный муравей» при помощи автоматов Мура и систем вложенных автоматов

В отличие от всех предыдущих работ, в работе [76] задача об «Умном муравье» решается при помощи автоматов Мура, а также при помощи систем вложенных автоматов Мили.

Для генерации пары вложенных автоматов Мили и автомата Мура предлагается использовать один и тот же островной генетический алгоритм, но с различными структурами особей, фитнесс-функциями, операторами мутации и скрещивания.

В первом случае особь представляет собой пару вложенных автоматов Мили, один из которых будем называть внешним, а другой – внутренним. Описание каждого автомата состоит из номера начального состояния и описания каждого состояния.

Описание состояния содержит описание двух переходов – перед муравьем есть еда, и ее нет. Для внешнего автомата любой из этих переходов может быть не определен. В такой ситуации переход совершается во вложенном автомате, у которого, в свою очередь, определены все переходы.

Описание перехода состоит из номера состояния, в которое он ведет, и действия, выполняемого при выборе этого перехода.

Во втором случае особь представляет собой автомат Мура, который состоит из номера начального состояния и описания каждого состояния. Описание состояния содержит действие, выполняемое при переходе в состояние, и описание двух переходов – перед муравьем есть еда, и ее нет.

Описание перехода состоит из номера состояния, в которое он ведет.

Особь представляет собой объект на языке *Java*, вида:

```
class Automaton {
   int initialState;
   Transition[][] transition;
   char[] stateAction[]; // только для автоматов Мура
   Automaton nestedAutomaton;
}
```

Островной алгоритм традиционно состоит из следующих этапов:

- создание начального поколения;
- мутация и скрещивание;

- обмен особями между островами;
- отбор особей для формирования следующего поколения.

Все острова заполняются случайно сгенерированными автоматами. Все автоматы имеют заранее заданное число состояний.

Для генерации автоматов Мура применяется функция приспособленности вида:

$F - \frac{T}{200}$, где F – количество съеденной еды, T – номер последнего хода, на котором муравей съел еду.

Данная функция не подходит для генерации пары взаимодействующих автоматов Мили, так как генерируются слабо определенные внешние автоматы, функциональность которых, состоит в передаче управления вложенному автомату. Поэтому для пары вложенных автоматов Мили применялась функция приспособленности вида:

$F - \frac{T}{200} + C \cdot Z$, где F – количество съеденной еды, T – номер последнего хода, на котором муравей съел еду, Z – число посещенных состояний у внешнего автомата, C – некоторый коэффициент.

В качестве основной стратегии формирования следующего поколения используется элитизм. При рассмотрении текущего поколения отбрасываются все особи, кроме некоторой доли наиболее приспособленных – элиты. Эти особи переходят в следующее поколение. После этого оно дополняется в определенной пропорции случайными особями, особями из текущего поколения, которые мутировали, и результатами скрещивания особей из текущего поколения. Особи, которые дают потомство, определяются «в поединке»: выбираются две случайные пары особей, и более приспособленная особь в каждой из них становится одним из родителей. Эта эволюция происходит независимо на каждом из островов.

Через фиксированное число поколений каждый остров меняется с другим случайным числом случайно выбранных элитных особей.

Также через некоторое число поколений происходит *большая мутация*.

С помощью разработанного алгоритма построен автомат Мура с 10 состояниями, который позволяет муравью съесть всю еду за 198 шагов (рис. 25).

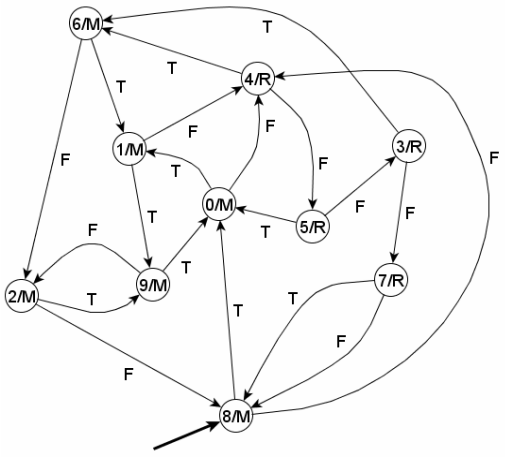

Рис. 25. Автомат, позволяющий муравью съесть всю еду за 198 шагов

При использовании рассматриваемого подхода и *девяти состояниях* в автомате Мура муравей за 198 ходов съедает только 86 единиц еды.

Авторам удалось сгенерировать систему автоматов (4, 6) (четыре состояния во внешнем автомате и шесть – во внутреннем), которая позволяет съесть муравью только 87 единиц еды за 185 ходов (рис. 26). Данный результат был получен при значении параметра *C* равного 0.01.

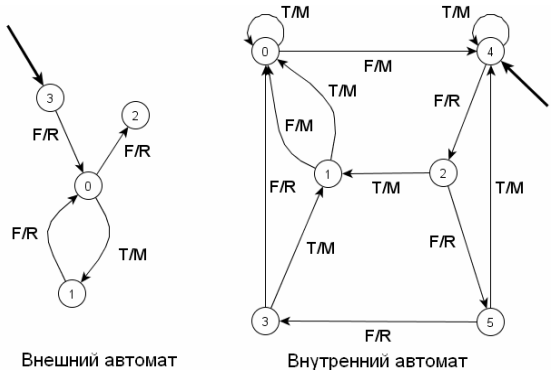

Рис. 26. Пример системы взаимодействующих автоматов Мили

1.2.3.4. Сравнение рассмотренных работ

В табл. 6 приведено сравнение рассмотренных работ.

Таблица 6. Сравнение работ, использующих автоматизированное построение автоматов

Название работы	Тип получаемого автомата	Метод высокоуровневого кодирования	Использование коэволюционного метода
Genetic programming. On the Programming of Computers by Means of Natural Selection.	Автомат Мили	Деревья выражений	нет
Применение генетических алгоритмов для построения автоматов с минимальным числом состояний для задачи об «Умном муравье».	Автомат Мили	Полные таблицы переходов	нет
Application of Genetic Algorithms for Construction of Moore Automaton and Systems of Interacting Mealy Automata in "Artificial Ant" Problem.	Автомат Мура и система автоматов Мили	Полные таблицы переходов	нет

1.2.4. Методы применения алгоритмов генетического программирования, в частности для генерации автоматов

1.2.4.1. Решение задачи классификации на основе графовых моделей

В статье [68] развивается идея эволюционирующих генетических операторов. Анализируются различные уровни мета-эволюции с использованием систем генетического программирования на основе графов. Система позволяет представлять как особей в пространстве поиска, так и сами генетические операторы в виде программ-графов, различающихся только набором операторов. Семь

вариантов мета-эволюции проверяются на трех реальных задачах классификации. Наиболее сложный вариант состоит из трех мета-уровней, где программа в виде графа на первом уровне осуществляет генетический поиск в базовом пространстве особей, программа-граф на втором уровне модифицирует программы первого уровня, а программы третьего мета уровня изменяют программы второго уровня и себя. На примерах показана возможность успешного применения мета-уровней. Можно выделить следующие отличия от исследования, проводимого по настоящему контракту.

- Суть работы – в эволюции только представленных графами генетических операторов.
- Основа представления программ в виде графов
- Специальное применение – задачи классификации.

По сути, статья [70] повторяет предыдущую, но имеет несколько смещенные акценты.

В данной работе представлена эволюция операторов генетического программирования методами генетического программирования. Более конкретно, мета-эволюция операторов рекомбинации в генетическом программировании на основе графов применяется и сравнивается с другими методами изменения и рекомбинации в генетическом программировании на основе графов. Демонстрируется, что непосредственное применение операторов рекомбинации к ним самим не дает желаемого результата. После ввода дополнительного уровня операторов рекомбинации (мета-уровня), которые модифицируют множество операторов рекомбинации, даже само-рекомбинация дополнительных операторов становится применимой. На задаче распознавания речи показано, что общая эффективность данной системы выше, чем в других вариантах генетического программирования на основе графов.

До публикации [70] основными формами представления генетических программ были деревья, линейные структуры и графы. В данной работе представлен новый тип структуры генетического программирования, полученный путем комбинирования автомата на верхнем уровне и линейных подпрограмм на нижнем, и названный *linear-graph*. Эта структура стала дальнейшим развитием разработанной ранее авторами данной работы структуры под названием *linear-tree*. В статье описывается предлагаемая структура, а также генетические операторы для ее модификации. Эффективность оценена в сравнении с линейными и древовидными программами. *Linear-graph* – это граф без циклов, каждая вершина которого представлена линейной программой. При входе в состояние запускается соответствующая подпрограмма. После выполнения подпрограммы в случае единственной исходящей дуги осуществляется переход, а при наличии нескольких дуг необходимая дуга выбирается в результате вычисления функции выбора перехода. Таким образом, эффективный анализ многих переменных при выборе перехода может быть проведен только вручную. Можно выделить следующие нетипичные для других работ отличия:

- не допускаются циклы в графе;
- выбор перехода выполняется на основе сравнения двух переменных или переменной с константой, точнее, в статье не указано ограничения числа анализируемых переменных, но в примерах используется сравнение не более двух переменных и проблема анализа значительного числа переменных не рассматривается.

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

В работе [71] рассматривается решение второй и третьей задачи.

Задача нумерации состояний

Приведем пример из указанной работы, иллюстрирующий важность решения задачи нумерации состояний для минимизации числа 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. 7 задает некоторый автомат Мили.

Таблица 7. 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. 27. Эта схема содержит три элемента `INV`, пять элементов `AND`, три элемента `OR` и два `D`-триггера.

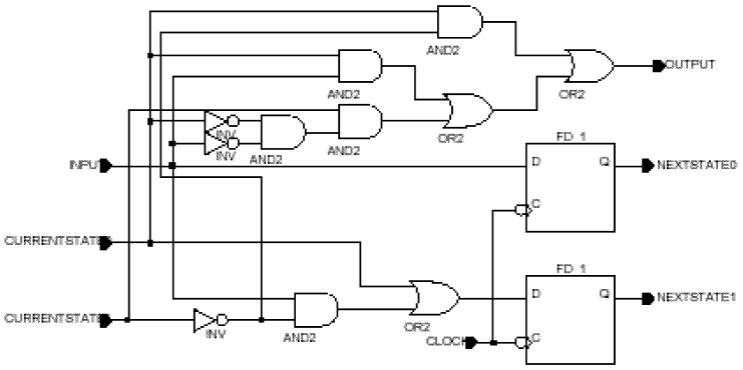

Рис. 27. Реализация автомата Мили при нумерации A_0

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. 28), содержащая всего два элемента `INV`, пять элементов `AND`, два элемента `OR` и два `D`-триггера.

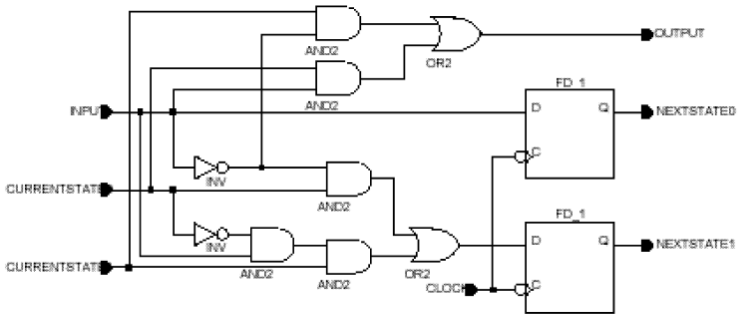

Рис. 28. Более экономичная реализация автомата Мили при нумерации A_1

Наивным решением задачи нумерации состояний является перебор всех возможных нумераций, построение минимальной схемы при заданной нумерации, выбор нумерации на которой достигается минимизация числа элементов схемы. Однако данный подход имеет экспоненциальную сложность по числу 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. 29.

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

Рис. 29. 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 [22]. 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. 30 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. 30. 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. Число 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. 31 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. 31. 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. 32

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

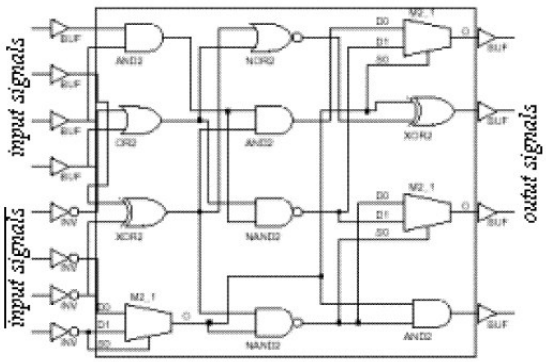

Рис. 32. Пример хромосомы, кодирующей схему

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.

1.2.4.3. Построение управляющей программы для человекоподобного робота (роботехника)

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

В работе [60] предлагается использовать традиционное генетическое программирование для эволюционного построения автомата, осуществляющего управление движением робота *Elvina.* Данный робот изображен на рис. 33.

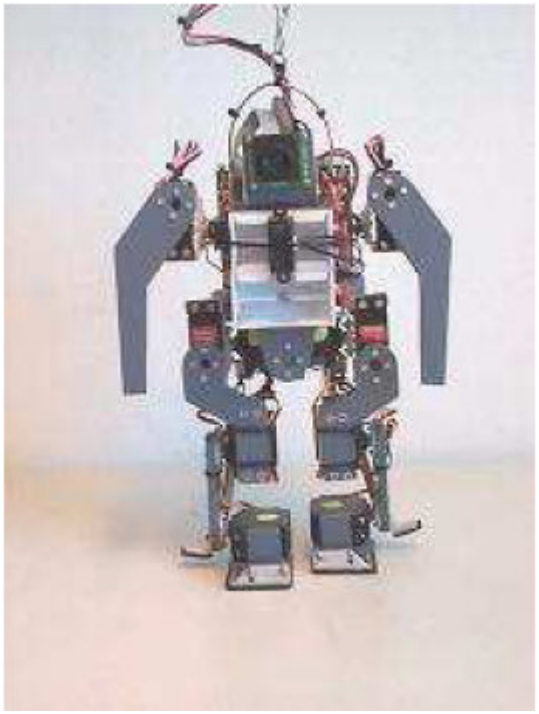

Рис. 33. Человекоподобный робот *Elvina*

Каждая из ног робота *Elvina* имеет пять степеней свободы, (одной из которых нельзя управлять). 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 переходов $f(j_1, j_2, ..., j_k) = [j_1', j_2', ..., j_k']$, которая принимает текущее состояние робота и возвращает состояние, в которое необходимо перейти. *Данная функция реализует автомат, 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 и конечные координаты частей тела. 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.

1.2.4.4. Построение контроллеров роботов (роботехника)

Область *эволюционной роботехники* (*Evolutionary robotics*, *ER*) занимается автоматическим созданием контроллеров роботов с использованием эволюционных алгоритмов. В большинстве случаев задача состоит в создании программного контроллера для мобильного робота. При этом контроллер обычно представляет собой программу, эволюционирующую с использованием генетического программирования. Контроллер обычно принимает данные с датчиков и генерирует управляющие сигналы для исполнительных устройств, возможно учитывая и модифицируя некоторое внутреннее состояние. Однако, архитектура полученного таким образом контроллера обычно равномерна и не имеет иерархии. Более того, эволюцию обычно делают ответственной за генерацию всего поведения робота, определение управляющих команд или действий и обработку входных данных до самого низкого уровня. К сожалению, физическое взаимодействие мобильных роботов с их окружением имеет очень сложную природу, еще сильнее усложняемую погрешностями датчиков и приводов. В результате, даже очень простая задача, как, например, обнаружение узнаваемого объекта и доставка в одно из двух помеченных мест в зависимости от его цвета, достаточно сложна для такого эволюционного решения. Кроме того, надежность полученного поведения робота сомнительна. Нельзя с уверенностью сказать, что детально описанное поведение будет иметь желаемый эффект. Однако возможность автоматической генерации контроллеров была бы важным достоинством эволюционной робототехники, если бы удалось обойти указанные трудности.

В работе [72] предлагается использовать эволюционирующий конечный автомат для выбора действий, каждое из которых описывается вручную или автоматически, но другим способом, и не изменяется в ходе эволюции. Ставится задача разработки эффективного метода автоматического проектирования контроллеров при заданных базовых элементов поведения робота, как на верхнем, так и на нижнем уровнях. Предлагается создавать распределенный механизм выбора действий, сопоставляя каждому модулю поведения отдельный управляющий автомат. Для экспериментов используется аппаратная платформа *LEGO RCX*. Отличия от нашего исследования следующие.

- Выбор перехода на основе анализа одной переменной-сообщения. Такую схему можно интерпретировать как учет только событий, но не защитных условий. Проблема решения задач с большим числом входных воздействий решается через применение множества параллельно работающих автоматов, каждый из которых отвечает за определенную деятельность.
- Специальное применение – роботехника.

Работа [73], по сути, повторяет предыдущую, но основное внимание в ней уделяется инкрементальным эволюционным алгоритмам, когда задача делится на части, которые решаются последовательно, сокращая тем самым пространство поиска и требования к ресурсам. Отмечаются следующие достоинства автоматического построения механизма выбора действий.

Автоматические методы могут рассматривать неочевидные решения, которые были бы пропущены человеком при ручном и полуавтоматическом проектировании. Мобильные роботы могут создаваться из расчета широкой применимости, а механизмы выбора действий могут потребоваться разные. В таких случаях может потребоваться генерация механизма выбора действий на основе описания задачи. Наличие возможности автоматического построения таких механизмов может уменьшить трудоммкость работ по сравнению с ручным построением каждого механизма. Кроме того, при ручном проектировании могут возникать трудности из-за недостаточного понимания имеющих место в реальности взаимодействий робота с окружением. Такие взаимодействия бывает сложно описать аналитически. Автоматическое проектирование способно анализировать характеристики взаимодействий робота более надежно, эффективно и точно.

1.2.4.5. Задача оптимизации

В работе [34] 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 в пространстве 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. 34.

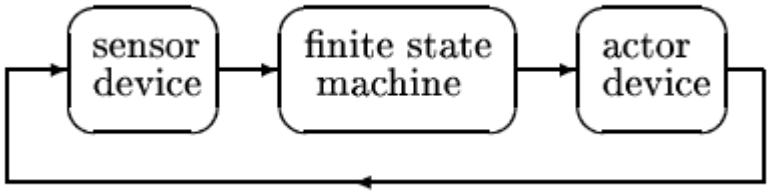

Рис. 34. Структура стратегии перемещения

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 $\sigma: R_+^2 \to X$. Эта 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 выходов 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 одновременно конструирует функции σ, δ, γ – дерево, являющееся особью генетического программирования, которое определяет сразу указанные 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. 35.

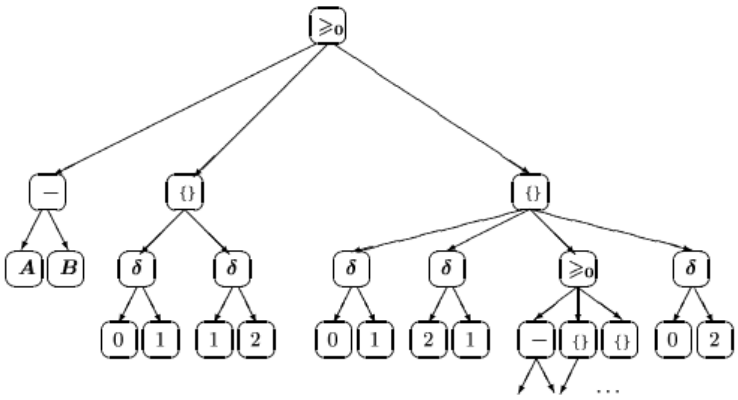

Рис. 35. Дерево выражений

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 на рис. 35 символом `{}`): 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. 36.

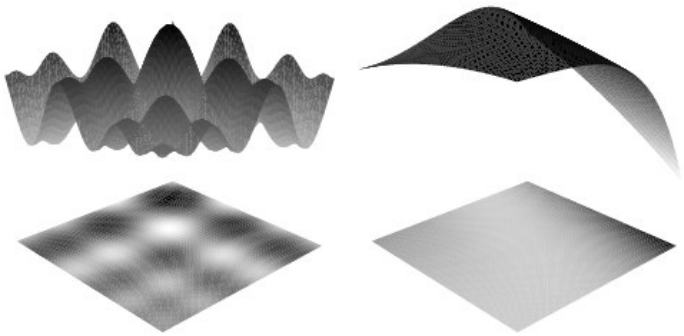

Рис. 36. Тестовые функции приспособленности Φ_1 и Φ_3

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.

1.2.4.6. 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. 37 приведен пример простого преобразователя двоичной записи числа `N` в двоичную запись числа `N + 1`. При этом двоичная запись числа подается с конца.

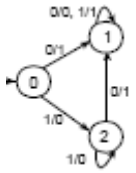

Рис. 37. Пример простого преобразователя

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 [54] 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 пометка ребер выполняется входными и выходными 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. 38 приведен пример рекомбинации.

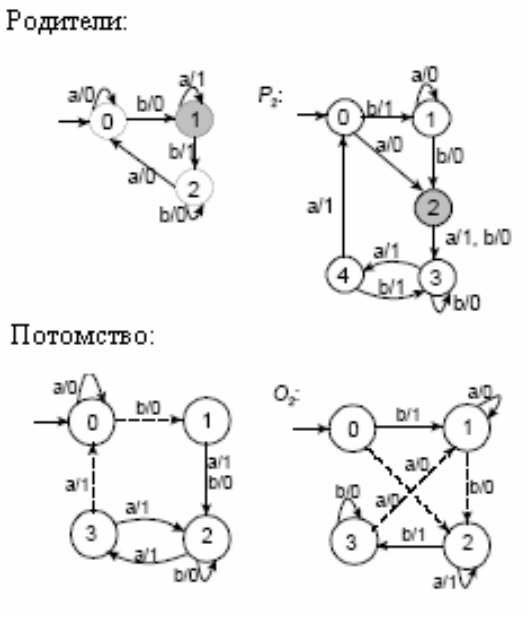

Рис. 38. Пример рекомбинации графов переходов

Операция мутации выполняется 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.

1.2.4.7. 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 [58] 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 [54].

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.

1.2.4.8. 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*. Автоматы этого класса отличаются от конечных автоматов тем, что они могут 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 [40] 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 [40] 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 состоит 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 число символов в стеке в конце разбора цепочек из множества примеров S. При этом

$stack*(M,w)=\frac{1}{1+\gamma}$, где γ – число 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 [40] введено ограничение на число 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 число символов, которые можно положить в стек фиксируется. Поэтому все строки имеют одинаковую длину. В работе [40] 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 [61] 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. 39.

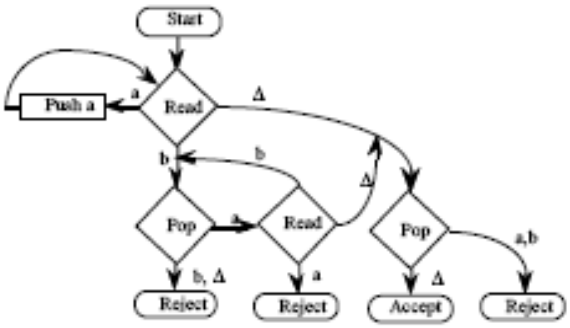

Рис. 39. Диаграмма переходов псевдосостояний

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. 39, 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.

1.2.4.9. Реализация систем со сложным поведением

В работе [75] используется автоматный подход для описания сущностей со сложным поведением. В соответствии с этим подходом сущность представляется в виде автоматизированного объекта – совокупности управляющего автомата и объекта управления. Объект управления характеризуется множеством вычислительных состояний, а также двумя наборами функций: множеством предикатов, отображающих вычислительные состояния в логические значения (истина или ложь), и множеством действий, позволяющих изменять вычислительные состояния.

Управляющий автомат определяется конечным множеством управляющих состояний, функцией переходов и функцией выходов (действий). Объект управления может быть как физическим, так и реализуемым программно. Несмотря на то, что предлагаемый подход может использоваться для объектов обоих типов, основное внимание будем уделять объектам, реализованным программно. При применении автоматного подхода поведение объекта управления при его проектировании можно сделать несложным: его предикаты и действия могут быть реализованы как простые, короткие функции, которые практически не содержат ветвлений.

При этом вся «логика» оказывается сосредоточенной в управляющем автомате. Описание логики в форме диаграммы переходов конечного автомата хорошо структурировано. Для некоторых задач эвристическое построение автомата или невозможно, или затруднительно. Кроме того, даже для простых задач автомат, построенный вручную, часто является неоптимальным. Эту проблему авторы предлагают решать методами генетического программирования. В качестве моделей вычислений в генетическом программировании чаще всего используют «низкоуровневые» модели (команды процессора, специализированные деревья, графы), которые имеют ограниченный набор элементарных операций (таких как, например, запись и чтение ячеек памяти, арифметические операции, вызовы подпрограмм и т.д.). Достоинство низкоуровневых моделей состоит в универсальности: с их помощью можно построить любую программу целиком, единообразно, вне зависимости от специфики решаемой задачи. Однако, такие модели обладают и серьезными недостатками. Во-первых, построенная автоматически программа из-за отсутствия высокоуровневой структуры редко бывает понятна человеку. Во-вторых, из-за того, что пространство допустимых программ в этом случае очень велико, генетическая оптимизация может потребовать большого времени. Устранить указанные недостатки можно путем перехода к высокоуровневым моделям вычислений. Для этого *в настоящей работе предлагается объединить автоматный и генетический подходы.* В качестве оптимизируемой модели вычислений предлагается использовать «автоматизированный объект». При этом реализация объекта управления производится вручную, а генетический алгоритм применяется для автоматического построения управляющего автомата. Автомат – «высокоуровневый» вычислитель, так как его элементарные операции (предикаты и действия) специфичны для конкретной задачи. Поэтому предлагаемый подход лишен указанных недостатков.

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

Объект O может управляться автоматом вида $A =< S, s_0, \Delta >$, где S – конечное множество управляющих состояний, s_0 – стартовое состояние, $\Delta : S \times \{0,1\}^n \to S \times Z^*$ – управляющая функция.

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

В каждом состоянии значимым является лишь определенный, небольшой набор предикатов, остальные же не влияют на значение управляющей функции. Именно это свойство позволяет существенно сократить размер описания состояний. Кроме того, использование этого свойства в процессе оптимизации позволяет получить автомат, более похожий на построенный вручную, а, следовательно, более понятный человеку. Один из подходов к сокращению размера хромосом состояний – ограничить число значимых в состоянии предикатов некоторой константой r. К таблице, задающей сужение управляющей функции на данное состояние, в этом случае добавляется битовый вектор, описывающий множество значимых предикатов. Полученное представление хромосомы назовем сокращенной таблицей состояния.

Число строк сокращенной таблицы 2^r, однако, константа r обычно невелика. Как показывает опыт, для большинства автоматизированных объектов достаточно, чтобы выполнялось неравество $r \le 5$. По сравнению с представлением в виде полной таблицы, в этом случае добавилась возможность мутации множества значимых предикатов. При этом каждый из них с некоторой вероятностью заменяется другим предикатом, который не принадлежит множеству. Мутация самой сокращенной таблицы происходит так же, как мутация полной таблицы. Скрещивание сокращенных таблиц – наиболее сложный из используемых алгоритмов. Поскольку родительские хромосомы, представленные сокращенными таблицами, могут иметь разные множества значимых предикатов, сначала необходимо выбрать, какие из этих предикатов будут значимы для хромосом детей. После этого заполняются таблицы обоих детей. В предлагаемой реализации оператора скрещивания на значения каждой строки таблицы ребенка влияют значения нескольких строк родительских таблиц. При этом конкретное значение, помещаемое в ячейку таблицы ребенка, определяется «голосованием» всех влияющих на нее ячеек родительских таблиц. Предлагаемый подход был апробирован на решении ряда задач, и показал свою эффективность.

1.2.4.10. Сравнение рассмотренных работ

В табл. 8 приведено сравнение рассмотренных работ.

Таблица 8. Сравнение работ, описывающих метода применения генетического программирования

Название работы	Решаемая задача	Автоматный подход	Высокоуровневое кодирование	Использование коэволюционного метода
Empirical Analysis of Different Levels of Meta-Evolution	Задача классификации	Нет	Графовая модель	Да
Meta-Evolution in Graph GP	Задача классификации	Нет	Графовая модель	Да
An Evolutionary Approach	Проектирование логических схем	Да	Нет	Нет
Evolutionary Based Approach for Control Programming of Humanoids	Построение управляющей программы для человекоподобного робота	Да	Нет	Нет
Simulated evolution of distributed FSA behaviour-based arbitration	Построение контроллеров роботов	Да	Нет	Нет
Evolving automatons for distributed behavior arbitration	Построение контроллеров роботов	Да	Нет	Да
Empirical Analysis of Different Levels of Meta-Evolution	Задача оптимизации	Да	Дерево разбора	Нет
The Induction of Finite Transducers Using Genetic Programming	Построение автоматических преобразователей	Да	Графовая модель	Нет
Genetic Algorithms for Automated Negotiations: A FSM-based Application Approach	Автоматические переговоры	Да	Графовая модель	Нет
ECHO: Explorations of Evolution in a Miniature World	Распознавание нерегулярных языков	Да	Графовая модель	Нет
Применение генетического программирования для реализации систем со сложным поведением	Реализация систем со сложным поведением	Да	Сокращенные таблицы переходов	Нет

1.3. Программы-симуляторы летательных аппаратов

Состояние развития симуляторов полета, определяет возможности и ограничения, с которыми приходится сталкиваться при исследованиях систем автоматического пилотирования.

Исследование применения генетических алгоритмов при построении систем управления летательными аппаратами предъявляет ряд особых требований к симуляторам. К примеру, генетический алгоритм не требует отображения внешнего облика летательного аппарата в режиме реального времени, по причине того, что симуляция без визуального интерфейса с вычислительной точки зрения работает намного быстрее обычной. Кроме того, точная симуляция кинематики и аэродинамики является фундаментально важной для применения генетических алгоритмов.

В настоящее время существует две основных категории симуляторов: игровые средства и симуляторы полета.

В игровых симуляторах важным аспектом является привлекательная визуализация, позволяющая игроку почувствовать себя в кабине летательного аппарата. При этом моделированию реальных сил действующих в полете уделяется второстепенная роль.

Симуляторы полета сосредотачивают свое внимание непосредственно на моделировании полета, расчету реальных сил и моментов, действующих на отдельные части летательного аппарата. Безусловно, в симуляторах полета также есть некоторый акцент на визуальных аспектах, но основные усилия и внимание направлены на аэродинамику и полетные факторы, существующие в реальном мире.

Современные симуляторы предоставляют возможность исследователям тестировать и оценивать результаты их работ при помощи большого набора параметров, рассчитываемых во время симуляции. Понятное трехмерное изображение окружающей среды с учетом физических и кинематических факторов открывает новые перспективы для робототехнических исследований. В случае, если основное внимание симулятора сосредоточено на аэродинамике, то средства визуализации обычно не так развиты.

Существует огромное разнообразие коммерческих симуляторов полетов. Графика в современных симуляторах полета очень реалистична. Однако, исследователей больше беспокоит динамика полета самолета. Поэтому оценка некоторых коммерческих представлена ниже.

1.3.1. Симулятор *Microsoft Flight Simulator*

Серия симуляторов полета *Microsoft Flight Simulator* [6] последних версий, вероятно, является наиболее реалистичной на рынке по своим графическим показателям (рис. 40), во многом благодаря использованию новой технологии *DirectX 10*.

Рис. 40. Визуализация в симуляторе *Microsoft Flight Simulator*

Аэродинамическая модель полета основывается на наборе таблиц и не зависит от подсистемы визуализации.

Моделирование отказов в работе летательного аппарата включает в себя симуляцию отказа оборудования, но без изменения причин, по которым происходит отказ оборудования.

Microsoft Flight Simulator представляет интерфейс *SimConnect*, а также архитектуру *FSUIPC*, обеспечивающую доступ к функциям и переменным, которые позволяют разработчикам создавать обновления для добавления или замены функциональных возможностей (рис. 41).

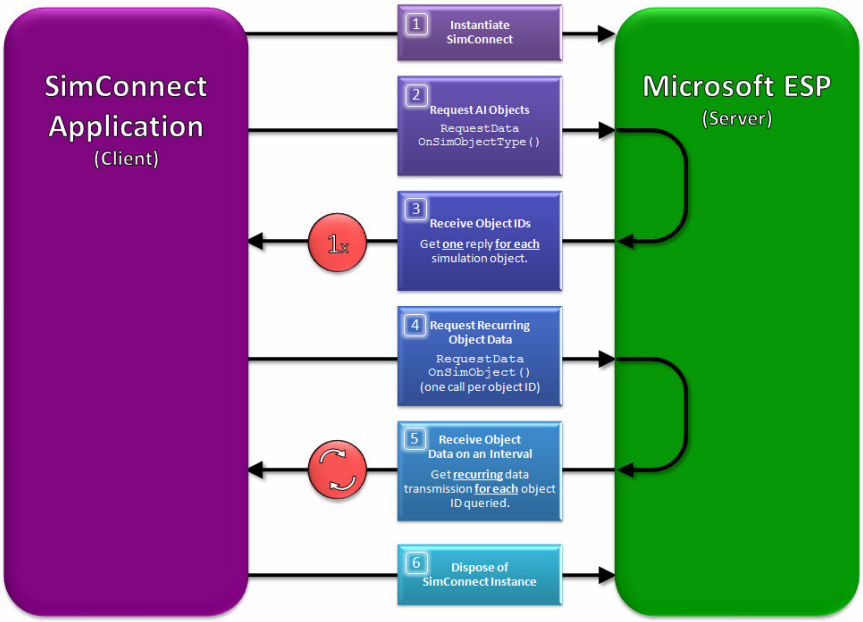

Рис. 41. Общая архитектура *FSUIPC*

1.3.2. Симулятор *X-Plane*

Симулятор *X-Plane* [7] был создан Остином Майером для симуляции самолета *Piper Archer* (рис. 42). Первая публичная версия под номером пять появилась в продаже в 2001 г.

Версия 8.00 появилась в конце 2004 года. После этого она несколько раз значительно модернизировалась. Новая версия, 8.60 вышла в феврале 2007 г. Разработчики продолжают совершенствовать аэродинамику и физическую модель, а основные усилия сейчас направлены на улучшение графики и моделирования Земли (в 2005 г. с версией 8.20 в продаже появилась версия *Global Scenery* со спутниковыми снимками земли высокого разрешения), улучшение функциональности систем летательных аппаратов (гидравлика, электрика, шасси, антиобледенение, герметизация и другие). Окончательная доработка восьмой версии – версия 8.64. В начале 2008 г. появилась бета-версия *X-Plane 9.00*. Всего бета-версий было 25, а потом – три релиз-кандидата. 16.04.2008 г. вышла окончательная версия *X-Plane 9.00*.

Рис. 42. Визуализация в симуляторе *X-Plane*

X-Plane использует метод конечных элементов для расчета динамики полета. В процессе работы симулятор, разбирает самолет на множество маленьких элементов, а затем определяет действующие на каждый элемент силы, которые потом преобразуются в ускорения, скорости и положения.

X-Plane также имеет возможность моделировать отказы, но как и у симуляторов *FSX*, невозможно менять способы, которыми оборудование выходит из строя.

Важным достоинством симуляторов *X-Plane* является их профессиональное использование и одобрение *FAA* для тренировок полетов на воздушном транспорте и получении сертификатов.

1.3.3. Симулятор *FlightGear*

FlightGear [8] – свободно распространяемый, кроссплатформенный симулятор. Целью проекта является создание симулятора для использования в исследовательских и научных применениях (рис. 43, 44).

Симулятор позволяет получать доступ к большому числу внутренних переменных состояния. Это позволяет удаленно управлять *FlightGear* при помощи внешнего сценария, а также использовать внешний модуль динамики полета (включая аппаратный автопилот).

Рис. 43. Приборная панель симулятора *FlightGear*

FlightGear позволяет пользователям выбирать между тремя основными моделями динамики полета. Более того, имеется возможность добавлять новые динамические модели, например, из среды *MATLAB*.

Основными моделями являются *JSBSim* и *YASim*.

Рис. 44. Визуализация в симуляторе *FlightGear*

Модель *JSBSim* является моделью динамики 2000 г. Она разработана Джоном Бернддом.

Модель *YASim* – другая модель *FDM*, использующая различные методы вычисления. Введена в 2002 г., разработана Энди Россом.

Модель *UIUC* – еще одна модель *FDM*, разработанная в *UIUC* Группой Прикладной Аэродинамики в Университете Иллинойса (UIUC Applied Aerodynamics Group at University of Illinois).

Гибкость этого симулятора позволяет исследователям выбирать комбинацию модулей, которые наиболее соответствуют целям и задачам исследования. В зависимости от целей можно сконцентрироваться на разработке или усовершенствовании либо графического содержания модели самолета, либо на создании систем автоматического управления летательным аппаратом. Возможно также создание мультиагентных систем, способных координировать свои действия.

1.3.4. Симулятор *AeroSim*

Симулятор *AeroSim* [9] представляет собой набор программ для пакета *MATLAB* и предоставляет полный набор инструментов для разработки нелинейных моделей динамики полета с шестью степенями свободы. Модели для пакета *Simulink* (рис. 45) включают в себя следующие блоки:

- блок нелинейного движения;
- блок линейной аэродинамики;
- расчета инерции летательного аппарата, включая изменение веса в полете (из-за расхода горючего);
- модель атмосферы, включая порывы ветра и турбулентность фон Кармана;
- блок магнитной модели земли (рис. 46);
- блок гравитационной модели земли.

В дополнении к данным блокам симулятор предоставляет базовые аналоговые сенсоры и нелинейные модели воздействия, блоки преобразования величин, блоки преобразования координат.

Симулятор также имеет несколько заранее реализованных моделей летательных аппаратов.

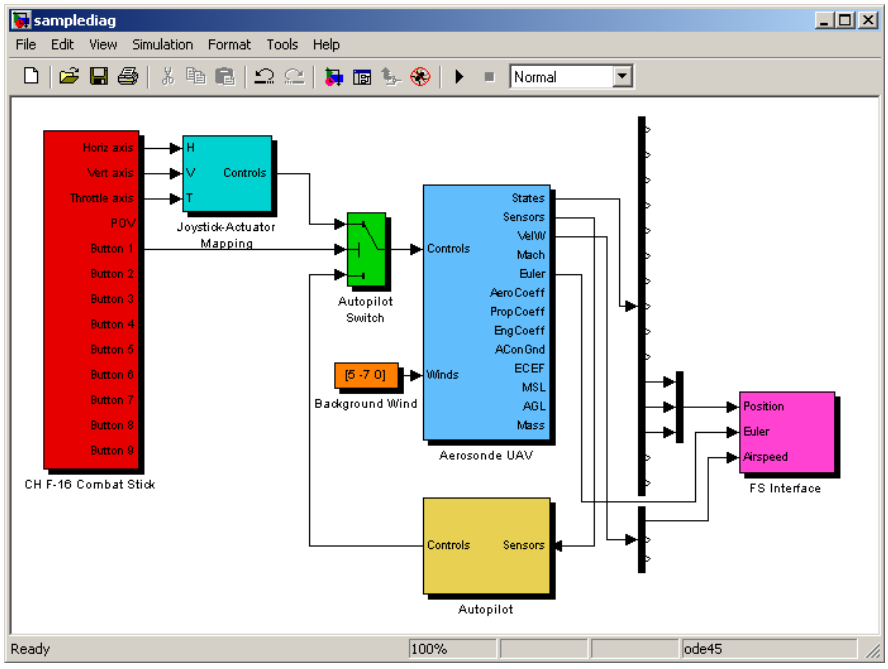

Рис. 45. Модель *Simulink* в симуляторе *AeroSim*

В связи с тем, что симулятор использует только базовые возможности пакета *Simulink*, существует возможность трансформировать модель летательного аппарата в исходный код на языке *C*.

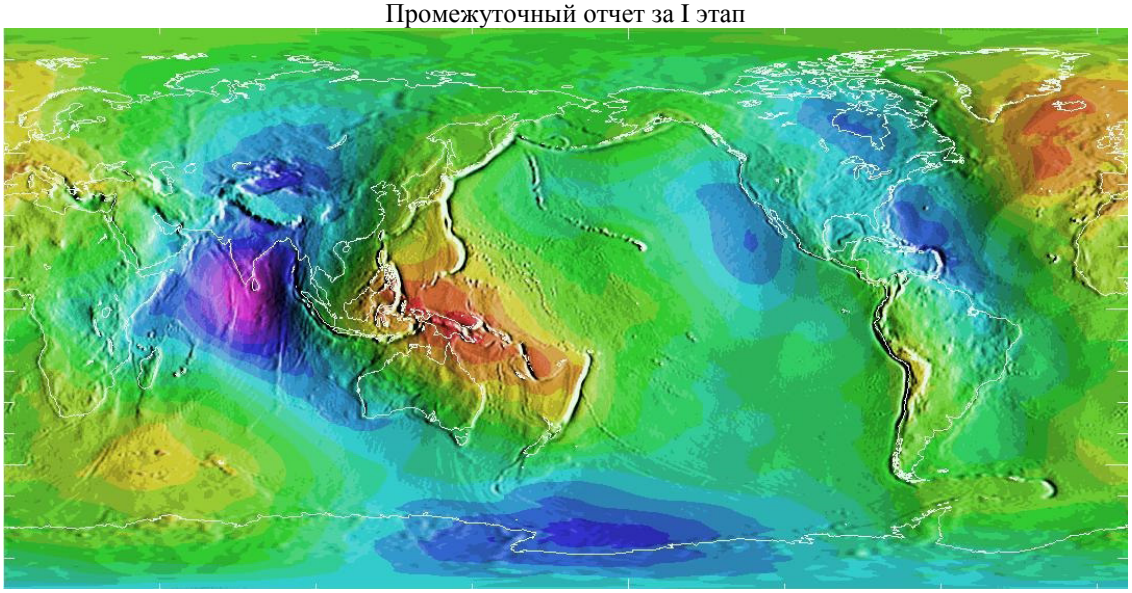

Рис. 46. Модель земли в симуляторе *AeroSim*

Динамика полета описывается нелинейными уравнениями движения шестого порядка, интегрирование которых по времени дает возможность работать симулятору. Уравнения положения реализованы с применением системы Родрига-Гамильтона. Это существенно повышает вычислительную эффективность и позволяет избежать особых точек.

Блок пересчета параметров Родрига-Гамильтона в углы Эйлера и направляющие косинусы также присутствует в пакете.

Симулятор позволяет считывать показания джойстика в случае, если необходимо ручное управление.

Результаты работы симулятора могут быть отображены в пакете *FlightGear* или *Microsoft Flight Simulator*.

1.3.5. Симулятор *Piccolo*

Симулятор *Piccolo* [10] – широко известная система автоматического пилотирования для небольших самолетов и вертолетов, созданная компанией *CloudCap Technology*.

Система успешно продается в качестве компонентов аппаратного обеспечения, которые могут быть установлены на маленьком самолете для обеспечения автономного полета. Компания также создала программный симулятор для тестирования оборудования с виртуальным объектом управления (рис. 47).

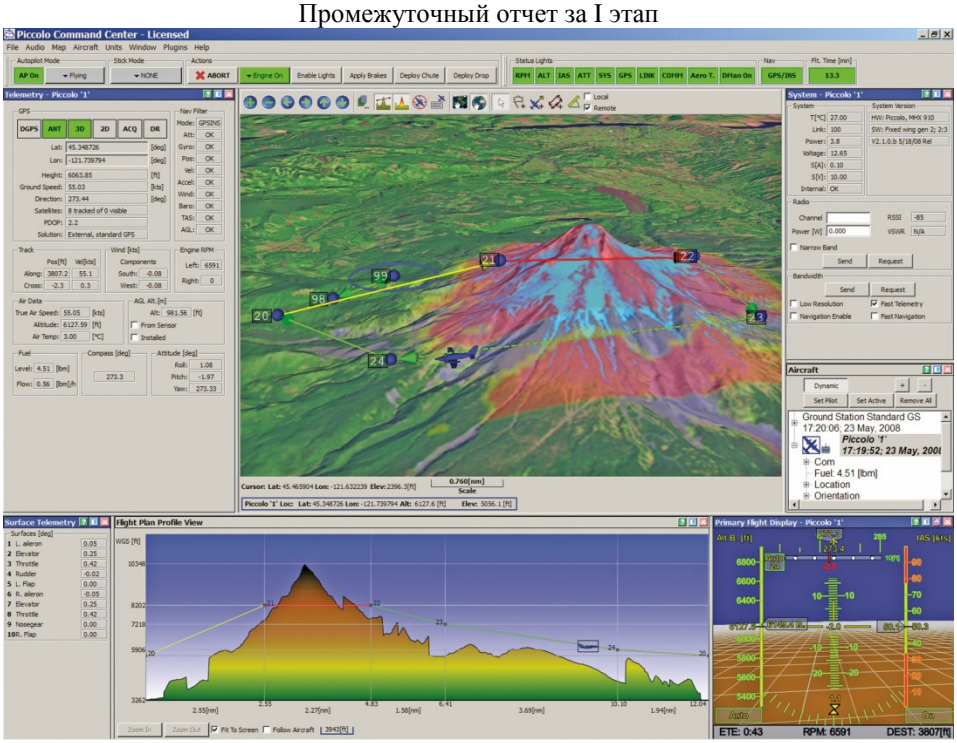

Рис. 47. Центр управления симулятором *Piccolo*

Графический компонент задействован на минимальном уровне, но с точки зрения динамики полета симулятор ведет себя очень реалистично при симулировании множества сил, действующих в полете. Существует возможность вывода данных симуляции в *Microsoft Flight Simulator* и в *FlightGear*.

1.3.6. Сравнение программ-симуляторов летательных аппаратов

В табл. 9 приведено сравнение рассмотренных программ-симуляторов по пяти параметрам:

- визуализация;
- достоверность симуляции динамики полета;
- статус исходного кода;
- качество интерфейса программирования;
- удобство моделирования.

Таблица 9. Сравнение программ-симуляторов летательных аппаратов

Название симулятора	Визуализация	Достоверность симуляции динамики полета	Статус исходного кода	Качество интерфейса программирования	Удобство моделирования
Microsoft Flight Simulator	Отлично	Хорошо	Закрыт	Отлично	Средствами языков высокого уровня
X-Plane	Отлично	Хорошо	Закрыт	Хорошо	Средствами языков высокого уровня
FlightGear	Отлично	Отлично	Открыт	Отлично	Средствами языков высокого уровня
AeroSim	Средствами FlightGear	Отлично	Открыт	Отлично	Средствами языка *C* и графически (в среде *Simulink*)
Piccolo	Хорошо	Хорошо	Закрыт	Плохо	Встроенный редактор

Выводы по главе 1

1. Сущности со сложным поведением отличаются от сущностей с простым поведением тем, что в ответ на одно и то же входное воздействие они могут выработать в зависимости от своего состояния различные выходные воздействия. Для проектирования и реализации сущностей со сложным поведением может эффективно применяться автоматное программирование.
2. В случае, когда автоматы описывают системы со сложным поведением, их проектирование является нетривиальной и трудоемкой 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.
3. Генетические алгоритмы и генетическое программирование успешно применяются для построения конечных автоматов для систем со сложным поведением. При этом для нескольких задач с помощью генетических алгоритмов могут быть построены решения, которые превосходят построенные вручную. В частности, с помощью генетических алгоритмов успешно строят системы управления человекоподобными роботами.
4. Для моделирования одиночного летательного аппарата на следующих этапах работы будет применяться один из следующих симуляторов: *Microsoft Flight Simulator, X-Plane, FlightGear, AeroSim*. Симулятор *Piccolo* не будет применяться, так как его интерфейс программирования не обладает достаточным качеством.

2. ВЫБОР И ОБОСНОВАНИЕ ОПТИМАЛЬНОГО ВАРИАНТА НАПРАВЛЕНИЯ ИССЛЕДОВАНИЙ

В настоящем разделе представлено обоснование оптимального варианта проведения исследований.

На основании результатов выполненного аналитического обзора предлагается следующий вариант направления исследований: будет проведена разработка новых структур хромосом алгоритмов генетического программирования, методов скрещивания и мутации для этих структур хромосом, методов вычисления функции приспособленности, а также будет проведено экспериментальное исследование разработанных методов на задаче управления беспилотным летательным аппаратом.

При этом экспериментальные исследования будут проводиться как на задаче управления одиночным летательным аппаратом, так и на задаче группового управления. После выполнения экспериментальных исследований будет проведена корректировка разработанных методов. По итогам работ будет зарегистрирована программа для ЭВМ и опубликованы четыре статьи в журналах из перечня ВАК.

Эксперименты, проведенные в рамках предыдущих работ по построению управляющих автоматов с помощью генетического программирования, показали, что существующие методы недостаточно эффективны для решения задачи построения системы управления беспилотным летательным аппаратом. Методы, которые будут разрабатываться в рамках работ по настоящему Государственному контракту, будут основываться на методах, рассмотренных в рамках аналитического обзора (первая глава настоящего отчета).

При проведении теоретических исследований будет разработан метод генетического программирования, основанный на представлении функции переходов управляющего конечного автомата в виде линейных бинарных графов. Представление в виде линейных бинарных графов интересно тем, что размер линейного бинарного графа прямо пропорционален числу букв в соответствующей булевой формуле.

Также, при проведении теоретических исследований особое внимание будет уделяться разработке методов, позволяющих учесть знания человека при построении управляющих конечных автоматов. Для этого будет разработан метод построения автоматов на основе обучающих примеров, а метод, позволяющий задавать ограничения в виде формул темпоральной логики, которые будут использоваться для верификации управляющего автомата в процессе вычисления функции приспособленности.

Кроме этого, при проведении теоретических исследований будут разработаны методы, позволяющие осуществлять построение автоматов в несколько этапов. Реализация этой идеи, весьма схожей с идеей динамического программирования, позволит разбить процесс построения управляющего конечного автомата на две стадии: построение состояний и построение автомата на основе этих состояний.

При проведении экспериментальных исследований разработанные методы будут сравниваться на примере решения задачи построения системы управления моделью беспилотного летательного аппарата. Экспериментальное исследование будет проводиться с использованием одной из программ-симуляторов летательных аппаратов, рассмотренных в рамках аналитического обзора (первая глава настоящего отчета). На основании результатов экспериментов методы генетического программирования будет модифицированы. Экспериментальное исследование на такой достаточно сложной задаче позволит более четко выделить сильные и слабые стороны разработанных методов.

Выводы по главе 2

1. Выбрано и обосновано оптимальное направление проведения исследований.

3. ПЛАН ПРОВЕДЕНИЯ ТЕОРЕТИЧЕСКИХ И ЭКСПЕРИМЕНТАЛЬНЫХ ИССЛЕДОВАНИЙ

В настоящем разделе представлен план проведения теоретических и экспериментальных исследований.

3.1. План проведения первого этапа теоретических и экспериментальных исследований

На первом этапе проведения исследований планируется провести следующие работы:

- разработка метода генетического программирования, основанного на использовании линейных бинарных графов для представления функции переходов автоматов (28.09.2009–14.10.2009):
  - разработка метода представления управляющего конечного автомата с помощью линейных бинарных графов;
  - разработка алгоритмов реализации операций мутации и скрещивания;
- программная реализация метода генетического программирования, основанного на использовании линейных бинарных графов для представления функции переходов автоматов (28.09.2009–14.10.2009):
  - программная реализация метода представления управляющего конечного автомата с помощью линейных бинарных графов;
  - программная реализация алгоритмов реализации операций мутации и скрещивания;
- экспериментальное исследование метода генетического программирования, основанного на использовании линейных бинарных графов для представления функции переходов автоматов на задаче построения автомата управления системой со сложным поведением (14.10.2009 – 21.10.2009).

Результаты работ первого этапа:

- метод генетического программирования для построения конечных автоматов, основанный на использовании линейных бинарных графов.
- научно-технический отчет;
- протоколы эксперментов.

3.2. План проведения второго этапа теоретических и экспериментальных исследований

На втором этапе исследований планируется провести следующие работы:

- разработка и программная реализация метода построения управляющих конечных автоматов на основе обучающих примеров с помощью генетического программирования (01.01.2010 – 28.02.2010):
  - разработка метода представления управляющих конечных автоматов в виде хромосом алгоритма генетического программирования;
  - разработка алгоритмов реализации операций мутации и скрещивания;
  - разработка алгоритма расстановки пометок на переходах автомата на основе обучающих примеров;
- разработка и программная реализация метода двухэтапного генетического программирования (01.01.2010 – 28.02.2010):

  - разработка методов генетического программирования, осуществляющих построение репозитория состояний;
  - разработка методов генетического программирования, осуществляющих построение автоматов на основе репозитория состояний;
- разработка и программная реализация метода применения верификации моделей при вычислении функции приспособленности (01.03.2010 – 31.03.2010);
- разработка программной модели беспилотного летательного аппарата и экспериментальное исследование разработанных методов на задаче построения системы управления моделью беспилотного летательного аппарата (01.05.2010 – 30.06.2010).

По итогам второго этапа работ планируется подготовить для публикации две статьи в журналах из перечня ВАК.

Результаты работ второго этапа:

- метод построения управляющих конечных автоматов на основе обучающих примеров с помощью генетического программирования;
- метод двухэтапного генетического программирования;
- метод применения верификации моделей при вычислении функции приспособленности;
- две статьи в журналах из перечня ВАК;
- протоколы экспериментов.

3.3. План проведения третьего этапа теоретических и экспериментальных исследований

На третьем этапе проведения исследований планируется провести следующие работы:

- корректировка разработанных методов с учетом результатов проведенных экспериментов (01.01.2011 – 31.01.2011);
- экспериментальное исследование модифицированных методов на задаче построения системы группового управления моделями беспилотных летательных аппаратов (01.02.2011 – 31.03.2011);
- обобщение и оценка результатов исследований. Оценка полноты решения задачи и достижения поставленных целей (01.05.2011 – 19.07.2011);
- обобщение и оценка результатов исследований – сопоставление и обобщение результатов анализа научно-информационных источников и теоретических и экспериментальных исследований (01.05.2011 – 19.07.2011);
- обобщение и оценка результатов исследований – оценка эффективности полученных результатов в сравнении с современным научно-техническим уровнем (01.05.2011 – 19.07.2011);
- обобщение и оценка результатов исследований – разработка рекомендаций по использованию результатов НИР при создании научно-образовательных курсов (01.05.2011 – 19.07.2011).

Результаты работ третьего этапа:

- модифицированные методы генетического программирования;
- две статьи в журналах из перечня ВАК, содержащие основные результаты НИР;
- свидетельство о регистрации программы для ЭВМ;
- рекомендации по использованию результатов НИР при создании научно-образовательных курсов;
- протоколы экспериментов;

- научно-технический отчет.

Выводы по главе 3

1. Разработан план проведения теоретических и экспериментальных исследований.
2. На первом этапе теоретических исследований будет проведена разработка метода генетического программирования, основанного на использовании линейных бинарных графов для представления функции переходов автоматов.

4. РАЗРАБОТКА И ПРОГРАММНАЯ РЕАЛИЗАЦИЯ МЕТОДА ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ, ОСНОВАННОГО НА ИСПОЛЬЗОВАНИИ ЛИНЕЙНЫХ БИНАРНЫХ ГРАФОВ ДЛЯ ПРЕДСТАВЛЕНИЯ ФУНКЦИИ ПЕРЕХОДОВ АВТОМАТОВ

В настоящем разделе описывается метод генетического программирования, основанного на использовании линейных бинарных графов для представления функции переходов автоматов. Кроме этого, приводится его программная реализация.

4.1. РАЗРАБОТКА МЕТОДА ПРЕДСТАВЛЕНИЯ УПРАВЛЯЮЩЕГО КОНЕЧНОГО АВТОМАТА С ПОМОЩЬЮ ЛИНЕЙНЫХ БИНАРНЫХ ГРАФОВ

В настоящем разделе описывается метод представления функции переходов управляющего автомата с помощью линейных бинарных графов. Этот метод предложен в работе [64].

4.1.1. Линейные бинарные графы

Разрешающая диаграмма [62] является удобным способом задания булевой функции, зависящей от конечного числа булевых переменных. Она представляет собой помеченный ацикличный ориентированный граф, в котором выделяют вершины двух типов:

- нетерминальные узлы;
- терминальные узлы.

При этом один из нетерминальных узлов является стартовым. Все остальные узлы достижимы из начального. Из каждого нетерминального узла выходит два ребра. Из терминальных узлов ребер не выходит. Метки в графе расставляются по следующим правилам:

- нетерминальные узлы помечаются названиями переменных;
- терминальные узлы помечаются значениями функции;
- ребра помечаются значениями переменных.

Для определения значения функции по значениям переменных необходимо пройти путь от стартового до терминального узла, и сформировать значение, которым помечен полученный терминальный узел. При этом из вершины, помеченной переменной x, переход производится по тому ребру, которое помечено тем же значением, что и значение переменной x. На рис. 48 приведен пример разрешающей диаграммы, реализующей булеву функцию $f = a \oplus b \oplus c$.

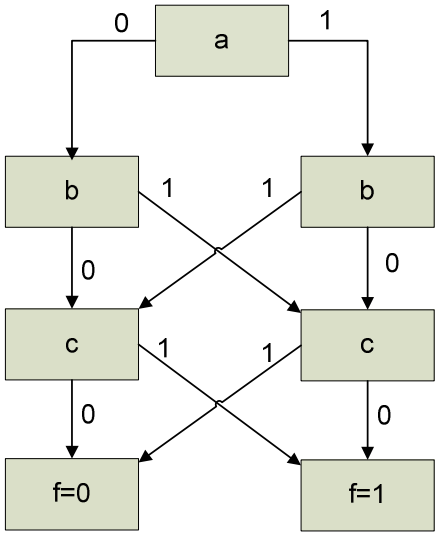

Рис. 48. Пример разрешающей диаграммы

Важным частным случаем разрешающих диаграмм являются линейные разрешающие диаграммы или линейные бинарные графы. Узлы линейного бинарного графа могут быть пронумерованы таким образом, что из каждого нетерминального узла одно из ребер ведет в узел со следующим номером. При этом число элементов, необходимых для реализации булевой формулы бинарным линейным графом, линейно зависит от числа букв в булевой формуле [67].

Пример линейного бинарного графа, реализующего булеву функцию $f = ab \vee c$, приведен на рис. 49.

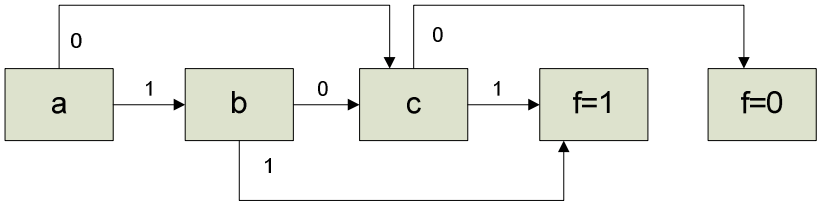

Рис. 49. Пример линейного бинарного графа

4.1.2. Представление дискретной функции булевых переменных линейным бинарным графом

Определим способ задания дискретной функции булевых переменных линейным бинарным графом. Пусть дискретная функция задана системой правил вида $f_i(X) = 1 \to g(X) = c_i$, где $X = (x_1 ... x_n)$ – вектор булевых переменных, $g(X)$ – дискретная функция, $f_i(X)$ – булевы функции, образующие полную и непротиворечивую систему, c_i – значения дискретной функции. Пусть для примера функция определяется следующей системой правил:

$$\begin{cases} ab \vee c \to L \\ \overline{a}\,\overline{c} \to R \\ a\overline{b}\,\overline{c} \to F \end{cases}$$

Теперь построим для каждой из булевых функций системы по линейному бинарному графу (рис. 50):

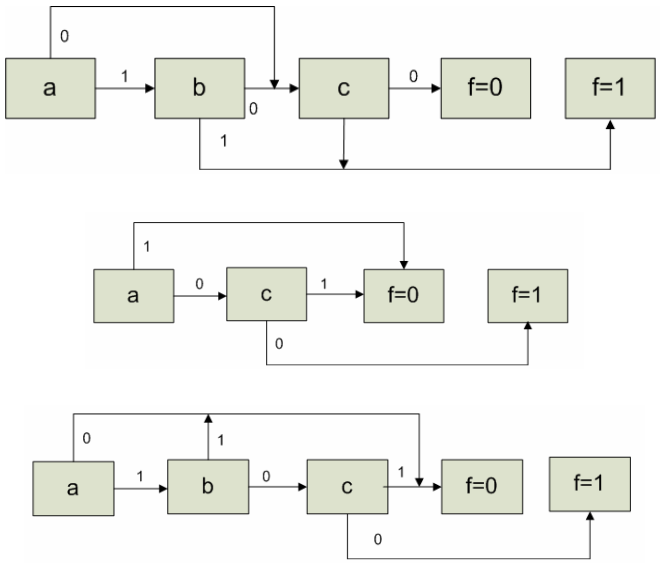

Рис. 50. Линейные бинарные графы функций системы

Заметим, что терминальные узлы, соответствующие значениям 0 и 1 всегда можно поменять местами, не нарушая свойства линейности бинарного графа. Перерисуем, сохраняя планарность, графы таким образом, чтобы терминальный узел, помеченный значением «ноль», будет стоять перед узлом, помеченным значением «единица» (рис. 51).

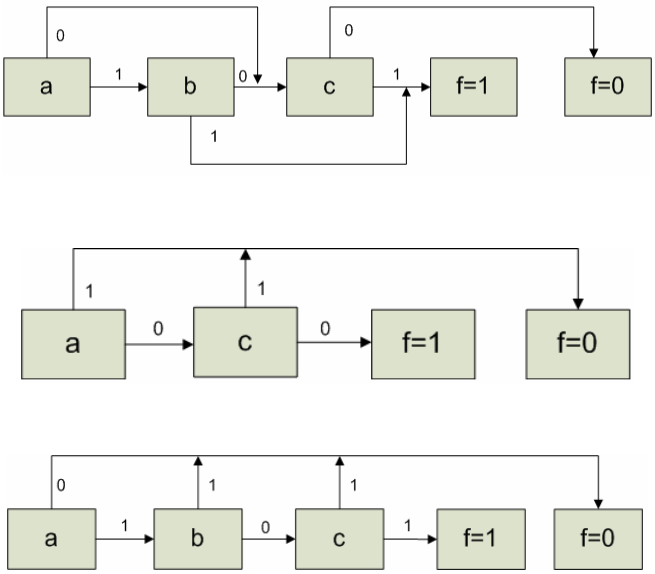

Рис. 51. Преобразованные линейные бинарные графы функций системы

После этого заменим в реализации каждой из перечисленных формул значение «единица» на соответствующее значение функции выхода. Теперь будем последовательно заменять терминальные узлы, помеченные нулем, бинарными линейными графами, соответствующими реализациям следующих булевых функций системы. Построенный бинарный граф будет также линейным.

В полученном таким образом линейном бинарном графе будет терминальный узел, помеченный нулем, однако этот узел оказывается недостижимым ни при каких значениях переменных, так как рассматриваемая система булевых функций, является полной. Поэтому весь линейный бинарный граф, выражающий последнюю булеву функцию, можно заменить терминальным узлом, помеченным соответствующим значением функции. В итоге получим линейный бинарный граф, изображенный на рис. 52.

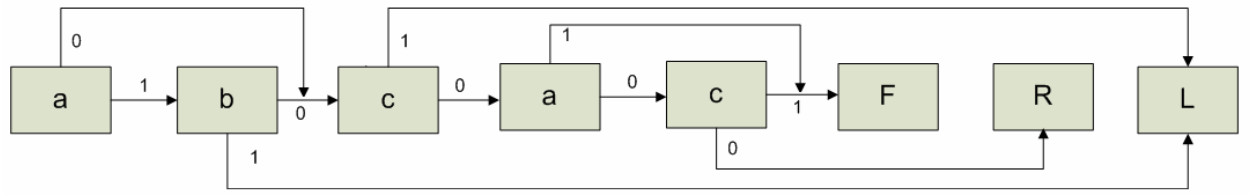

Рис. 52. Представление дискретной функции линейным бинарным графом

Произвольная дискретная функция булевых переменных может быть выражена линейным бинарным графом аналогичным образом.

4.1.3. Представление функции переходов управляющего автомата линейными бинарными графами

Опишем предлагаемый метод представления функции переходов автомата линейными бинарными графами. Зададим для каждого состояния $q \in Q$ функцию $\sigma_q : X \to Q \times Y$, такую что $\sigma_q(x) = (\delta(q, x), \lambda(q, x))$ для $\forall x \in X$. Здесь Q – множество состояний управляющего автомата, X – множество входных воздействий, Y – множество выходных воздействий, $\delta : Q \times X \to Q$ – функция переходов, $\lambda : Q \times X \to Y$ – функция выхода. Будем называть функции σ_q функциями переходов из состояний.

Каждая из функций переходов из состояний может быть определена собственным линейным бинарным графом, так как является дискретной функцией булевых переменных. Таким образом, функция переходов автомата в целом может быть представлена упорядоченным набором линейных бинарных графов.

На рис. 53 приведен пример автомата Мили. При этом *a*, *b*, *c* — входные переменные булевого типа, *L*, *R*, *F* — выходные воздействия. Этот же автомат, представленный линейными бинарными графами, приведен на рис. 54.

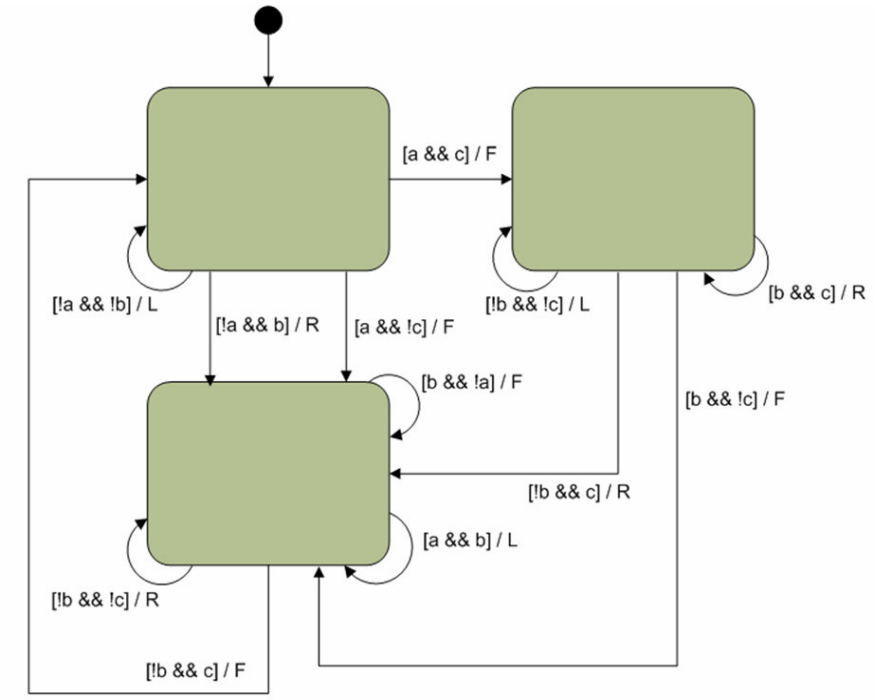

Рис. 53. Пример автомата Мили

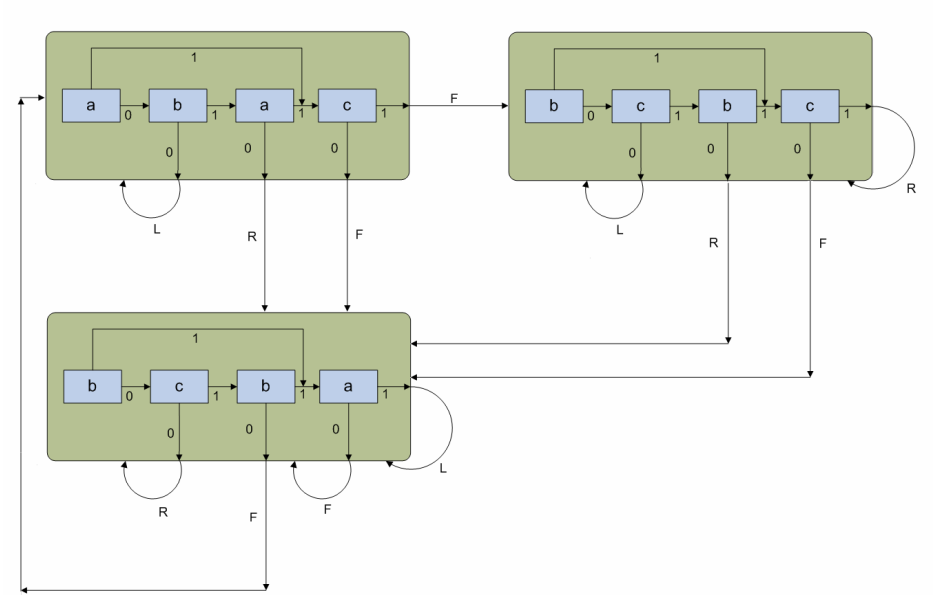

Рис. 54. Представление автомата Мили линейными бинарными графами

4.2. РАЗРАБОТКА АЛГОРИТМОВ РЕАЛИЗАЦИИ ОПЕРАЦИЙ МУТАЦИИ И СКРЕЩИВАНИЯ

В настоящем разделе описываются операции мутации и скрещивания функций переходов управляющего автомата, представленных линейными бинарными графами. Для этого определяется представление линейного бинарного графа в виде хромосомы и генетические операции над введенным представлением.

4.2.1. Представление линейного бинарного графа в виде хромосомы

Определим представление линейных бинарных графов в виде хромосом:

- Граф представляет собой упорядоченный список узлов. При этом считается, что узлы перечисляются в таком порядке, что одно из выходящих ребер каждого узла ведет в следующий узел.
- В каждом нетерминальном узле хранится переменная, которой помечен данный узел (переменная расщепления), значение этой переменной, соответствующее переходу в следующий узел, и номер узла, в который производится переход при инверсном значении переменной расщепления.
- В каждом из терминальных узлов хранится значение соответствующей функции переходов.

Описанное представление хромосомы может быть записано в виде строки. При этом терминальные узлы являются последними узлами линейного бинарного графа. Поэтому возможно отдельно выделить части, кодирующие терминальные и нетерминальные узлы.

Продемонстрируем это на примере. Закодируем в виде строки граф, приведенный на рис. 55.

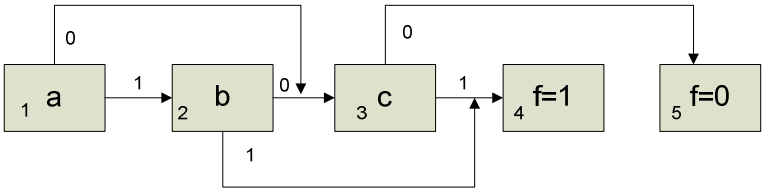

Рис. 55. Пример линейного бинарного графа

Заметим, что узлы этого графа пронумерованы в требуемом порядке слева направо. Тогда этому графу будет соответствовать следующая строка:

```
a13b04c15 10,
```

в которой указанные выше части бинарного графа разделены пробелом.

Поясним первые три символа этой строки. Из первого узла, помеченного переменной *a*, переход в соседнюю вершину выполняется по значению «единица», а переход по значению «ноль» выполняется в третий узел.

4.2.2. Генетические операции

В качестве генетических операций над управляющими автоматами, представленными линейными бинарными графами будем использовать:

- случайное порождение автомата – в каждом состоянии создается случайный линейный бинарный граф;
- скрещивание автоматов – линейные бинарные графы в соответствующих состояниях скрещиваются. Номер стартового состояния копируется из случайного родителя;
- мутация автомата – в линейном бинарном графе, который соответствует случайному состоянию, производится мутация.

При этом считается, что число состояний в автомате фиксировано. Поэтому противоречий при выполнении определенных таким образом операций не возникает.

Определим теперь генетические операции над линейными бинарными графами.

Для скрещивания линейных бинарных графов можно применить любой из известных методов скрещивания строк, отдельно скрещивая части, описывающие нетерминальные и терминальные узлы. Например, это может быть одноточечный или двухточечный кроссовер [66]. При этом часть, соответствующая терминальным узлам, должна быть скопирована полностью из одной родительской

хромосомы. В ином случае терминальная часть при скрещивании упрощается и может в итоге быть вырождена.

Операция скрещивания для строк разной длины должна быть модифицирована, так как в ином случае возможна ситуация, когда получившийся линейный бинарный граф будет некорректен: ребра, скопированные из одного из родителей, могут вести в несуществующие узлы. Модификация заключается во введении дополнительного требования: число узлов, скопированных из второй хромосомы, должно быть достаточно для корректности порожденного линейного бинарного графа.

Операция мутации может быть определена следующим образом: выбирается произвольный нетерминальный узел, после этого в него вносится одно из следующих изменений:

- замена переменной, которой помечен модифицируемый узел;
- замена значения переменной, которым помечен переход в следующий в порядке нумерации узел;
- замена номера узла, в который производится переход при несовпадении значения переменной.

4.3. Программная реализация

В настоящей главе рассматривается пример программной реализации метода генетического программирования, основанного на предложенном представлении функции переходов, на языке программирования *Java*.

4.3.1. Представление автоматов в виде программных объектов

При использовании изложенного метода представления функции переходов автомат управления может быть представлен объектом следующего вида:

```
public class Automata<Action> {
    public static class Transition<Action> {
         int destination;
         Action action;
    }

    List<LinearBinaryGraph<Transition<Action>>> functions;
    int startState;
}
```

Для простоты здесь и далее в листингах приводятся только те члены класса, которые относятся непосредственно к внутреннему представлению особи в генетическом программировании.

Здесь `startState` – стартовое состояние автомата, `functions` – список линейных бинарных графов, выражающих функции переходов из каждого состояния автомата. Класс `Action` определяет выходные действия автомата и является параметром *generic*-класса `Automata`. Переходу автомата соответствует вложенный класс `Transition`. Каждый переход характеризуется номером состояния, в который ведет переход, и ассоциированным действием.

Теперь определим представление линейного бинарного графа в виде объекта языка *Java*:

```
public class LinearBinaryGraph<Result> {
    public static class Node<Result> {
        int predicate;
        boolean forward;
        int transition;
```

```
        Result result;
    }

    List<Node<Result>> nodes;
}
```

Здесь линейный бинарный граф определяется списком узлов. Класс `Result` определяет класс значений функции, представляемой линейным бинарным графом. Каждый узел линейного бинарного графа характеризуется следующими параметрами:

- `predicate` – номер предиката, по которому осуществляется расщепление в данном узле;
- `forward` – булево значение переменной, которым помечен переход в следующий в порядке нумерации узел;
- `transition` – номера узла, в который производится переход при несовпадении значения переменной;
- `result` – значение функции, которым помечен данный узел. В нетерминальных узлах это значение равно `null`.

4.3.2. Программная реализация генетических операций

Реализуем генетические операции непосредственно в самих классах, соответствующих особям. Сначала определим интерфейсы особей генетического программирования в общем виде следующим образом:

```
public interface Evolvable<Cromosome> {
    Chromosome cross(Cromosome other);
    Chromosome mutate();
}
```

Интерфейс `Evolvable` является параметризуемым по классу-хромосоме. Метод `cross` соответствует генетической операции скрещивания. Параметром метода является хромосома, с которой производится скрещивание. Метод `mutate` соответствует генетической операции мутации, его результатом является мутировавшая хромосома.

Интерфейс `Evolvable` спроектирован таким образом, что хромосомы не могут быть модифицированы в процессе генетических операций. Поэтому одна и та же хромосома линейного бинарного графа может быть использована в нескольких хромосомах автомата. Кроме того, это упрощает перенос на многопоточную архитектуру, так как генетические операции могут безопасно выполняться одновременно из нескольких потоков.

Теперь реализуем этот интерфейс в классах-хромосомах `Automata` и `LinearBinaryGraph`. Сигнатуры классов изменятся следующим образом:

```
public class Automata<Action> implements Evolvable<Automata<Action>>

public class LinearBinaryGraph<Result> implements
                  Evolvable<LinearBinaryGraph<Result>>
```

После этого определим в классе `Automata<Action>` программную реализацию методов интерфейса `Evolvable<Automata<Action>>`:

```
public Automata<Action> cross(Automata<Action> other) {
    assert functions.size() == other.functions.size();

    final int stateCount = functions.size();
    List<LinearBinaryGraph<Transition<Action>>> newFunctions =
        new ArrayList<LinearBinaryGraph<Transition<Action>>();

    for (int state = 0; state < stateCount; state++) {
        LinearBinaryGraph<Transition<Action>> first =
                functions.get(state);
        LinearBinaryGraph<Transition<Action>> second =
                other.functions.get(state);
        newFunctions.add(first.cross(second));
    }

    int newStartState = Math.random() > 0.5 ? startState :
                other.startState;

    return new Automata<Action>(newStartState, newFunctions);
}

public Automata<Action> mutate() {
   final int stateCount = functions.size();
   int randomState = (int) (Math.random() * stateCount);
   LinearBinaryGraph<Transition<Action>> newFunction =
                functions.get(randomState).mutate();

   List<LinearBinaryGraph<Transition<Action>>> newFunctions = new
                ArrayList<LinearBinaryGraph<Transition<Action>>>(functi
                ons);
   newFunctions.set(randomState, newFunction);

   return new Automata<Action>(startState, newFunctions);
}
```

Здесь прямо реализуются разработанные алгоритмы скрещивания и мутации автоматов, представленных линейными бинарными графами.

Приведем реализацию генетических операций в классе `LinearBinaryGraph<Result>`. Будем использовать в качестве операции скрещивания линейных бинарных графов одноточечный кроссовер над строками фиксированной длины. Также предполагается, что число нетерминальных узлов в линейных бинарных графах внутри одной популяции фиксировано.

```
public LinearBinaryGraph<Result> cross(LinearBinaryGraph<Result> other) {
    assert nodes.size() == other.nodes.size();

    int nonterms = getNonTermsCount();
    int split = (int) (Math.random() * nonterms);
    List<Node<Result>> newNodes = new ArrayList<Node<Result>>();
```

```
    // Одноточечный кроссовер нетерминальных частей
    for (int index = 0; index < nonterms; index++) {
        newNodes.add(index < fs ? nodes.get(index) :
                  other.nodes.get(index));
    }

    // Копирование терминальных узлов из случайного родителя
    List<Node<Result>> terms = Math.random() < 0.5 ?
                  nodes.sublist(nonterms) :
                  other.nodes.sublist(nonterms);
    newNodes.addAll(terms);

    return new LinearBinaryGraph<Result>(newNodes);
}

public LinearBinaryGraph<Result> mutate() {
    List<Node<Result>> newNodes = new ArrayList<Node<Result>>(nodes);

    int index = (int) (Math.random() * getNonTermsCount());
    Node<Result> newNode = new Node<Result>(nodes.get(index));
    if (Math.random() < TRANSITION_MUTATE_RATE) {
        int newTransition = index + 1 + (int) (Math.random() *
                  (nodes.size() - index - 1));
        newNode.transition = newTransition;
    } else if (Math.random() < TRANSITION_MUTATE_RATE +
                  PREDICATE_MUTATE_RATE) {
        int newPredicate = (int) (Math.random() * getPredicatesCount());
        newNode.predicate = newPredicate;
    } else {
        newNode.forward = !newNode.forward;
    }

    newNodes.set(index, newNode);
    return new LinearBinaryGraph<Result>(newNodes);
}
```

Приведенная реализация зависит от следующих членов класса:

- `getNonTermsCount()` – возвращает количество нетерминальных узлов;
- `TRANSITION_MUTATE_RATE` – вероятность изменения перехода при мутации нетерминального узла;
- `PREDICATE_MUTATE_RATE` – вероятность изменения предиката при мутации нетерминального узла;
- `getRandomResult()` – возвращает случайное значение представляемой функции.

Операция скрещивания в данной реализации выполняет одноточечный кроссовер терминальных узлов и копирование нетерминальных узлов из случайного родителя.

Операция мутации выбирает случайный нетерминальный узел. После этого с вероятностью `TRANSITION_MUTATE_RATE` в нем изменяется переход, с вероятностью `PREDICATE_MUTATE_RATE` – переменная расщепления, с оставшейся вероятностью – булево значение, которым помечен переход в следующий узел.

Выводы по главе 4

1. Линейные бинарные графы являются важным частным случаем бинарных разрешающих диаграмм. Важным свойством линейных бинарных графов является то, что число элементов, необходимых для реализации булевой формулы бинарным линейным графом, линейно зависит от числа букв в буевой формуле.
2. Предложен метод представления автоматов на основе представления функций переходов линейными бинарными графами.
3. Приведена программная реализация этого метода на языке программирования *Java*.

5. ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ МЕТОДА ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ, ОСНОВАННОГО НА ИСПОЛЬЗОВАНИИ ЛИНЕЙНЫХ БИНАРНЫХ ГРАФОВ ДЛЯ ПРЕДСТАВЛЕНИЯ ФУНКЦИИ ПЕРЕХОДОВ АВТОМАТОВ

В настоящем разделе приведены результаты экспериментальных исследований метода генетического программирования, основанного на использовании линейных бинарных графов для представления функции переходов автоматов.

5.1. Задача «Умный муравей-3»

Разработанный подход был протестирован на задаче «Умный муравей-3» [63]. Приведем описание задачи. Муравей находится на случайном игровом поле. Поле представляет собой тор размером 32×32 клетки. При этом муравей видит перед собой некоторую область (рис. 56).

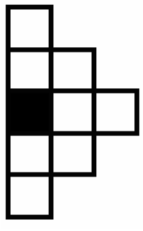

Рис. 56. Видимая муравью область

Еда (яблоки) в каждой клетке располагается с некоторой вероятностью μ. Значение μ является параметром задачи. Игра длится 200 ходов, за каждый из которых муравей может сделать одно из трех действий: поворот налево, поворот направо, шаг вперед. Если после хода муравей попадает на клетку, где есть яблоко, то оно съедается. Целью задачи является построение стратегии поведения муравья, при которой математическое ожидание числа съеденных яблок максимально.

Автомат управления муравьем в этой задаче имеет восемь (число видимых клеток) булевых входных переменных. Каждая из них определяет, есть ли яблоко в клетке, соответствующей переменной.

Для определения эффективности предложенного подхода было выполнено его сравнение с методом представления функций переходов полными таблицами.

Эксперимент заключался в сравнении полученных значений функции (объема съеденной еды) за фиксированное число шагов. Запуск производился со следующими настройками: стратегия отбора – элитизм, для размножения отбираются 25 % популяции, имеющих наибольшее значение фитнесс-функции; частота мутации – 2 %; размер популяции – 200 особей; число популяций – 100; фитнесс-функция – среднее значение съеденной еды на 200 случайных полях. Поля внутри одной популяции совпадают, поля для различных популяций – различны. Последнее измерение фитнесс-функции осуществлялось на случайном наборе из 2000 полей.

Результаты эксперимента приведены в табл. 10.

Таблица 10. Результаты эксперимента

μ	0.04			
Количество состояний	2	4	8	16
Полные таблицы переходов	17.18	15.94	15.03	13.68
Предложенный метод	18.16	20.07	17.12	15.53

Анализ результатов показывает, что в проведенном эксперименте предложенный метод является более эффективным по сравнению с представлением функции переходов полными таблицами.

5.2. Результаты экспериментов

На рисунках ниже приводятся графики, отображающие ход эволюции при выводе автоматов с различным числом состояний.

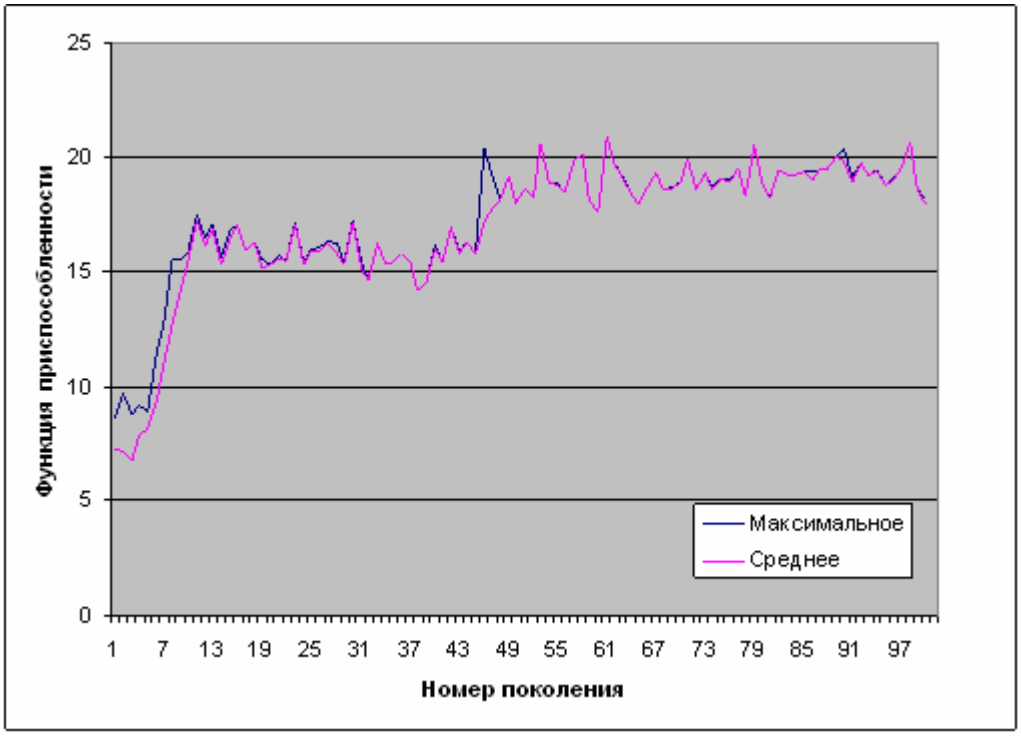

Рис. 57. Ход эволюции для автоматов с двумя состояниями

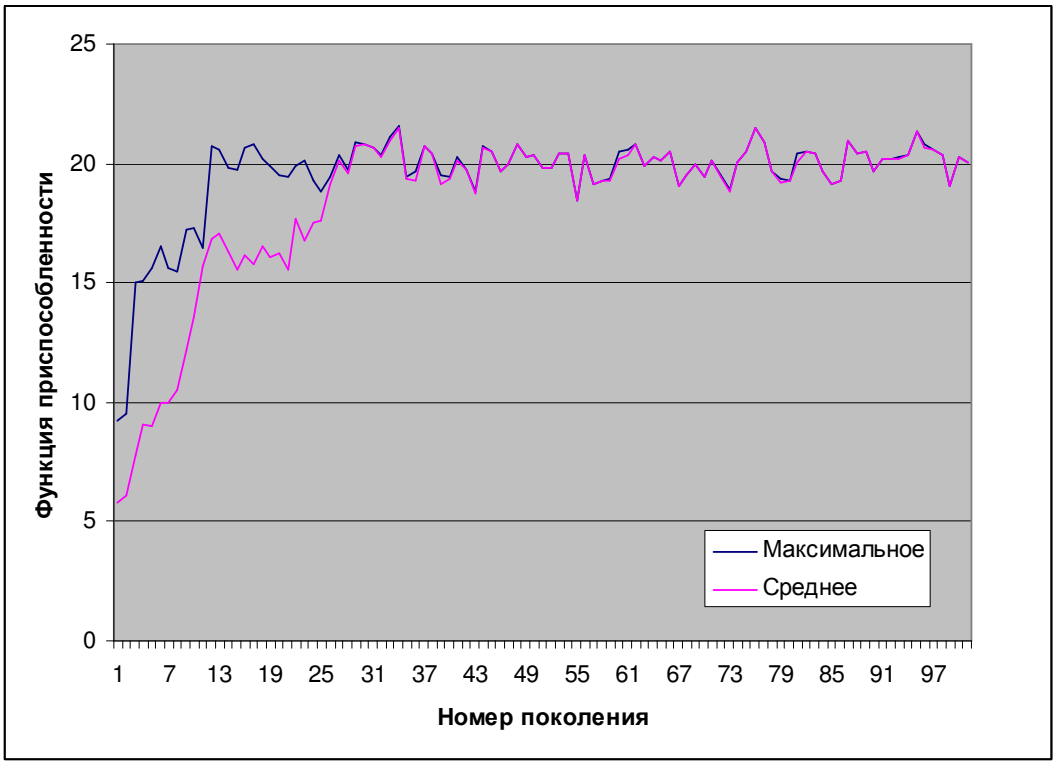

Рис. 58. Ход эволюции для автоматов с четырьмя состояниями

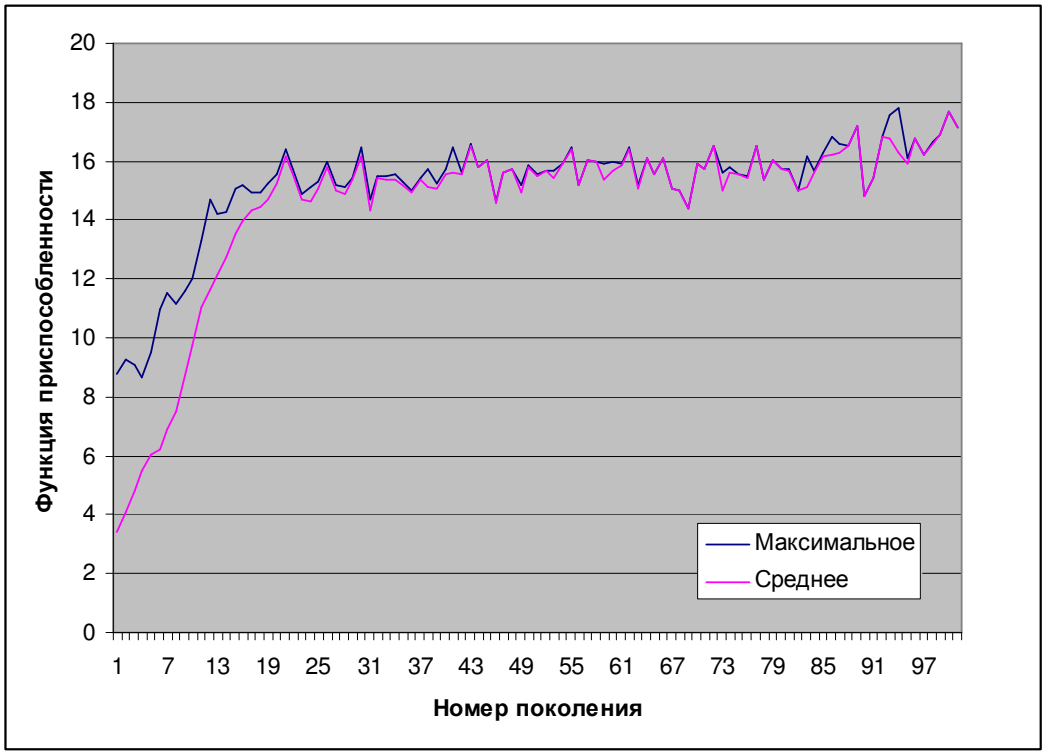

Рис. 59. Ход эволюции для автоматов с восемью состояниями

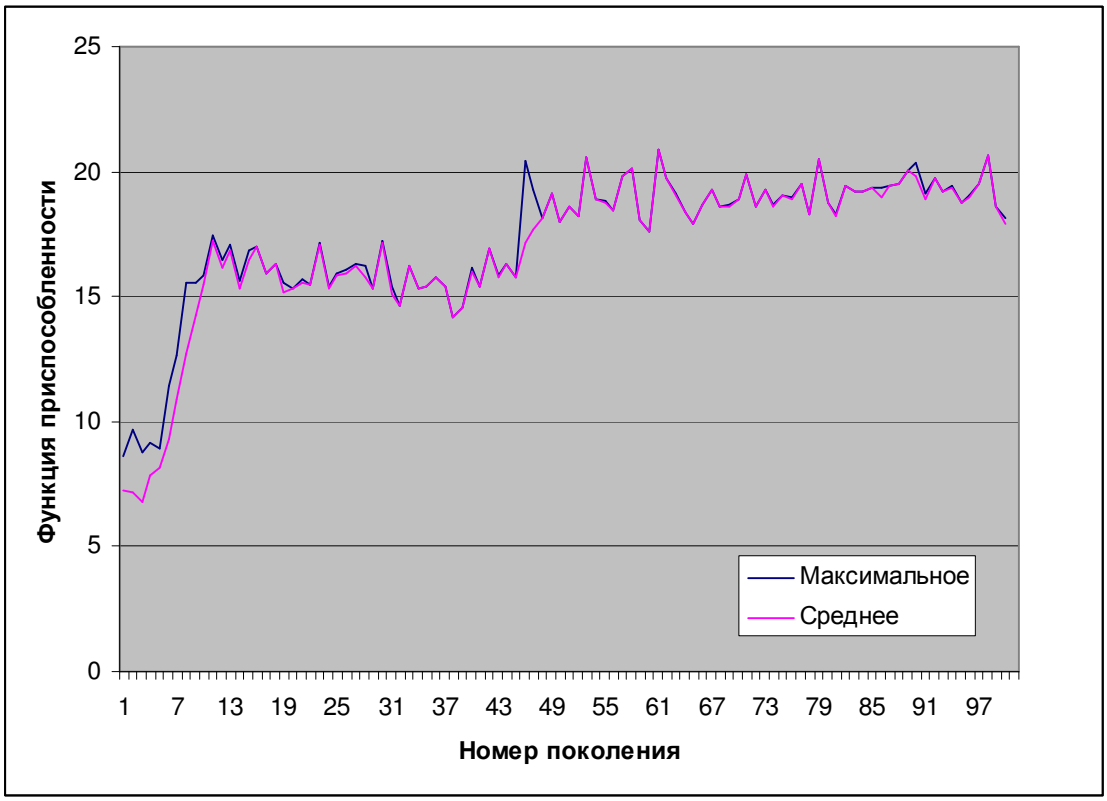

Рис. 60. Ход эволюции для автоматов с шестнадцатью состояниями

5.3. Протоколы экспериментов

Ниже приводится протокол работы программы в ходе вычислительных экспериментов.

5.3.1. Построение автоматов с двумя состояниями

```
INFO: [evolved] 1, max: 8,59, avg: 7,21, div: 0,424 / 0,447 in 17 sec
Oct 17, 2009 9:01:52 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 2, max: 9,7, avg: 7,15, div: 0,431 / 0,439 in 17 sec
Oct 17, 2009 9:02:10 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 3, max: 8,77, avg: 6,78, div: 0,415 / 0,429 in 17 sec
Oct 17, 2009 9:02:27 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 4, max: 9,11, avg: 7,83, div: 0,397 / 0,411 in 17 sec
Oct 17, 2009 9:02:45 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 5, max: 8,91, avg: 8,15, div: 0,364 / 0,386 in 17 sec
Oct 17, 2009 9:03:02 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 6, max: 11,43, avg: 9,33, div: 0,354 / 0,332 in 17 sec
Oct 17, 2009 9:03:20 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 7, max: 12,66, avg: 10,92, div: 0,357 / 0,338 in 17 sec
Oct 17, 2009 9:03:38 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 8, max: 15,54, avg: 12,72, div: 0,228 / 0,310 in 17 sec
Oct 17, 2009 9:03:55 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 9, max: 15,52, avg: 14,29, div: 0,160 / 0,294 in 17 sec
```

```
Oct 17, 2009 9:04:13 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 10, max: 15,84, avg: 15,47, div: 0,153 / 0,272 in 17 sec
Oct 17, 2009 9:04:30 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 11, max: 17,43, avg: 17,23, div: 0,170 / 0,276 in 17 sec
Oct 17, 2009 9:04:48 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 12, max: 16,5, avg: 16,19, div: 0,145 / 0,202 in 17 sec
Oct 17, 2009 9:05:05 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 13, max: 17,04, avg: 16,81, div: 0,108 / 0,147 in 17 sec
Oct 17, 2009 9:05:29 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 14, max: 15,62, avg: 15,29, div: 0,084 / 0,279 in 24 sec
Oct 17, 2009 9:05:48 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 15, max: 16,83, avg: 16,43, div: 0,043 / 0,129 in 18 sec
Oct 17, 2009 9:06:06 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 16, max: 17,0, avg: 16,96, div: 0,061 / 0,227 in 17 sec
Oct 17, 2009 9:06:25 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 17, max: 15,94, avg: 15,94, div: 0,084 / 0,067 in 18 sec
Oct 17, 2009 9:06:42 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 18, max: 16,31, avg: 16,31, div: 0,084 / 0,235 in 17 sec
Oct 17, 2009 9:07:00 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 19, max: 15,53, avg: 15,19, div: 0,069 / 0,079 in 17 sec
Oct 17, 2009 9:07:19 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 20, max: 15,34, avg: 15,32, div: 0,079 / 0,241 in 19 sec
Oct 17, 2009 9:07:38 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 21, max: 15,68, avg: 15,53, div: 0,063 / 0,070 in 19 sec
Oct 17, 2009 9:07:57 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 22, max: 15,5, avg: 15,48, div: 0,064 / 0,232 in 18 sec
Oct 17, 2009 9:08:15 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 23, max: 17,14, avg: 17,11, div: 0,078 / 0,064 in 18 sec
Oct 17, 2009 9:08:32 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 24, max: 15,36, avg: 15,32, div: 0,075 / 0,233 in 17 sec
Oct 17, 2009 9:08:50 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 25, max: 15,94, avg: 15,88, div: 0,074 / 0,066 in 17 sec
Oct 17, 2009 9:09:08 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 26, max: 16,09, avg: 15,92, div: 0,062 / 0,233 in 17 sec
Oct 17, 2009 9:09:25 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 27, max: 16,28, avg: 16,26, div: 0,072 / 0,055 in 17 sec
Oct 17, 2009 9:09:42 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 28, max: 16,2, avg: 15,75, div: 0,050 / 0,236 in 17 sec
Oct 17, 2009 9:10:00 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 29, max: 15,35, avg: 15,31, div: 0,075 / 0,064 in 17 sec
Oct 17, 2009 9:10:17 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 30, max: 17,22, avg: 17,18, div: 0,070 / 0,237 in 17 sec
Oct 17, 2009 9:10:35 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 31, max: 15,36, avg: 15,11, div: 0,051 / 0,073 in 17 sec
Oct 17, 2009 9:10:52 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 32, max: 14,65, avg: 14,65, div: 0,066 / 0,232 in 17 sec
Oct 17, 2009 9:11:10 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 33, max: 16,25, avg: 16,25, div: 0,080 / 0,064 in 17 sec
```

```
Oct 17, 2009 9:11:27 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 34, max: 15,29, avg: 15,29, div: 0,077 / 0,236 in 17 sec
Oct 17, 2009 9:11:44 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 35, max: 15,43, avg: 15,43, div: 0,029 / 0,065 in 17 sec
Oct 17, 2009 9:12:01 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 36, max: 15,79, avg: 15,79, div: 0,079 / 0,232 in 17 sec
Oct 17, 2009 9:12:18 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 37, max: 15,39, avg: 15,39, div: 0,073 / 0,071 in 17 sec
Oct 17, 2009 9:12:36 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 38, max: 14,17, avg: 14,17, div: 0,079 / 0,239 in 17 sec
Oct 17, 2009 9:12:53 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 39, max: 14,58, avg: 14,56, div: 0,068 / 0,072 in 17 sec
Oct 17, 2009 9:13:15 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 40, max: 16,15, avg: 16,03, div: 0,078 / 0,233 in 21 sec
Oct 17, 2009 9:13:32 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 41, max: 15,42, avg: 15,42, div: 0,070 / 0,066 in 17 sec
Oct 17, 2009 9:13:50 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 42, max: 16,89, avg: 16,89, div: 0,075 / 0,237 in 17 sec
Oct 17, 2009 9:14:07 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 43, max: 15,83, avg: 15,80, div: 0,060 / 0,067 in 17 sec
Oct 17, 2009 9:14:24 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 44, max: 16,3, avg: 16,30, div: 0,062 / 0,234 in 17 sec
Oct 17, 2009 9:14:41 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 45, max: 15,75, avg: 15,75, div: 0,037 / 0,069 in 17 sec
Oct 17, 2009 9:14:58 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 46, max: 20,42, avg: 17,18, div: 0,047 / 0,231 in 17 sec
Oct 17, 2009 9:15:16 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 47, max: 19,28, avg: 17,65, div: 0,049 / 0,059 in 17 sec
Oct 17, 2009 9:15:33 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 48, max: 18,11, avg: 18,11, div: 0,041 / 0,222 in 17 sec
Oct 17, 2009 9:15:50 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 49, max: 19,14, avg: 19,14, div: 0,027 / 0,040 in 17 sec
Oct 17, 2009 9:16:07 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 50, max: 18,02, avg: 18,02, div: 0,042 / 0,217 in 17 sec
Oct 17, 2009 9:16:25 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 51, max: 18,61, avg: 18,61, div: 0,041 / 0,041 in 17 sec
Oct 17, 2009 9:16:42 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 52, max: 18,19, avg: 18,19, div: 0,047 / 0,220 in 17 sec
Oct 17, 2009 9:16:59 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 53, max: 20,6, avg: 20,60, div: 0,012 / 0,040 in 17 sec
Oct 17, 2009 9:17:16 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 54, max: 18,9, avg: 18,90, div: 0,036 / 0,218 in 17 sec
Oct 17, 2009 9:17:34 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 55, max: 18,8, avg: 18,77, div: 0,022 / 0,037 in 17 sec
Oct 17, 2009 9:17:51 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 56, max: 18,41, avg: 18,41, div: 0,043 / 0,215 in 17 sec
Oct 17, 2009 9:18:08 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 57, max: 19,8, avg: 19,80, div: 0,024 / 0,039 in 17 sec
```

```
Oct 17, 2009 9:18:25 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 58, max: 20,16, avg: 20,16, div: 0,039 / 0,219 in 17 sec
Oct 17, 2009 9:18:42 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 59, max: 18,07, avg: 18,07, div: 0,050 / 0,040 in 17 sec
Oct 17, 2009 9:19:00 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 60, max: 17,57, avg: 17,57, div: 0,046 / 0,220 in 17 sec
Oct 17, 2009 9:19:17 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 61, max: 20,91, avg: 20,91, div: 0,046 / 0,040 in 17 sec
Oct 17, 2009 9:19:34 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 62, max: 19,71, avg: 19,71, div: 0,032 / 0,219 in 17 sec
Oct 17, 2009 9:19:51 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 63, max: 19,12, avg: 19,06, div: 0,033 / 0,041 in 17 sec
Oct 17, 2009 9:20:09 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 64, max: 18,36, avg: 18,36, div: 0,035 / 0,216 in 17 sec
Oct 17, 2009 9:20:26 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 65, max: 17,93, avg: 17,93, div: 0,042 / 0,035 in 17 sec
Oct 17, 2009 9:20:43 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 66, max: 18,64, avg: 18,64, div: 0,044 / 0,218 in 17 sec
Oct 17, 2009 9:21:00 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 67, max: 19,29, avg: 19,29, div: 0,043 / 0,038 in 17 sec
Oct 17, 2009 9:21:18 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 68, max: 18,56, avg: 18,56, div: 0,038 / 0,219 in 17 sec
Oct 17, 2009 9:21:35 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 69, max: 18,64, avg: 18,57, div: 0,048 / 0,041 in 17 sec
Oct 17, 2009 9:21:52 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 70, max: 18,92, avg: 18,92, div: 0,042 / 0,220 in 17 sec
Oct 17, 2009 9:22:09 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 71, max: 19,87, avg: 19,87, div: 0,039 / 0,039 in 17 sec
Oct 17, 2009 9:22:27 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 72, max: 18,62, avg: 18,62, div: 0,036 / 0,218 in 17 sec
Oct 17, 2009 9:22:44 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 73, max: 19,31, avg: 19,31, div: 0,047 / 0,040 in 17 sec
Oct 17, 2009 9:23:01 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 74, max: 18,68, avg: 18,62, div: 0,038 / 0,218 in 17 sec
Oct 17, 2009 9:23:19 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 75, max: 19,09, avg: 19,09, div: 0,041 / 0,039 in 17 sec
Oct 17, 2009 9:23:36 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 76, max: 18,95, avg: 18,91, div: 0,044 / 0,218 in 17 sec
Oct 17, 2009 9:23:53 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 77, max: 19,54, avg: 19,54, div: 0,043 / 0,037 in 17 sec
Oct 17, 2009 9:24:10 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 78, max: 18,26, avg: 18,26, div: 0,041 / 0,220 in 17 sec
Oct 17, 2009 9:24:27 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 79, max: 20,47, avg: 20,47, div: 0,043 / 0,040 in 17 sec
Oct 17, 2009 9:24:45 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 80, max: 18,75, avg: 18,75, div: 0,035 / 0,218 in 17 sec
Oct 17, 2009 9:25:02 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 81, max: 18,26, avg: 18,21, div: 0,040 / 0,034 in 17 sec
```

```
Oct 17, 2009 9:25:19 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 82, max: 19,47, avg: 19,17, div: 0,044 / 0,220 in 17 sec
Oct 17, 2009 9:25:37 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 83, max: 19,17, avg: 19,17, div: 0,038 / 0,037 in 17 sec
Oct 17, 2009 9:26:13 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 84, max: 19,19, avg: 19,17, div: 0,033 / 0,217 in 36 sec
Oct 17, 2009 9:26:33 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 85, max: 19,35, avg: 19,33, div: 0,039 / 0,037 in 19 sec
Oct 17, 2009 9:26:52 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 86, max: 19,34, avg: 18,95, div: 0,031 / 0,219 in 18 sec
Oct 17, 2009 9:27:30 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 87, max: 19,4, avg: 19,40, div: 0,039 / 0,038 in 38 sec
Oct 17, 2009 9:28:06 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 88, max: 19,52, avg: 19,52, div: 0,028 / 0,217 in 36 sec
Oct 17, 2009 9:28:41 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 89, max: 20,02, avg: 20,02, div: 0,041 / 0,039 in 34 sec
Oct 17, 2009 9:29:09 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 90, max: 20,36, avg: 19,80, div: 0,029 / 0,215 in 28 sec
Oct 17, 2009 9:29:35 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 91, max: 19,13, avg: 18,92, div: 0,035 / 0,037 in 25 sec
Oct 17, 2009 9:29:52 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 92, max: 19,76, avg: 19,76, div: 0,039 / 0,219 in 17 sec
Oct 17, 2009 9:30:09 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 93, max: 19,2, avg: 19,20, div: 0,026 / 0,039 in 17 sec
Oct 17, 2009 9:30:27 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 94, max: 19,43, avg: 19,38, div: 0,049 / 0,216 in 17 sec
Oct 17, 2009 9:30:44 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 95, max: 18,77, avg: 18,77, div: 0,025 / 0,037 in 17 sec
Oct 17, 2009 9:31:01 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 96, max: 19,09, avg: 19,00, div: 0,036 / 0,220 in 17 sec
Oct 17, 2009 9:31:18 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 97, max: 19,49, avg: 19,49, div: 0,037 / 0,032 in 17 sec
Oct 17, 2009 9:31:36 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 98, max: 20,62, avg: 20,62, div: 0,022 / 0,212 in 17 sec
Oct 17, 2009 9:31:53 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 99, max: 18,6, avg: 18,60, div: 0,044 / 0,031 in 17 sec
Oct 17, 2009 9:32:10 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 100, max: 18,16, avg: 17,90, div: 0,022 / 0,211 in 17 sec
Oct 17, 2009 9:37:55 PM ru.ifmo.ctddev.autoant.Ant2Problem main
INFO: result: 18.16[ 2 states, 0,04] in 344 sec
```

5.3.2. Построение автоматов с четырьмя состояниями

```
Oct 17, 2009 9:38:12 PM ru.ifmo.ctddev.autoant.fx.Genome <init>
INFO: [created] max: 7,92, avg: 4,28, div: 0,452 / 0,454 in 17 sec
Oct 17, 2009 9:38:30 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 1, max: 9,26, avg: 5,83, div: 0,427 / 0,446 in 17 sec
Oct 17, 2009 9:38:47 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 2, max: 9,53, avg: 6,12, div: 0,390 / 0,435 in 17 sec
```

```
Oct 17, 2009 9:39:04 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 3, max: 15,0, avg: 7,74, div: 0,426 / 0,420 in 17 sec
Oct 17, 2009 9:39:21 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 4, max: 15,09, avg: 9,04, div: 0,397 / 0,410 in 17 sec
Oct 17, 2009 9:39:38 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 5, max: 15,66, avg: 8,98, div: 0,359 / 0,382 in 17 sec
Oct 17, 2009 9:39:56 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 6, max: 16,52, avg: 9,99, div: 0,361 / 0,323 in 17 sec
Oct 17, 2009 9:40:13 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 7, max: 15,64, avg: 9,96, div: 0,342 / 0,305 in 17 sec
Oct 17, 2009 9:40:30 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 8, max: 15,44, avg: 10,55, div: 0,312 / 0,321 in 17 sec
Oct 17, 2009 9:40:47 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 9, max: 17,25, avg: 12,20, div: 0,298 / 0,312 in 17 sec
Oct 17, 2009 9:41:05 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 10, max: 17,33, avg: 13,56, div: 0,240 / 0,303 in 17 sec
Oct 17, 2009 9:41:22 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 11, max: 16,48, avg: 15,70, div: 0,190 / 0,291 in 17 sec
Oct 17, 2009 9:41:39 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 12, max: 20,73, avg: 16,84, div: 0,230 / 0,269 in 17 sec
Oct 17, 2009 9:41:56 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 13, max: 20,58, avg: 17,06, div: 0,233 / 0,246 in 17 sec
Oct 17, 2009 9:42:14 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 14, max: 19,8, avg: 16,30, div: 0,210 / 0,218 in 17 sec
Oct 17, 2009 9:42:31 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 15, max: 19,71, avg: 15,56, div: 0,215 / 0,174 in 17 sec
Oct 17, 2009 9:42:48 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 16, max: 20,66, avg: 16,16, div: 0,170 / 0,301 in 17 sec
Oct 17, 2009 9:43:05 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 17, max: 20,77, avg: 15,80, div: 0,234 / 0,174 in 17 sec
Oct 17, 2009 9:43:23 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 18, max: 20,22, avg: 16,54, div: 0,200 / 0,302 in 17 sec
Oct 17, 2009 9:43:40 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 19, max: 19,88, avg: 16,06, div: 0,190 / 0,169 in 17 sec
Oct 17, 2009 9:43:57 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 20, max: 19,54, avg: 16,27, div: 0,202 / 0,293 in 17 sec
Oct 17, 2009 9:44:15 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 21, max: 19,43, avg: 15,53, div: 0,215 / 0,139 in 17 sec
Oct 17, 2009 9:44:32 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 22, max: 19,89, avg: 17,66, div: 0,200 / 0,280 in 17 sec
Oct 17, 2009 9:44:49 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 23, max: 20,1, avg: 16,80, div: 0,155 / 0,127 in 17 sec
Oct 17, 2009 9:45:06 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 24, max: 19,29, avg: 17,54, div: 0,127 / 0,284 in 17 sec
Oct 17, 2009 9:45:23 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 25, max: 18,84, avg: 17,63, div: 0,116 / 0,148 in 17 sec
Oct 17, 2009 9:45:41 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 26, max: 19,43, avg: 19,12, div: 0,127 / 0,288 in 17 sec
```

```
Oct 17, 2009 9:45:58 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 27, max: 20,32, avg: 20,09, div: 0,129 / 0,176 in 17 sec
Oct 17, 2009 9:46:15 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 28, max: 19,77, avg: 19,61, div: 0,114 / 0,264 in 17 sec
Oct 17, 2009 9:46:33 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 29, max: 20,85, avg: 20,73, div: 0,105 / 0,126 in 17 sec
Oct 17, 2009 9:46:50 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 30, max: 20,84, avg: 20,84, div: 0,089 / 0,261 in 17 sec
Oct 17, 2009 9:47:07 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 31, max: 20,67, avg: 20,62, div: 0,101 / 0,100 in 17 sec
Oct 17, 2009 9:47:24 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 32, max: 20,37, avg: 20,27, div: 0,090 / 0,248 in 17 sec
Oct 17, 2009 9:47:42 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 33, max: 21,12, avg: 20,99, div: 0,078 / 0,082 in 17 sec
Oct 17, 2009 9:47:59 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 34, max: 21,56, avg: 21,49, div: 0,080 / 0,238 in 17 sec
Oct 17, 2009 9:48:16 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 35, max: 19,45, avg: 19,39, div: 0,058 / 0,063 in 17 sec
Oct 17, 2009 9:48:34 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 36, max: 19,69, avg: 19,29, div: 0,044 / 0,226 in 17 sec
Oct 17, 2009 9:48:51 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 37, max: 20,76, avg: 20,72, div: 0,039 / 0,052 in 17 sec
Oct 17, 2009 9:49:08 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 38, max: 20,45, avg: 20,45, div: 0,049 / 0,224 in 17 sec
Oct 17, 2009 9:49:25 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 39, max: 19,5, avg: 19,11, div: 0,042 / 0,046 in 17 sec
Oct 17, 2009 9:49:43 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 40, max: 19,42, avg: 19,37, div: 0,037 / 0,224 in 17 sec
Oct 17, 2009 9:50:00 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 41, max: 20,29, avg: 20,16, div: 0,052 / 0,048 in 17 sec
Oct 17, 2009 9:50:17 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 42, max: 19,77, avg: 19,72, div: 0,047 / 0,223 in 17 sec
Oct 17, 2009 9:50:35 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 43, max: 18,84, avg: 18,77, div: 0,042 / 0,044 in 17 sec
Oct 17, 2009 9:50:52 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 44, max: 20,73, avg: 20,69, div: 0,045 / 0,219 in 17 sec
Oct 17, 2009 9:51:09 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 45, max: 20,53, avg: 20,48, div: 0,053 / 0,042 in 17 sec
Oct 17, 2009 9:51:27 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 46, max: 19,69, avg: 19,67, div: 0,024 / 0,220 in 17 sec
Oct 17, 2009 9:51:44 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 47, max: 19,98, avg: 19,98, div: 0,025 / 0,040 in 17 sec
Oct 17, 2009 9:52:01 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 48, max: 20,83, avg: 20,83, div: 0,048 / 0,218 in 17 sec
Oct 17, 2009 9:52:19 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 49, max: 20,26, avg: 20,26, div: 0,034 / 0,040 in 17 sec
Oct 17, 2009 9:52:36 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 50, max: 20,37, avg: 20,37, div: 0,044 / 0,217 in 17 sec
```

```
Oct 17, 2009 9:52:53 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 51, max: 19,85, avg: 19,85, div: 0,039 / 0,040 in 17 sec
Oct 17, 2009 9:53:11 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 52, max: 19,81, avg: 19,81, div: 0,030 / 0,218 in 17 sec
Oct 17, 2009 9:53:28 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 53, max: 20,46, avg: 20,46, div: 0,040 / 0,035 in 17 sec
Oct 17, 2009 9:53:45 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 54, max: 20,42, avg: 20,42, div: 0,028 / 0,212 in 17 sec
Oct 17, 2009 9:54:02 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 55, max: 18,43, avg: 18,43, div: 0,026 / 0,030 in 17 sec
Oct 17, 2009 9:54:20 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 56, max: 20,36, avg: 20,36, div: 0,031 / 0,212 in 17 sec
Oct 17, 2009 9:54:37 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 57, max: 19,16, avg: 19,16, div: 0,026 / 0,030 in 17 sec
Oct 17, 2009 9:54:54 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 58, max: 19,28, avg: 19,28, div: 0,029 / 0,211 in 17 sec
Oct 17, 2009 9:55:11 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 59, max: 19,34, avg: 19,26, div: 0,029 / 0,029 in 17 sec
Oct 17, 2009 9:55:29 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 60, max: 20,52, avg: 20,22, div: 0,031 / 0,212 in 17 sec
Oct 17, 2009 9:55:46 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 61, max: 20,56, avg: 20,38, div: 0,025 / 0,030 in 17 sec
Oct 17, 2009 9:56:03 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 62, max: 20,77, avg: 20,77, div: 0,028 / 0,212 in 17 sec
Oct 17, 2009 9:56:20 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 63, max: 19,89, avg: 19,89, div: 0,032 / 0,027 in 17 sec
Oct 17, 2009 9:56:38 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 64, max: 20,24, avg: 20,24, div: 0,026 / 0,212 in 17 sec
Oct 17, 2009 9:56:55 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 65, max: 20,1, avg: 20,10, div: 0,029 / 0,026 in 17 sec
Oct 17, 2009 9:57:12 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 66, max: 20,47, avg: 20,47, div: 0,029 / 0,210 in 17 sec
Oct 17, 2009 9:57:30 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 67, max: 19,04, avg: 19,04, div: 0,024 / 0,026 in 17 sec
Oct 17, 2009 9:57:47 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 68, max: 19,53, avg: 19,53, div: 0,021 / 0,212 in 17 sec
Oct 17, 2009 9:58:04 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 69, max: 19,97, avg: 19,97, div: 0,021 / 0,023 in 17 sec
Oct 17, 2009 9:58:22 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 70, max: 19,45, avg: 19,45, div: 0,023 / 0,208 in 17 sec
Oct 17, 2009 9:58:39 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 71, max: 20,13, avg: 20,12, div: 0,027 / 0,020 in 17 sec
Oct 17, 2009 9:58:56 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 72, max: 19,51, avg: 19,46, div: 0,028 / 0,208 in 17 sec
Oct 17, 2009 9:59:13 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 73, max: 18,9, avg: 18,85, div: 0,026 / 0,022 in 17 sec
Oct 17, 2009 9:59:31 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 74, max: 20,03, avg: 20,03, div: 0,024 / 0,209 in 17 sec
```

```
Oct 17, 2009 9:59:48 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 75, max: 20,53, avg: 20,53, div: 0,022 / 0,024 in 17 sec
Oct 17, 2009 10:00:05 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 76, max: 21,52, avg: 21,52, div: 0,025 / 0,208 in 17 sec
Oct 17, 2009 10:00:22 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 77, max: 20,86, avg: 20,86, div: 0,020 / 0,020 in 17 sec
Oct 17, 2009 10:00:40 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 78, max: 19,65, avg: 19,65, div: 0,016 / 0,207 in 17 sec
Oct 17, 2009 10:00:57 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 79, max: 19,34, avg: 19,20, div: 0,019 / 0,022 in 17 sec
Oct 17, 2009 10:01:14 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 80, max: 19,27, avg: 19,27, div: 0,025 / 0,209 in 17 sec
Oct 17, 2009 10:01:32 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 81, max: 20,4, avg: 20,05, div: 0,024 / 0,021 in 17 sec
Oct 17, 2009 10:01:49 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 82, max: 20,52, avg: 20,52, div: 0,016 / 0,209 in 17 sec
Oct 17, 2009 10:02:06 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 83, max: 20,41, avg: 20,41, div: 0,023 / 0,020 in 17 sec
Oct 17, 2009 10:02:23 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 84, max: 19,69, avg: 19,69, div: 0,022 / 0,208 in 17 sec
Oct 17, 2009 10:02:41 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 85, max: 19,14, avg: 19,14, div: 0,018 / 0,019 in 17 sec
Oct 17, 2009 10:02:58 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 86, max: 19,32, avg: 19,32, div: 0,018 / 0,207 in 17 sec
Oct 17, 2009 10:03:15 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 87, max: 20,98, avg: 20,98, div: 0,019 / 0,019 in 17 sec
Oct 17, 2009 10:03:33 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 88, max: 20,44, avg: 20,44, div: 0,018 / 0,206 in 17 sec
Oct 17, 2009 10:03:50 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 89, max: 20,52, avg: 20,52, div: 0,022 / 0,019 in 17 sec
Oct 17, 2009 10:04:07 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 90, max: 19,63, avg: 19,63, div: 0,019 / 0,206 in 17 sec
Oct 17, 2009 10:04:25 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 91, max: 20,23, avg: 20,23, div: 0,020 / 0,020 in 17 sec
Oct 17, 2009 10:04:42 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 92, max: 20,2, avg: 20,20, div: 0,022 / 0,204 in 17 sec
Oct 17, 2009 10:04:59 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 93, max: 20,25, avg: 20,23, div: 0,023 / 0,019 in 17 sec
Oct 17, 2009 10:05:16 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 94, max: 20,32, avg: 20,32, div: 0,027 / 0,208 in 17 sec
Oct 17, 2009 10:05:34 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 95, max: 21,31, avg: 21,31, div: 0,016 / 0,021 in 17 sec
Oct 17, 2009 10:05:51 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 96, max: 20,82, avg: 20,68, div: 0,025 / 0,209 in 17 sec
Oct 17, 2009 10:06:08 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 97, max: 20,6, avg: 20,60, div: 0,026 / 0,023 in 17 sec
Oct 17, 2009 10:06:25 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 98, max: 20,37, avg: 20,37, div: 0,025 / 0,206 in 17 sec
```

```
Oct 17, 2009 10:06:42 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 99, max: 19,03, avg: 19,03, div: 0,022 / 0,023 in 17 sec
Oct 17, 2009 10:07:00 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 100, max: 20,28, avg: 20,28, div: 0,023 / 0,211 in 17 sec
Oct 17, 2009 10:12:45 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
Oct 17, 2009 10:12:45 PM ru,ifmo,ctddev,autoant,Ant2Problem main
INFO: result: 20,075[ 4 states, 0,04] in 345 sec
```

5.3.3. Построение автоматов с восемью состояниями

```
Oct 17, 2009 10:13:05 PM ru.ifmo.ctddev.autoant.fx.Genome <init>
INFO: [created] max: 9,81, avg: 3,11, div: 0,454 / 0,454 in 19 sec
Oct 17, 2009 10:13:24 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 1, max: 8,81, avg: 3,40, div: 0,437 / 0,446 in 19 sec
Oct 17, 2009 10:13:42 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 2, max: 9,26, avg: 4,08, div: 0,420 / 0,437 in 17 sec
Oct 17, 2009 10:13:59 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 3, max: 9,09, avg: 4,84, div: 0,422 / 0,428 in 17 sec
Oct 17, 2009 10:14:16 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 4, max: 8,66, avg: 5,47, div: 0,413 / 0,416 in 17 sec
Oct 17, 2009 10:14:33 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 5, max: 9,49, avg: 6,04, div: 0,411 / 0,404 in 17 sec
Oct 17, 2009 10:14:51 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 6, max: 10,98, avg: 6,22, div: 0,347 / 0,388 in 17 sec
Oct 17, 2009 10:15:08 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 7, max: 11,54, avg: 6,90, div: 0,355 / 0,375 in 17 sec
Oct 17, 2009 10:15:25 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 8, max: 11,14, avg: 7,50, div: 0,334 / 0,366 in 17 sec
Oct 17, 2009 10:15:42 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 9, max: 11,56, avg: 8,71, div: 0,313 / 0,351 in 17 sec
Oct 17, 2009 10:16:00 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 10, max: 12,0, avg: 9,75, div: 0,249 / 0,321 in 17 sec
Oct 17, 2009 10:16:17 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 11, max: 13,29, avg: 11,03, div: 0,257 / 0,278 in 17 sec
Oct 17, 2009 10:16:34 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 12, max: 14,7, avg: 11,66, div: 0,262 / 0,270 in 17 sec
Oct 17, 2009 10:16:51 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 13, max: 14,2, avg: 12,13, div: 0,236 / 0,253 in 17 sec
Oct 17, 2009 10:17:09 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 14, max: 14,29, avg: 12,77, div: 0,196 / 0,238 in 17 sec
Oct 17, 2009 10:17:26 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 15, max: 15,05, avg: 13,53, div: 0,173 / 0,217 in 17 sec
Oct 17, 2009 10:17:43 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 16, max: 15,16, avg: 13,95, div: 0,171 / 0,192 in 17 sec
Oct 17, 2009 10:18:01 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 17, max: 14,94, avg: 14,31, div: 0,124 / 0,301 in 17 sec
Oct 17, 2009 10:18:18 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 18, max: 14,94, avg: 14,45, div: 0,107 / 0,169 in 17 sec
Oct 17, 2009 10:18:35 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
```

```
INFO: [evolved] 19, max: 15,23, avg: 14,68, div: 0,147 / 0,281 in 17 sec
Oct 17, 2009 10:18:52 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 20, max: 15,53, avg: 15,25, div: 0,125 / 0,141 in 17 sec
Oct 17, 2009 10:19:10 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 21, max: 16,38, avg: 16,14, div: 0,121 / 0,269 in 17 sec
Oct 17, 2009 10:19:27 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 22, max: 15,75, avg: 15,55, div: 0,099 / 0,117 in 17 sec
Oct 17, 2009 10:19:44 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 23, max: 14,89, avg: 14,68, div: 0,099 / 0,257 in 17 sec
Oct 17, 2009 10:20:02 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 24, max: 15,13, avg: 14,62, div: 0,093 / 0,104 in 17 sec
Oct 17, 2009 10:20:19 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 25, max: 15,3, avg: 15,08, div: 0,108 / 0,258 in 17 sec
Oct 17, 2009 10:20:36 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 26, max: 15,96, avg: 15,79, div: 0,103 / 0,102 in 17 sec
Oct 17, 2009 10:20:53 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 27, max: 15,2, avg: 15,01, div: 0,068 / 0,246 in 17 sec
Oct 17, 2009 10:21:11 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 28, max: 15,1, avg: 14,88, div: 0,072 / 0,075 in 17 sec
Oct 17, 2009 10:21:28 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 29, max: 15,45, avg: 15,38, div: 0,077 / 0,239 in 17 sec
Oct 17, 2009 10:21:45 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 30, max: 16,44, avg: 16,15, div: 0,052 / 0,074 in 17 sec
Oct 17, 2009 10:22:03 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 31, max: 14,71, avg: 14,35, div: 0,064 / 0,233 in 17 sec
Oct 17, 2009 10:22:20 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 32, max: 15,51, avg: 15,43, div: 0,047 / 0,064 in 17 sec
Oct 17, 2009 10:22:37 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 33, max: 15,47, avg: 15,37, div: 0,066 / 0,231 in 17 sec
Oct 17, 2009 10:22:55 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 34, max: 15,53, avg: 15,36, div: 0,052 / 0,059 in 17 sec
Oct 17, 2009 10:23:12 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 35, max: 15,28, avg: 15,16, div: 0,063 / 0,229 in 17 sec
Oct 17, 2009 10:23:29 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 36, max: 15,03, avg: 14,92, div: 0,045 / 0,061 in 17 sec
Oct 17, 2009 10:23:46 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 37, max: 15,43, avg: 15,38, div: 0,048 / 0,227 in 17 sec
Oct 17, 2009 10:24:04 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 38, max: 15,74, avg: 15,11, div: 0,044 / 0,046 in 17 sec
Oct 17, 2009 10:24:21 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 39, max: 15,24, avg: 15,08, div: 0,048 / 0,225 in 17 sec
Oct 17, 2009 10:24:38 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 40, max: 15,73, avg: 15,54, div: 0,036 / 0,047 in 17 sec
Oct 17, 2009 10:24:55 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 41, max: 16,49, avg: 15,64, div: 0,039 / 0,222 in 17 sec
Oct 17, 2009 10:25:13 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 42, max: 15,63, avg: 15,56, div: 0,049 / 0,041 in 17 sec
Oct 17, 2009 10:25:30 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
```

```
INFO: [evolved] 43, max: 16,6, avg: 16,53, div: 0,049 / 0,222 in 17 sec
Oct 17, 2009 10:25:47 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 44, max: 15,81, avg: 15,78, div: 0,042 / 0,048 in 17 sec
Oct 17, 2009 10:26:05 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 45, max: 16,06, avg: 16,06, div: 0,038 / 0,221 in 17 sec
Oct 17, 2009 10:26:22 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 46, max: 14,64, avg: 14,60, div: 0,036 / 0,035 in 17 sec
Oct 17, 2009 10:26:39 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 47, max: 15,63, avg: 15,62, div: 0,047 / 0,221 in 17 sec
Oct 17, 2009 10:26:56 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 48, max: 15,76, avg: 15,73, div: 0,037 / 0,039 in 17 sec
Oct 17, 2009 10:27:14 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 49, max: 15,17, avg: 14,91, div: 0,034 / 0,214 in 17 sec
Oct 17, 2009 10:27:31 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 50, max: 15,86, avg: 15,77, div: 0,034 / 0,033 in 17 sec
Oct 17, 2009 10:27:48 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 51, max: 15,52, avg: 15,50, div: 0,033 / 0,215 in 17 sec
Oct 17, 2009 10:28:06 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 52, max: 15,69, avg: 15,67, div: 0,031 / 0,032 in 17 sec
Oct 17, 2009 10:28:23 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 53, max: 15,65, avg: 15,44, div: 0,029 / 0,214 in 17 sec
Oct 17, 2009 10:28:40 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 54, max: 15,91, avg: 15,91, div: 0,022 / 0,029 in 17 sec
Oct 17, 2009 10:28:58 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 55, max: 16,46, avg: 16,43, div: 0,014 / 0,210 in 17 sec
Oct 17, 2009 10:29:15 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 56, max: 15,18, avg: 15,18, div: 0,022 / 0,025 in 17 sec
Oct 17, 2009 10:29:32 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 57, max: 16,05, avg: 16,05, div: 0,016 / 0,209 in 17 sec
Oct 17, 2009 10:29:50 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 58, max: 16,0, avg: 15,97, div: 0,019 / 0,019 in 17 sec
Oct 17, 2009 10:30:07 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 59, max: 15,92, avg: 15,38, div: 0,020 / 0,208 in 17 sec
Oct 17, 2009 10:30:24 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 60, max: 15,95, avg: 15,69, div: 0,016 / 0,020 in 17 sec
Oct 17, 2009 10:30:42 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 61, max: 15,9, avg: 15,86, div: 0,013 / 0,207 in 17 sec
Oct 17, 2009 10:30:59 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 62, max: 16,45, avg: 16,38, div: 0,022 / 0,021 in 17 sec
Oct 17, 2009 10:31:16 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 63, max: 15,21, avg: 15,04, div: 0,013 / 0,205 in 17 sec
Oct 17, 2009 10:31:33 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 64, max: 16,07, avg: 16,07, div: 0,014 / 0,019 in 17 sec
Oct 17, 2009 10:31:51 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 65, max: 15,56, avg: 15,56, div: 0,017 / 0,206 in 17 sec
Oct 17, 2009 10:32:08 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 66, max: 16,07, avg: 16,07, div: 0,018 / 0,018 in 17 sec
Oct 17, 2009 10:32:25 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
```

```
INFO: [evolved] 67, max: 15,09, avg: 15,09, div: 0,015 / 0,205 in 17 sec
Oct 17, 2009 10:32:43 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 68, max: 15,01, avg: 15,00, div: 0,012 / 0,017 in 17 sec
Oct 17, 2009 10:33:00 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 69, max: 14,41, avg: 14,41, div: 0,016 / 0,205 in 17 sec
Oct 17, 2009 10:33:17 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 70, max: 15,89, avg: 15,89, div: 0,017 / 0,017 in 17 sec
Oct 17, 2009 10:33:35 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 71, max: 15,76, avg: 15,76, div: 0,010 / 0,205 in 17 sec
Oct 17, 2009 10:33:52 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 72, max: 16,52, avg: 16,51, div: 0,011 / 0,016 in 17 sec
Oct 17, 2009 10:34:09 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 73, max: 15,61, avg: 15,02, div: 0,015 / 0,205 in 17 sec
Oct 17, 2009 10:34:27 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 74, max: 15,81, avg: 15,64, div: 0,014 / 0,015 in 17 sec
Oct 17, 2009 10:34:44 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 75, max: 15,56, avg: 15,56, div: 0,018 / 0,206 in 17 sec
Oct 17, 2009 10:35:01 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 76, max: 15,46, avg: 15,40, div: 0,016 / 0,017 in 17 sec
Oct 17, 2009 10:35:18 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 77, max: 16,55, avg: 16,52, div: 0,016 / 0,204 in 17 sec
Oct 17, 2009 10:35:36 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 78, max: 15,39, avg: 15,39, div: 0,015 / 0,014 in 17 sec
Oct 17, 2009 10:35:53 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 79, max: 16,02, avg: 16,02, div: 0,011 / 0,203 in 17 sec
Oct 17, 2009 10:36:10 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 80, max: 15,73, avg: 15,73, div: 0,014 / 0,013 in 17 sec
Oct 17, 2009 10:36:28 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 81, max: 15,75, avg: 15,67, div: 0,013 / 0,203 in 17 sec
Oct 17, 2009 10:36:45 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 82, max: 14,99, avg: 14,99, div: 0,010 / 0,014 in 17 sec
Oct 17, 2009 10:37:02 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 83, max: 16,13, avg: 15,13, div: 0,012 / 0,203 in 17 sec
Oct 17, 2009 10:37:20 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 84, max: 15,67, avg: 15,63, div: 0,012 / 0,012 in 17 sec
Oct 17, 2009 10:37:37 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 85, max: 16,28, avg: 16,13, div: 0,012 / 0,202 in 17 sec
Oct 17, 2009 10:37:54 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 86, max: 16,82, avg: 16,25, div: 0,009 / 0,012 in 17 sec
Oct 17, 2009 10:38:12 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 87, max: 16,58, avg: 16,29, div: 0,011 / 0,205 in 17 sec
Oct 17, 2009 10:38:29 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 88, max: 16,5, avg: 16,50, div: 0,012 / 0,011 in 17 sec
Oct 17, 2009 10:38:46 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 89, max: 17,19, avg: 17,19, div: 0,013 / 0,201 in 17 sec
Oct 17, 2009 10:39:03 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 90, max: 14,84, avg: 14,84, div: 0,011 / 0,010 in 17 sec
Oct 17, 2009 10:39:21 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
```

```
INFO: [evolved] 91, max: 15,42, avg: 15,42, div: 0,009 / 0,203 in 17 sec
Oct 17, 2009 10:39:38 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 92, max: 16,85, avg: 16,80, div: 0,010 / 0,009 in 17 sec
Oct 17, 2009 10:39:55 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 93, max: 17,54, avg: 16,75, div: 0,009 / 0,201 in 17 sec
Oct 17, 2009 10:40:13 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 94, max: 17,81, avg: 16,29, div: 0,011 / 0,009 in 17 sec
Oct 17, 2009 10:40:30 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 95, max: 16,1, avg: 15,89, div: 0,006 / 0,200 in 17 sec
Oct 17, 2009 10:40:47 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 96, max: 16,74, avg: 16,74, div: 0,005 / 0,008 in 17 sec
Oct 17, 2009 10:41:04 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 97, max: 16,24, avg: 16,24, div: 0,002 / 0,199 in 17 sec
Oct 17, 2009 10:41:22 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 98, max: 16,64, avg: 16,61, div: 0,004 / 0,198 in 17 sec
Oct 17, 2009 10:41:39 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 99, max: 16,88, avg: 16,88, div: 0,003 / 0,196 in 17 sec
Oct 17, 2009 10:41:56 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 100, max: 17,67, avg: 17,67, div: 0,002 / 0,197 in 17 sec
Oct 17, 2009 10:47:41 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: result: 17,1205[ 8 states, 0,04] in 344 sec
```

5.3.4. Построение автоматов с шестнадцатью состояниями

```
Oct 17, 2009 10:47:58 PM ru.ifmo.ctddev.autoant.fx.Genome <init>
INFO: [created] max: 6,04, avg: 3,18, div: 0,456 / 0,454 in 17 sec
Oct 17, 2009 10:48:15 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 1, max: 7,31, avg: 3,32, div: 0,446 / 0,445 in 17 sec
Oct 17, 2009 10:48:33 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 2, max: 8,1, avg: 4,38, div: 0,430 / 0,436 in 17 sec
Oct 17, 2009 10:48:50 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 3, max: 8,18, avg: 4,33, div: 0,431 / 0,426 in 17 sec
Oct 17, 2009 10:49:08 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 4, max: 8,01, avg: 4,48, div: 0,419 / 0,418 in 17 sec
Oct 17, 2009 10:49:25 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 5, max: 7,54, avg: 4,74, div: 0,411 / 0,406 in 17 sec
Oct 17, 2009 10:49:42 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 6, max: 6,86, avg: 4,79, div: 0,388 / 0,396 in 17 sec
Oct 17, 2009 10:50:00 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 7, max: 7,58, avg: 5,07, div: 0,393 / 0,383 in 17 sec
Oct 17, 2009 10:50:17 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 8, max: 7,94, avg: 5,40, div: 0,375 / 0,373 in 17 sec
Oct 17, 2009 10:50:35 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 9, max: 9,8, avg: 6,36, div: 0,359 / 0,363 in 17 sec
Oct 17, 2009 10:50:52 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 10, max: 9,94, avg: 7,09, div: 0,346 / 0,358 in 17 sec
Oct 17, 2009 10:51:14 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 11, max: 13,06, avg: 7,91, div: 0,333 / 0,352 in 21 sec
Oct 17, 2009 10:51:31 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
```

```
INFO: [evolved] 12, max: 12,92, avg: 8,53, div: 0,315 / 0,339 in 17 sec
Oct 17, 2009 10:51:49 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 13, max: 13,39, avg: 10,22, div: 0,313 / 0,321 in 17 sec
Oct 17, 2009 10:52:06 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 14, max: 12,84, avg: 10,30, div: 0,300 / 0,305 in 17 sec
Oct 17, 2009 10:52:24 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 15, max: 14,85, avg: 11,37, div: 0,259 / 0,287 in 17 sec
Oct 17, 2009 10:52:41 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 16, max: 16,09, avg: 12,49, div: 0,271 / 0,279 in 17 sec
Oct 17, 2009 10:52:59 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 17, max: 14,34, avg: 12,59, div: 0,260 / 0,278 in 17 sec
Oct 17, 2009 10:53:16 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 18, max: 16,55, avg: 14,77, div: 0,258 / 0,269 in 17 sec
Oct 17, 2009 10:53:36 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 19, max: 15,38, avg: 14,75, div: 0,230 / 0,251 in 19 sec
Oct 17, 2009 10:53:55 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 20, max: 16,46, avg: 15,77, div: 0,222 / 0,248 in 19 sec
Oct 17, 2009 10:54:14 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 21, max: 16,27, avg: 15,50, div: 0,173 / 0,236 in 18 sec
Oct 17, 2009 10:54:32 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 22, max: 15,63, avg: 15,26, div: 0,225 / 0,232 in 18 sec
Oct 17, 2009 10:54:51 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 23, max: 15,82, avg: 15,53, div: 0,188 / 0,225 in 18 sec
Oct 17, 2009 10:55:09 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 24, max: 15,11, avg: 14,75, div: 0,187 / 0,210 in 18 sec
Oct 17, 2009 10:55:28 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 25, max: 16,67, avg: 15,66, div: 0,175 / 0,189 in 18 sec
Oct 17, 2009 10:55:46 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 26, max: 17,01, avg: 15,69, div: 0,166 / 0,296 in 18 sec
Oct 17, 2009 10:56:04 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 27, max: 16,2, avg: 15,64, div: 0,133 / 0,166 in 18 sec
Oct 17, 2009 10:56:23 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 28, max: 15,75, avg: 15,63, div: 0,148 / 0,287 in 18 sec
Oct 17, 2009 10:56:41 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 29, max: 16,53, avg: 15,73, div: 0,143 / 0,158 in 18 sec
Oct 17, 2009 10:56:59 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 30, max: 16,17, avg: 15,98, div: 0,153 / 0,284 in 18 sec
Oct 17, 2009 10:57:18 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 31, max: 16,98, avg: 15,97, div: 0,149 / 0,150 in 18 sec
Oct 17, 2009 10:57:36 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 32, max: 16,14, avg: 15,29, div: 0,144 / 0,278 in 18 sec
Oct 17, 2009 10:57:54 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 33, max: 15,53, avg: 15,39, div: 0,129 / 0,143 in 18 sec
Oct 17, 2009 10:58:12 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 34, max: 15,67, avg: 15,43, div: 0,127 / 0,274 in 18 sec
Oct 17, 2009 10:58:31 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 35, max: 15,84, avg: 15,63, div: 0,131 / 0,129 in 18 sec
Oct 17, 2009 10:58:49 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
```

```
INFO: [evolved] 36, max: 15,28, avg: 15,12, div: 0,116 / 0,268 in 18 sec
Oct 17, 2009 10:59:08 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 37, max: 16,34, avg: 16,23, div: 0,110 / 0,117 in 18 sec
Oct 17, 2009 10:59:26 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 38, max: 15,48, avg: 15,36, div: 0,084 / 0,252 in 18 sec
Oct 17, 2009 10:59:44 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 39, max: 15,38, avg: 15,14, div: 0,090 / 0,085 in 18 sec
Oct 17, 2009 11:00:03 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 40, max: 15,55, avg: 15,47, div: 0,081 / 0,247 in 18 sec
Oct 17, 2009 11:00:21 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 41, max: 16,35, avg: 16,25, div: 0,075 / 0,087 in 18 sec
Oct 17, 2009 11:00:39 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 42, max: 15,95, avg: 15,89, div: 0,088 / 0,242 in 18 sec
Oct 17, 2009 11:00:58 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 43, max: 15,94, avg: 15,88, div: 0,082 / 0,078 in 18 sec
Oct 17, 2009 11:01:16 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 44, max: 15,77, avg: 15,76, div: 0,079 / 0,240 in 18 sec
Oct 17, 2009 11:01:34 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 45, max: 15,88, avg: 15,84, div: 0,072 / 0,076 in 18 sec
Oct 17, 2009 11:01:53 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 46, max: 16,42, avg: 16,36, div: 0,069 / 0,236 in 18 sec
Oct 17, 2009 11:02:11 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 47, max: 15,47, avg: 15,43, div: 0,070 / 0,071 in 18 sec
Oct 17, 2009 11:02:29 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 48, max: 15,9, avg: 15,84, div: 0,059 / 0,234 in 18 sec
Oct 17, 2009 11:02:47 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 49, max: 14,82, avg: 14,82, div: 0,064 / 0,067 in 18 sec
Oct 17, 2009 11:03:05 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 50, max: 16,96, avg: 16,90, div: 0,055 / 0,231 in 17 sec
Oct 17, 2009 11:03:24 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 51, max: 16,28, avg: 16,11, div: 0,057 / 0,059 in 18 sec
Oct 17, 2009 11:03:42 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 52, max: 15,87, avg: 15,86, div: 0,054 / 0,228 in 18 sec
Oct 17, 2009 11:04:00 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 53, max: 15,11, avg: 15,06, div: 0,046 / 0,054 in 18 sec
Oct 17, 2009 11:04:19 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 54, max: 15,61, avg: 15,60, div: 0,045 / 0,225 in 18 sec
Oct 17, 2009 11:04:37 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 55, max: 16,12, avg: 14,77, div: 0,047 / 0,049 in 18 sec
Oct 17, 2009 11:04:55 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 56, max: 15,42, avg: 14,66, div: 0,043 / 0,225 in 18 sec
Oct 17, 2009 11:05:13 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 57, max: 15,7, avg: 15,68, div: 0,049 / 0,050 in 18 sec
Oct 17, 2009 11:05:32 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 58, max: 15,27, avg: 15,18, div: 0,057 / 0,224 in 18 sec
Oct 17, 2009 11:05:50 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 59, max: 15,21, avg: 15,19, div: 0,051 / 0,050 in 18 sec
Oct 17, 2009 11:06:08 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
```

```
INFO: [evolved] 60, max: 13,89, avg: 13,80, div: 0,046 / 0,224 in 18 sec
Oct 17, 2009 11:06:26 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 61, max: 14,85, avg: 14,71, div: 0,045 / 0,045 in 18 sec
Oct 17, 2009 11:06:44 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 62, max: 16,73, avg: 16,73, div: 0,046 / 0,221 in 18 sec
Oct 17, 2009 11:07:03 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 63, max: 15,74, avg: 15,74, div: 0,041 / 0,041 in 18 sec
Oct 17, 2009 11:07:21 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 64, max: 15,37, avg: 15,34, div: 0,043 / 0,219 in 17 sec
Oct 17, 2009 11:07:39 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 65, max: 15,35, avg: 15,35, div: 0,037 / 0,042 in 18 sec
Oct 17, 2009 11:07:57 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 66, max: 15,44, avg: 15,41, div: 0,036 / 0,218 in 18 sec
Oct 17, 2009 11:08:16 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 67, max: 16,05, avg: 16,01, div: 0,040 / 0,041 in 18 sec
Oct 17, 2009 11:08:34 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 68, max: 15,5, avg: 15,41, div: 0,035 / 0,217 in 18 sec
Oct 17, 2009 11:08:52 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 69, max: 15,79, avg: 15,73, div: 0,031 / 0,037 in 17 sec
Oct 17, 2009 11:09:10 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 70, max: 15,19, avg: 15,19, div: 0,038 / 0,214 in 17 sec
Oct 17, 2009 11:09:27 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 71, max: 16,44, avg: 16,44, div: 0,031 / 0,035 in 17 sec
Oct 17, 2009 11:09:45 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 72, max: 16,23, avg: 16,23, div: 0,032 / 0,213 in 17 sec
Oct 17, 2009 11:10:02 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 73, max: 16,03, avg: 15,92, div: 0,029 / 0,029 in 17 sec
Oct 17, 2009 11:10:20 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 74, max: 15,73, avg: 15,37, div: 0,029 / 0,212 in 17 sec
Oct 17, 2009 11:10:37 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 75, max: 15,4, avg: 15,40, div: 0,027 / 0,029 in 17 sec
Oct 17, 2009 11:10:55 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 76, max: 16,53, avg: 16,53, div: 0,027 / 0,211 in 17 sec
Oct 17, 2009 11:11:13 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 77, max: 16,21, avg: 16,21, div: 0,027 / 0,026 in 17 sec
Oct 17, 2009 11:11:30 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 78, max: 14,88, avg: 14,88, div: 0,024 / 0,211 in 17 sec
Oct 17, 2009 11:11:48 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 79, max: 14,3, avg: 14,30, div: 0,023 / 0,025 in 17 sec
Oct 17, 2009 11:12:05 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 80, max: 15,26, avg: 15,10, div: 0,021 / 0,209 in 17 sec
Oct 17, 2009 11:12:23 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 81, max: 15,02, avg: 15,02, div: 0,021 / 0,021 in 17 sec
Oct 17, 2009 11:12:40 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 82, max: 15,35, avg: 15,35, div: 0,019 / 0,208 in 17 sec
Oct 17, 2009 11:12:58 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 83, max: 17,02, avg: 17,02, div: 0,013 / 0,019 in 17 sec
Oct 17, 2009 11:13:20 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
```

```
INFO: [evolved] 84, max: 16,0, avg: 16,00, div: 0,017 / 0,205 in 22 sec
Oct 17, 2009 11:13:38 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 85, max: 15,15, avg: 15,15, div: 0,017 / 0,015 in 17 sec
Oct 17, 2009 11:13:55 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 86, max: 14,98, avg: 14,98, div: 0,012 / 0,204 in 17 sec
Oct 17, 2009 11:14:13 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 87, max: 16,19, avg: 16,19, div: 0,013 / 0,014 in 17 sec
Oct 17, 2009 11:14:30 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 88, max: 16,23, avg: 16,23, div: 0,013 / 0,204 in 17 sec
Oct 17, 2009 11:14:48 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 89, max: 16,23, avg: 16,12, div: 0,014 / 0,014 in 17 sec
Oct 17, 2009 11:15:05 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 90, max: 15,6, avg: 14,92, div: 0,018 / 0,205 in 17 sec
Oct 17, 2009 11:15:23 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 91, max: 16,17, avg: 16,17, div: 0,017 / 0,015 in 17 sec
Oct 17, 2009 11:15:41 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 92, max: 15,51, avg: 15,51, div: 0,015 / 0,205 in 17 sec
Oct 17, 2009 11:15:58 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 93, max: 16,17, avg: 16,17, div: 0,017 / 0,015 in 17 sec
Oct 17, 2009 11:16:16 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 94, max: 15,85, avg: 15,85, div: 0,018 / 0,204 in 17 sec
Oct 17, 2009 11:16:33 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 95, max: 14,65, avg: 14,60, div: 0,016 / 0,015 in 17 sec
Oct 17, 2009 11:16:51 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 96, max: 17,06, avg: 17,06, div: 0,015 / 0,204 in 17 sec
Oct 17, 2009 11:17:08 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 97, max: 15,9, avg: 15,90, div: 0,009 / 0,013 in 17 sec
Oct 17, 2009 11:17:26 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 98, max: 15,9, avg: 15,90, div: 0,010 / 0,203 in 17 sec
Oct 17, 2009 11:17:43 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 99, max: 14,93, avg: 14,93, div: 0,009 / 0,011 in 17 sec
Oct 17, 2009 11:18:01 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: [evolved] 100, max: 14,89, avg: 14,89, div: 0,010 / 0,202 in 17 sec
Oct 17, 2009 11:23:49 PM ru.ifmo.ctddev.autoant.fx.Genome evolve
INFO: result: 15,531[ 16 states, 0,04] in 347 sec
```

Выводы по главе 5

1. Проведена апробация метода, разработанного в главе 4, на примере задачи «Умный муравей–3».
2. Приведены протоколы экспериментов для построения управляющего автомата для задачи «Умный муравей-3» с двумя, четырьмя, восемью и шестнадцатью состояниями.
3. На следующих этапах работы будет выполнено экспериментальное исследование разработанного метода на задаче построения системы управления беспилотным летательным аппаратом.

ЗАКЛЮЧЕНИЕ

В результате исследований, выполненных на первом этапе работ по контракту, были выполнены следующие работы:

- выполнение аналитического обзора;
- выбор и обоснование оптимального варианта направления исследований;
- подготовка плана проведения теоретических и экспериментальных исследований;
- разработка и программная реализация метода генетического программирования, основанного на использовании линейных бинарных графов для представления функции переходов автоматов;
- экспериментальное исследование метода генетического программирования, основанного на использовании линейных бинарных графов для представления функции переходов автоматов, на примере построения автомата управления системой со сложным поведением в задаче «Умный муравей–3».

Аналитический обзор был выполнен по следующим направлениям:

- системы со сложным поведением и автоматное программирование;
- методы построения конечных автоматов с помощью генетических алгоритмов и генетического программирования;
- программы-симуляторы летательных аппаратов.

В результате выполнения аналитического обзора разработан план проведения теоретических и экспериментальных исследований. В соответствии с этим планом в рамках теоретических исследований будут разработаны следующие методы генетического программирования:

- метод генетического программирования, основанный на использовании линейных бинарных графов для представления функции переходов автоматов;
- метод построения управляющих конечных автоматов на основе обучающих примеров с помощью генетического программирования;
- метод двухэтапного генетического программирования;
- метод применения верификации моделей при вычислении функции приспособленности.

В соответствии с планом экспериментальных исследований будут проведены эксперименты с одной из следующих программ-симуляторов летательных аппаратов: *Microsoft Flight Simulator*, *X-Plane*, *FlightGear*, *AeroSim*. По результатам экспериментальных исследований будет проведена модификация разработанных методов. В заключение работы будет проведено обобщение и оценка результатов исследований.

Результаты выполненных работ позволяют утверждать, что научно-технический уровень исследований превышает уровень 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. *Заде Л., Дезоер Ч.* Теория линейных систем. Метод пространства состояний. М.: Наука, 1970.
6. Материалы сайта http://www.microsoft.com/games/flightsimulatorX/
7. Материалы сайта http://www.x-plane.com/
8. Материалы сайта http://www.flightgear.org/
9. Материалы сайта http://www.u-dynamics.com/aerosim/
10. Материалы сайта http://www.cloudcaptech.com/
11. *Непейвода Н. Н.* Стили и методы программирования. М.: Интернет-Университет Информационных технологий, 2005.
12. *Николенко С. И.* Лекции по генетическим алгоритмам. http://logic.pdmi.ras.ru/~sergey/teaching/ml/
13. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирования. СПб: Питер, 2009.
14. *Полимеразная цепная реакция.* http://en.wikipedia.org/wiki/Polymerase_chain_reaction
15. *Туккель Н. И., Шалыто А. А.* Система управления дизель-генератором (фрагмент). Программирование с явным выделением состояний. Проектная документация. http://is.ifmo.ru/projects/dg/
16. *Туккель Н. И., Шалыто А. А.* От тьюрингова программирования к автоматному // Мир ПК. 2002. № 2, с. 144–149. http://is.ifmo.ru/works/turing/
17. *Хопкрофт Дж., Мотвани Р., Ульман Дж.* Введение в теорию автоматов, языков и вычислений. М.: Вильямс, 2002.
18. *Царев Ф. Н., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов с минимальным числом состояний для задачи об «Умном муравье» / Тезисы научно-технической конференции «Научно-программное обеспечение в образовании и научных исследованиях». СПбГУ ПУ. 2008, с. 209–215.
19. *Шалыто А. А.* Switch-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
20. *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
21. *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
22. *Armstrong, D.* A programmed algorithm for assigning internal codes to sequential machines // IRE Transactions on Electronic Computers. EC.11. 1962. I.4, pp.466–472.
23. *Axelrod R.* The Evolution of Cooperation. NY: Basic Books, 1984.

24. *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
25. *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
26. *Benson K.* Evolving Finite State Machines with Embedded Genetic Programming for Automatic Target Detection. http://ieeexplore.ieee.org/iel5/6997/18853/00870838.pdf
27. *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
28. *Chambers L.* Practical Handbook of Genetic Algorithms. Complex Coding Systems. Volumes I. II, III. CRC Press, 1999.
29. *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
30. *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
31. *Fogel D.* The Evolution of Intelligent Decision Making in Gaming // Cybernetics and Systems. 2001. 22, pp. 223–236.
32. *Fogel L.* Autonomous Automata // Industrial Research. 1962. V.4, pp. 14–19.
33. *Fogel L., Owens A., Walsh M.* Artificial Intelligence through Simulated Evolution. NY: Wiley, 1966.
34. *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
35. *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
36. *Gold E. M.* Complexity of automaton identification from given data // Information and Control. 1978. № 37, pp. 302–320.
37. *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.
38. *Hillis W.* Co-Evolving Parasites Improve Simulated Evolution as an Optimization Procedure / In Artificial Life II. MA: Addison-Wesley. 1992, pp. 313–324.
39. *Holland J.* Adaptation in Natural and Artificial Systems. University of Michigan Press, 1975.
40. *Holland J.* ECHO: Explorations of Evolution in a Miniature World / Proceedings of the Second Conference on Artificial Life. Redwood City. CA: Addison-Wesley, 1990.
41. *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

42. *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
43. *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
44. *Juillie H., Pollack J.* Coevolving the «Ideal» Trainer: Application to the Discovery of Cellular Automata Rules. 1998. http://citeseer.ist.psu.edu/16712.html
45. *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
46. *Koza J.* Genetic programming. On the Programming of Computers by Means of Natural Selection. MA: The MIT Press, 1998.
47. *Levy S.* Artificial Life: The Quest for a New Creation. New York: Pantheon. 1992, 380 p.
48. *Linton R.* Adapting binary fitness functions in genetic algorithms / Proceedings of the 42nd annual Southeast regional conference. NY: ACM Press. 2004, pp. 391–395.
49. *Lucas M., Reynolds T.J.* Learning DFA: Evolution versus Evidence Driven State Merging.
50. *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
51. *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.
52. *Mitchell M.* An Introduction to Genetic Algorithms. MA: The MIT Press, 1996.
53. *Mitchell M., Crutchfield J. P., Hraber P. T.* Evolving cellular automata to perform computations. Physica D. 75. 1993, pp. 361–391. http://web.cecs.pdx.edu/~mm/mech-imped.pdf
54. *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
55. *Nedjah N., Mourelle L.* Mealy Finite State Machines: An Evolutionary Approach // International Journal of Innovative Computing, Information and Control. 2006. V.2, I. 4.
56. *Ray T.* An Approach to the Synthesis of Life / In Artificial Life II. MA: Addison-Wesley 1992, pp. 371–408.
57. *Reynolds C. W.* Competition, Coevolution and the Game of Tag / Proceedings of Artificial Life IV. Cambridge. MA: MIT Pressю. 1994, pp. 59–69. http://www.red3d.com/cwr/papers/1994/alife4.html
58. *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
59. *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
60. *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
61. *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
62. *Bryant R. E.* Graph-based algorithms for boolean function manipulation / IEEE Transactions on Computers, 35. 1986. № 8, pp. 677–691.

63. *Бедный Ю. Д., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов в задаче «Умный муравей». СПбГУИТМО, 2007. http://is.ifmo.ru/works/ant
64. *Данилов В. Р.* Методы представления функции переходов при генерации автоматов управления на основе генетического программирования. Магистерская диссертация. СПбГУ ИТМО, 2009.
65. *Данилов В. Р.* Метод представления автоматов деревьями решений для использования в генетическом программировании //Научно-технический вестник СПбГУ ИТМО. Автоматное программирование. 2008. Выпуск 53, с. 103–108.
66. *Гладков Л. А, Курейчик В. В., Курейчик В. М.* Генетические алгоритмы. М.: Физматлит, 2006.
67. *Шалыто А. А.* Логическое управление. Методы аппаратной и программной реализации. СПб.: Наука. 2000, 780 с.
68. *Kantschik W., Dittrich P., Brameier M., Banzhaf W.* Empirical Analysis of Different Levels of Meta-Evolution / Proceedings of the Congress on Evolutionary Computation. Washington D.C.: IEEE Press. 1999, pp. 2086–2093. http://citeseer.ist.psu.edu/kantschik99empirical.html
69. *Kantschik W., Banzhaf W.* Linear-Graph GP. A new GP Structure / Proceedings of the 4th European Conference on Genetic Programming, EuroGP 2002. Kinsale: Springer-Verlag. 2002, pp. 83–92. http://citeseer.ist.psu.edu/kantschik02lineargraph.html
70. *Kantschik W., Dittrich P., Brameier M., Banzhaf W.* Meta-Evolution in Graph GP / Genetic Programming: Second European Workshop (EuroGP'99). Berlin: Springer. 1999, pp. 15–28. http://citeseer.ist.psu.edu/kantschik99metaevolution.html
71. *Nedjah N., Mourelle L.* Mealy Finite State Machines: An Evolutionary Approach // International Journal of Innovative Computing, Information and Control. 2006. V.2, I. 4.
72. *Petrovic P.* Simulated evolution of distributed FSA behaviour-based arbitration. Poster at the Eighth Scandinavian Conference on Artificial Intelligence (SCAI'03). 2003. http://www.idi.ntnu.no/grupper/ai/eval/incremental/scai03-poster-petrovic.pdf
73. *Petrovic P.* Evolving automatons for distributed behavior arbitration. Technical Report. Norwegian University of Science and Technology. 2005. http://www.idi.ntnu.no/~petrovic/shortpaper.pdf
74. *Petrovic P.* Comparing Finite-State Automata Representation with GP-trees. Technical Report. Norwegian University of Science and Technology. 2006. http://www.idi.ntnu.no/~petrovic/fsatr/fsatr.pdf
75. *Поликарпова Н. И., Точилин В. Н., Шалыто А. А.* Применение генетического программирования для реализации систем со сложным поведением. / Сборник трудов VI-ой Международной научно-практической конференции «Интегрированные модели и мягкие вычисления в искусственном интеллекте». Том 2. М.: Физматлит. 2007, с. 598–604. http://is.ifmo.ru/genalg/_polikarpova.pdf
76. Davydov A., Sokolov D., Tsarev F. Application of Genetic Algorithms for Construction of Moore Automaton and Systems of Interacting Mealy Automata in «Artificial Ant» Problem. / Proceedings of the Second Spring Young Researchers' Colloquium on Software Engineering. SPbSU. 2008. V. 1, pp. 51–54.