Министерство образования и науки Российской Федерации

УДК 004.4’242 ГРНТИ 28.23.29 Инв. №

УТВЕРЖДЕНО:
Исполнитель: **Государственное образовательное учреждение высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики»**
От имени Руководителя организации **_____/В. Н. Васильев/** **М.П.**

НАУЧНО-ТЕХНИЧЕСКИЙ ОТЧЕТ

о выполнении 2 этапа Государственного контракта
№ П2236 от 11 ноября 2009 г. и Дополнению от 15 марта 2010 г. № 1/П2236

Исполнитель: Государственное образовательное учреждение высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики»
Программа (мероприятие): Федеральная целевая программа «Научные и научно-педагогические кадры инновационной России» на 2009-2013 гг., в рамках реализации мероприятия № 1.2.1 Проведение научных исследований научными группами под руководством докторов наук.
Проект: Применение методов искусственного интеллекта в разработке управляющих программных систем
Руководитель проекта: **_____/Шалыто Анатолий Абрамович** (подпись)

Санкт-Петербург
2010 г.

СПИСОК ОСНОВНЫХ ИСПОЛНИТЕЛЕЙ

по Государственному контракту П2236 от 11 ноября 2009 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:

доктор технических наук, профессор	_____ подпись, дата	Шалыто А. А.

Исполнители темы:

кандидат технических наук, без ученого звания	_____ подпись, дата	Шопырин Д. Г.
кандидат технических наук, доцент	_____ подпись, дата	Гуров В. С.
кандидат технических наук, без ученого звания	_____ подпись, дата	Корнеев Г. А.
без ученой степени, без ученого звания	_____ подпись, дата	Царев Ф. Н.
без ученой степени, без ученого звания	_____ подпись, дата	Клебан В. О.
без ученой степени, без ученого звания	_____ подпись, дата	Егоров К. В.

без ученой степени, без ученого звания _____ Суясов Д. И.
подпись, дата

без ученой степени, без ученого звания _____ Кошевой А. А.
подпись, дата

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

без ученой степени, без ученого звания _____ Скорынин П. А.
подпись, дата

без ученой степени, без ученого звания _____ Попов С. И.
подпись, дата

без ученой степени, без ученого звания _____ Попов Ю. И.
подпись, дата

без ученой степени, без ученого звания _____ Буздалов М. В.
подпись, дата

без ученой степени, без ученого звания _____ Малаховски Я. М.
подпись, дата

без ученой степени, без ученого звания _____ Федотов П. В.
подпись, дата

РЕФЕРАТ

Отчет 138 с., 3 гл., 76 рис., 10 табл., 50 источников.

Ключевые слова: искусственный интеллект; управляющие системы; программные системы; генетические алгоритмы; коэволюция.

В настоящем отчете излагаются результаты выполнения *поисковых научно-исследовательских работ по направлениям «Механика», «Информатика», «Математика» по проблеме «Применение методов искусственного интеллекта в разработке управляющих программных систем»*, 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 22 октября 2009 г. № 3/НК-408П) по конкурсу № НК-408П «Проведение поисковых научно-исследовательских работ по направлениям: «Механика», «Информатика», «Математика» в рамках мероприятия 1.2.1 «Проведение научных исследований научными группами под руководством докторов наук» по направлению 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. Разработка, программная реализация и экспериментальное исследование метода применения генетического программирования для построения систем управления виртуальными роботами для упрощенной модели игры в футбол;
2. Разработка, программная реализация и экспериментальное исследование метода применения коэволюции для построения взаимодействующих управляющих конечных автоматов;
3. Разработка, программная реализация и экспериментальное исследование метода применения коэволюции для построения управляющих конечных автоматов, решающих задачи навигации;
4. Разработка, программная реализация и экспериментальное исследование метода совместного применения конечных автоматов и нейронных сетей;
5. Разработка метода применения верификации моделей программ при построении управляющих конечных автоматов с помощью генетического программирования.

При выполнении второго этапа работ использовался следующий инструментарий:

1. Работы членов коллектива, опубликованные в рамках НИР по схожей тематике.
2. Язык программирования Java.
3. Интегрированная среда разработки Eclipse.
4. Персональный компьютер.
5. Требования к оформлению публикаций.
6. Часть 4 Гражданского кодекса Российской Федерации.
7. ГОСТ 7.32-2001 «Отчет о научно-исследовательской работе. Структура и правила оформления».

Описываются метод применения генетического программирования для построения систем управления виртуальными роботами для упрощенной модели игры в футбол, метод применения

коэволюции для построения взаимодействующих управляющих конечных автоматов, метод применения коэволюции для построения управляющих конечных автоматов, решающих задачи навигации, метод совместного применения конечных автоматов и нейронных сетей, метод применения верификации моделей программ при построении управляющих конечных автоматов с помощью генетического программирования. Для указанных методов приведены их программные реализации и результаты экспериментальных исследований.

К отчету приложены копии текстов статей, опубликованных по результатам второго этапа работ, а также копии свидетельств о регистрации программы для ЭВМ.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящем отчете излагаются результаты выполнения *поисковых научно-исследовательских работ по направлениям «Механика», «Информатика», «Математика» по проблеме «Применение методов искусственного интеллекта в разработке управляющих программных систем»*, 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 22 октября 2009 г. № 3/НК-408П) по конкурсу № НК-408П «Проведение поисковых научно-исследовательских работ по направлениям: «Механика», «Информатика», «Математика» в рамках мероприятия 1.2.1 «Проведение научных исследований научными группами под руководством докторов наук» по направлению 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. Разработка, программная реализация и экспериментальное исследование метода применения генетического программирования для построения систем управления виртуальными роботами для упрощенной модели игры в футбол;
2. Разработка, программная реализация и экспериментальное исследование метода применения коэволюции для построения взаимодействующих управляющих конечных автоматов;
3. Разработка, программная реализация и экспериментальное исследование метода применения коэволюции для построения управляющих конечных автоматов, решающих задачи навигации;
4. Разработка, программная реализация и экспериментальное исследование метода совместного применения конечных автоматов и нейронных сетей;
5. Разработка метода применения верификации моделей программ при построении управляющих конечных автоматов с помощью генетического программирования.

Отчет имеет следующую структуру. В первой главе приводятся описания, программная реализация и результаты экспериментального исследования следующих методов:

- метод применения генетического программирования для построения систем управления виртуальными роботами для упрощенной модели игры в футбол;
- метод применения коэволюции для построения взаимодействующих управляющих конечных автоматов;
- метод применения коэволюции для построения управляющих конечных автоматов, решающих задачи навигации;
- метод совместного применения конечных автоматов и нейронных сетей;
- метод применения верификации моделей программ при построении управляющих конечных автоматов с помощью генетического программирования.

Во второй главе приводятся результаты теоретических и экспериментальных исследований, включающих систематизацию и предварительную оценку полученных результатов, оценку полноты решения задач и достижения поставленных целей, сопоставление и обобщение результатов анализа научно-информационных источников и теоретических и экспериментальных исследований, оценку эффективности полученных результатов в сравнении с современным научно-техническим уровнем, рекомендации по возможности использования результатов проведенных НИР в реальном секторе

экономики, рекомендации по использованию результатов НИР при создании научно-образовательных курсов.

В третьей главе приводятся результаты публикации результатов НИР, включающие копии двух свидетельств о регистрации программы для ЭВМ, заключение экспертной комиссии по открытому опубликованию, а также копии трех статей, опубликованных в журналах ВАК со ссылкой на проведение НИР в рамках реализации ФЦП «Научные и научно-педагогические кадры инновационной России» на 2009 – 2013 годы.

В заключении дается общая оценка работ по этапу.

Исследования, проводимые, например, в университетах Йорка, Брунеля (Великобритания), в исследовательском центре *British Telecom*, в университете Дрекселя (Филадельфия, США), в университет страны басков (Испания), показывают, что методы поиска в условиях ограничений могут успешно применяться для решения ряда задач, возникающих при разработке программных систем. В число таких задач входят: автоматическое построение тестов, разбиение программ на модули, прогнозирование времени отклика.

Методы поиска в условиях ограничений, в свою очередь, являются одним из разделов искусственного интеллекта. Среди других разделов искусственного интеллекта, которые могут найти применение при построении управляющих программных систем, можно назвать: мультиагентные системы, обучение с подкреплением, искусственные нейронные сети, робототехника, методы поиска при условиях ограничений, представление знаний, логический вывод, планирование действий, методы вероятностных рассуждений и рассуждений в условиях неопределенности, в том числе нечеткие множества и нечеткая логика, обработка естественного языка, машинное зрение.

Одним из важнейших направлений искусственного интеллекта является создание роботов. Исследования по этому направлению ведутся в большом числе научных центров и университетов во всем мире.

В большинстве работ методы искусственного интеллекта применяются либо при программной реализации управляющей системы, либо для решения задач обеспечения процесса разработки. В настоящей работе делается попытка распространить область применения методов искусственного интеллекта на задачи автоматизированного построения управляющих программных систем. При этом в качестве предметной области выбрано построение систем управления малоразмерными мобильными роботами.

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

1. АНАЛИТИЧЕСКИЙ ОТЧЕТ О ПРОВЕДЕНИИ ТЕОРЕТИЧЕСКИХ И ЭКСПЕРИМЕНТАЛЬНЫХ ИССЛЕДОВАНИЙ

В настоящем разделе приведен аналитический отчет о проведении теоретических и экспериментальных исследований.

1.1. РАЗРАБОТКА, ПРОГРАММНАЯ РЕАЛИЗАЦИЯ И ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ МЕТОДА ПРИМЕНЕНИЯ ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ ДЛЯ ПОСТРОЕНИЯ СИСТЕМ УПРАВЛЕНИЯ ВИРТУАЛЬНЫМИ РОБОТАМИ ДЛЯ УПРОЩЕННОЙ МОДЕЛИ ИГРЫ В ФУТБОЛ

В настоящем разделе приводится описание метода применения генетического программирования для построения систем управления виртуальными роботами для упрощенной модели игры в футбол.

1.1.1. Игра «Виртуальный футбол»

Игра для программистов «Виртуальный футбол» представляет собой эмулятор виртуального мира, в котором две команды роботов соревнуются друг с другом в мастерстве игры в футбол. Примером аналогичной платформы является программный продукт *The RoboCup Soccer Simulator* [12].

Команда, принимающая участие в игре, должна состоять из одного или нескольких игроков. Каждый игрок является виртуальным роботом под управлением программы, написанной на языке программирования *Java*. В нашем случае каждая команда будет состоять ровно из одного игрока.

Эмулятор игры поставляется в виде отдельной программы, для запуска которой необходимо наличие на компьютере пользователя установленной *JRE* (*Java Runtime Environment)* версии не ниже *1.5*. После создания, команда роботов помещается на виртуальное футбольное поле, на котором ей предстоит встретиться с другой командой подобных роботов. Процесс игры соперников друг против друга можно наблюдать, используя тот же эмулятор (рис. 1).

Рис. 1. Окно визуализации

1.1.2. Объекты игры «Виртуальный футбол»

Процесс игры в «Виртуальный футбол» проходит на двумерном прямоугольном поле, периметр которого ограничен жесткими стенками. С левой и правой стороны его расположены ворота. В середине поля размечен центральный круг. На рис. 2 представлена подробная схема игрового поля.

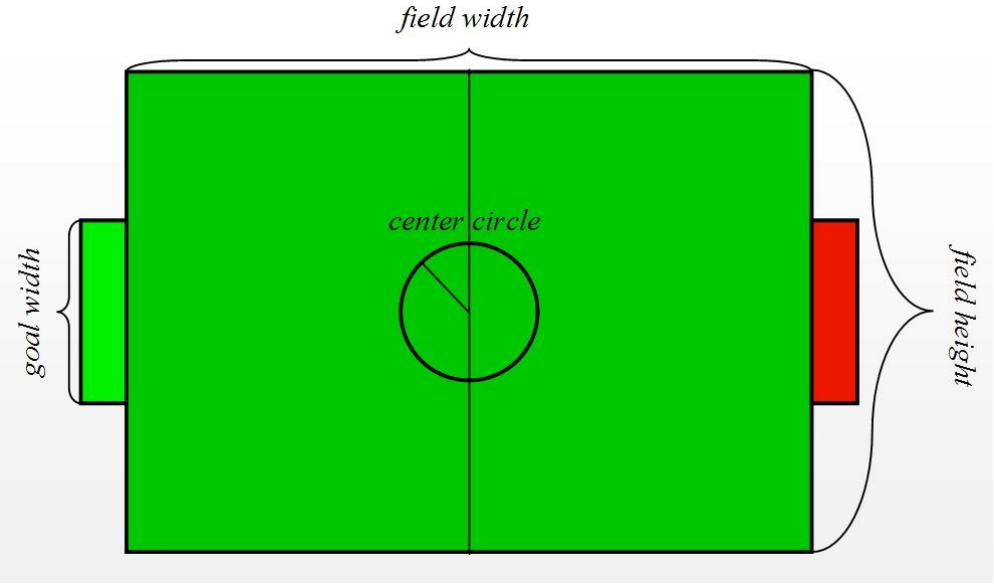


Рис. 2. Схема игрового поля

Динамическими объектами игры являются игроки и мяч. Они представляют собой объекты цилиндрической формы. Игроки делятся на две команды, каждая из которых стремится защитить свои ворота и забить гол в чужие.

1.1.3. Математическая модель игры

Основой для математической модели игры «Виртуальный футбол» является упрощенная модель реальной игры роботов цилиндрической формы с механическим приводом на плоском прямоугольном поле с воротами, ограниченным жесткими бортами.

1.1.3.1. Кинематические характеристики объектов

Каждый объект игры в любой ее момент обладает такими кинематическими характеристиками как положение и скорость. Игроки также обладают ускорением, максимальной развиваемой скоростью и максимальной угловой скоростью, учитываемой при совершении поворота. Введение этих ограничений обеспечивает приближение математической модели игры к жизни.

Также игроки и мяч обладают собственной массой, которая влияет на изменение траектории движения в результате столкновения данных объектов.

1.1.3.2. Модель столкновений

Столкновение мяча и игроков обрабатываются как абсолютно упругие столкновения объектов круглой формы [26]. Потерь энергии при столкновении не происходит. Это означает, что суммы энергий объектов в момент до столкновения и в момент сразу после столкновения равны. На рис. 3 изображен типичный случай столкновения игрока и мяча.

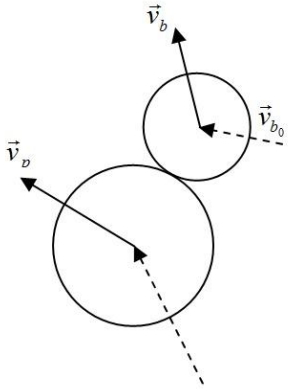


Рис. 3. Соударение мяча и игрока

После соударения мяча или игрока со стенкой скорость движения первого сохраняется, а направление движения изменяется по правилу «угол падения равен углу отражения». Угол падения в данном случае является углом между скоростью движения объекта и плоскостью стенки в момент соударения.

Данный случай иллюстрирует рис. 4. Здесь $\vec{v}_0$ $\vec{v}_0$ – вектор скорости объекта непосредственно до момента столкновения, $\vec{v}$ – новый вектор скорости сразу после.

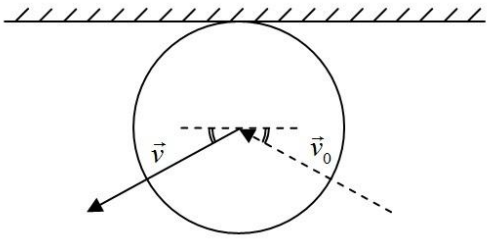


Рис. 4. Соударение робота-футболиста со стенкой

1.1.3.3. Кинематические характеристики роботов

Как было упомянуто ранее, игроки обладают заданным ускорением. Практически это означает, что игрок не может изменить свою скорость в заданный промежуток времени на значение, по абсолютной величине превышающее заданную константу для игры. Также игрок не может изменить угол направления движения на величину большую заданной. Графическая интерпретация данных параметров представлена на рис. 5.

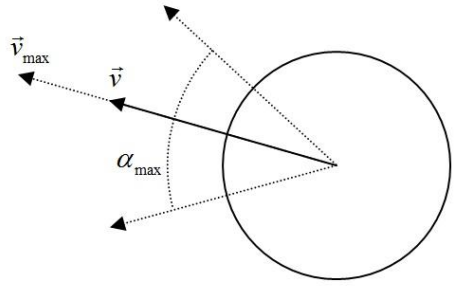


Рис. 5. Кинематические параметры робота

1.1.4. Правила игры

При разработке правил игры в «Виртуальный футбол» был сделан акцент на простоте реализации и максимальной автоматизации проверки соблюдения правил.

1.1.4.1. Начало игры

Игра начинается с разведения мяча в центре поля. Первый игрок команды, защищающей левые ворота, является разводящим в данном случае.

1.1.4.2. Разведение мяча

Мяч помещается в центральную точку поля. Разводящий игрок – представитель команды роботов – находится в случайной точке центрального круга, в его половине, ближайшей к обороняемым воротам. Остальные игроки обеих команд находятся каждый на своей половине поля, не включая зону центрального круга. Положения игроков случайны, а распределение по области, разрешенной для нахождения в момент вбрасывания, является равномерным по ней.

1.1.4.3. Процесс игры

Игрокам разрешены любые столкновения, как с мячом, так и с противником.

1.1.4.4. Блокировка мяча

В процессе игры, вследствие работы алгоритма роботов, может возникнуть такая ситуация, когда мяч будет заблокирован одним или несколькими игроками в углу, или у самого края поля (рис. 6).

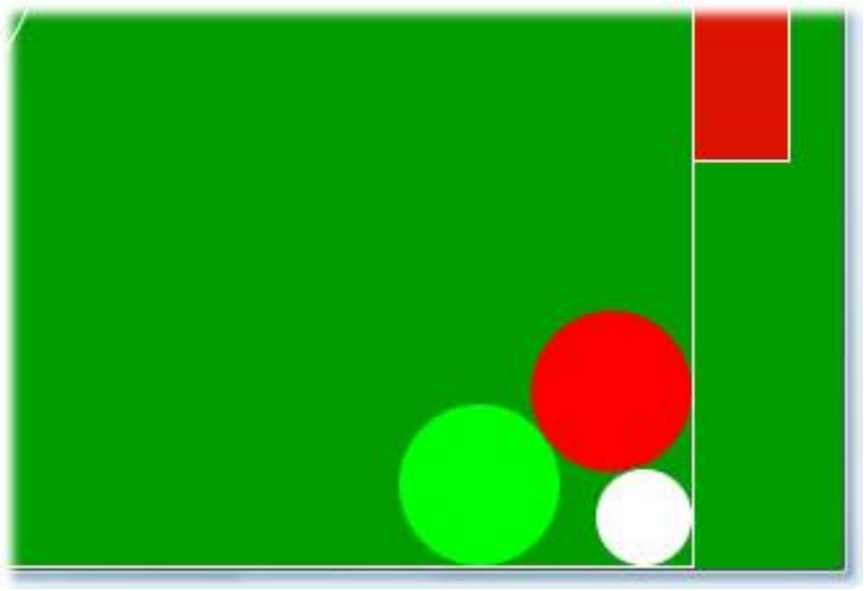


Рис. 6. Блокировка мяча

В данном случае, по истечению указанного времени, отведенного на возможность разблокировать мяч, производится принудительное разведение мяча в середине поля. При этом, разводящая команда не меняется с момента последнего разведения.

1.1.4.5. Условие засчитывания гола

При касании мячом левого или правого края поля в том месте, где расположены ворота, команде, защищающей их, засчитывается пропущенный мяч. Команде-противнику в этом случае прибавляется одно очко к счету. После этого происходит разведение мяча игроком, представляющим команду, пропустившую мяч.

1.1.4.6. Конец игры

Конец игры (рис. 7) наступает по истечении отведенного для нее времени. Этот параметр можно задать в конфигурационном файле приложения.

Рис. 7. Конец игры

1.1.5. Управляющие алгоритмы в игре «Виртуальный футбол»

Каждый алгоритм, управляющий роботом в игре «Виртуальный футбол», является скомпилированной *java*-программой. *Class*-файлы данной программы должны быть помещены в отдельный *jar*-архив, который может быть загружен в игру с помощью диалога формирования команд (рис. 8).

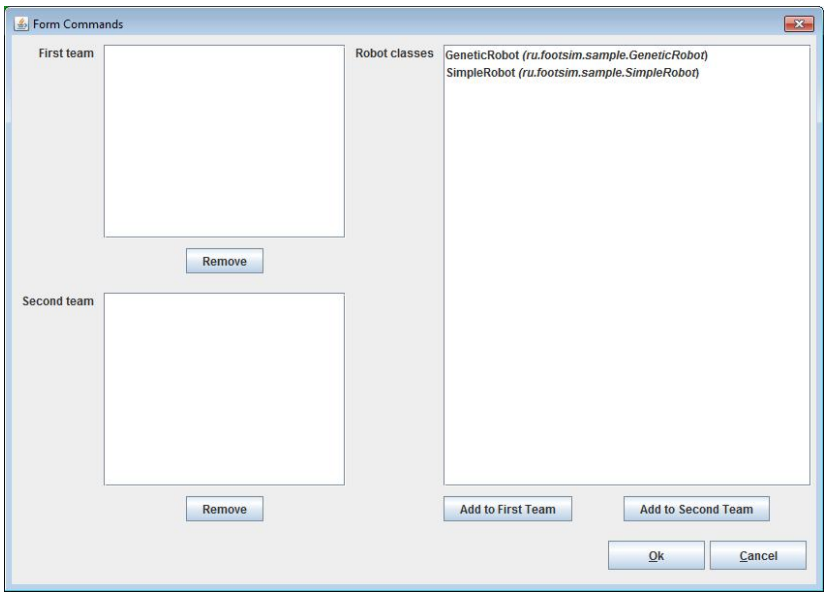


Рис. 8. Диалог формирования команд

1.1.5.1. Класс Robot

Для каждого управляющего алгоритма требуется наличие главного *java*-класса, который должен наследоваться от абстрактного класса *Robot*, предоставляющего основное *API* (API – сокращение от «application programming interface» – программный интерфейс приложения) управления роботом во время игры. Полное имя данного класса должно быть задано в манифесте (манифест – специальный файл с именем *MANIFEST.MF*, находящийся в *jar*-архиве, в каталоге *META-INF*) *jar*-архива под заголовком с именем *Robot-class*.

Класс *Robot* предоставляет следующие функции управления роботом-футболистом:

- void setSpeed(double);
- void setTurnAngleLeft(double);
- void setTurnAngleRight(double);
- void setSpeedVector(Vector).

Первый метод позволяет задать желаемую скорость движения робота. Реальная скорость движения может отличаться после вызова этого метода из-за наличия ускорения и ограничения скорости движения робота.

Второй и третий методы позволяют изменить угол направления движения, совершая, соответственно, поворот налево или направо. Данные методы учитывают ограничения, накладываемые на изменение угла, а именно максимальную угловую скорость. Если аргумент метода превосходит эту величину, то происходит изменение угла на максимально разрешенное значение.

Последний метод объединяет в себе функциональность первых двух и обеспечивает простоту задания скорости игрока. Данный метод все так же учитывает ограничения, накладываемые на скорость и угол поворота объекта, описанные выше.

Данный набор методов позволяет полностью контролировать поведение робота в рамках игры «Виртуальный футбол».

1.1.5.2. Класс StatusEvent

Каждый временной тик игры (наименьший временной отрезок игры) для всех роботов вызывается метод *void onStatus(StatusEvent)*. Данный метод является абстрактным методом класса

Robot и требует наличия реализации в каждом управляющем алгоритме. Переопределение данного метода создателем робота задает алгоритм управления роботом в процессе игры. Его единственный аргумент содержит в себе всю необходимую информацию об окружающей среде на момент вызова данного метода. Класс *StatusEvent* предоставляет пользователю следующий набор функций:

- *BallInfo getBallInfo();*
- *FieldInfo getFieldInfo();*
- *GameStatistics getGameStatistics();*
- *PlayerInfo getMyInfo();*
- *List<PlayerInfo> getOthersInfo().*

С помощью них управляющий алгоритм может получить отчет об игровой ситуации, включающий в себя положения и скорости всех игроков и мяча, их линейные размеры, текущий счет и конфигурацию игрового поля. Игровое поле характеризуется линейными размерами, шириной ворот и радиусом центрального круга.

UML диаграмма [2] соответствующих классов, необходимых при создании управляющих алгоритмов в игре, приведена на рис. 9.

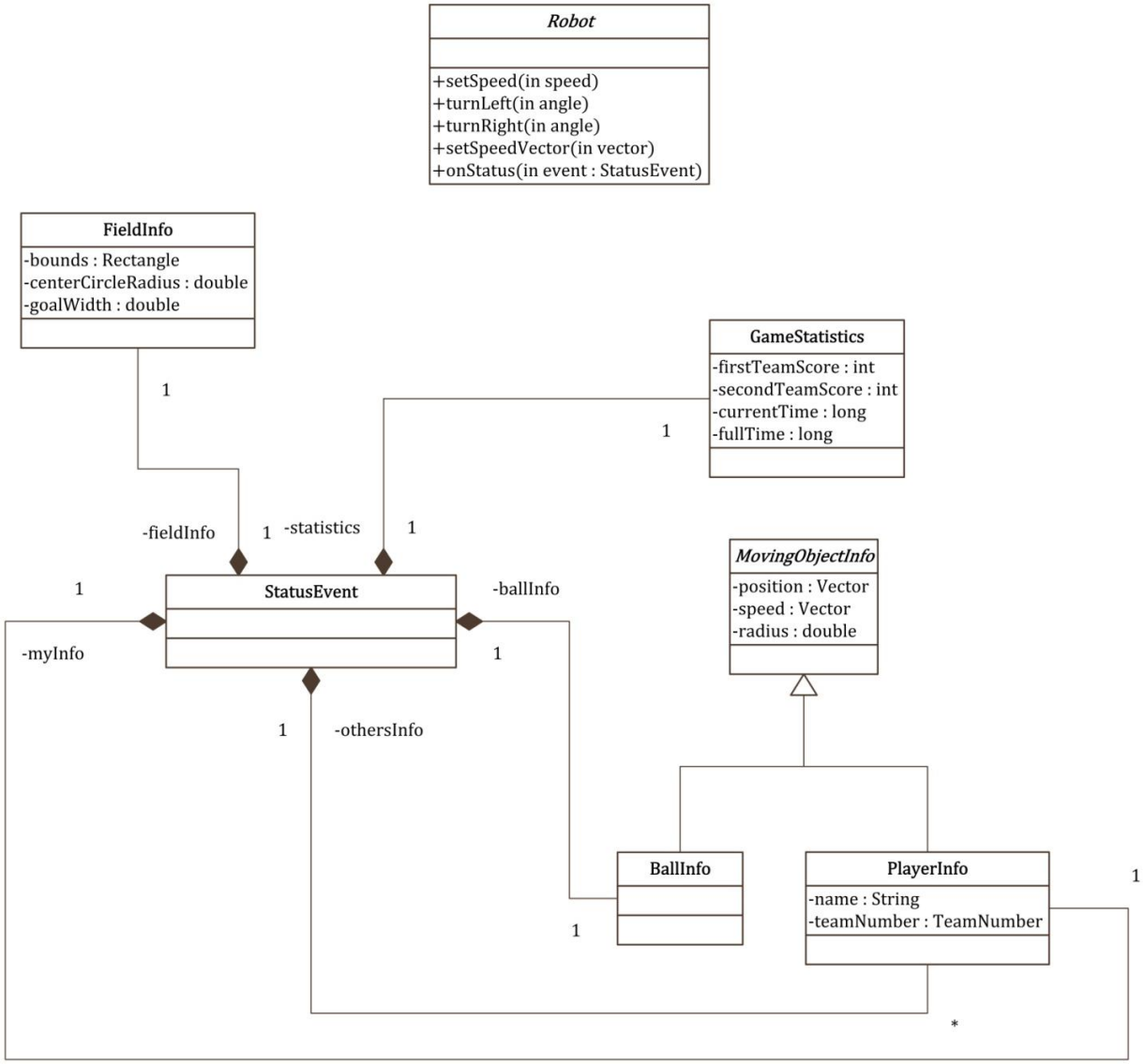


Рис. 9. Диаграмма классов

1.1.6. Математическая модель робота-футболиста

1.1.6.1. Автоматное представление

Для управления роботом-футболистом – объектом управления игры «Виртуальный футбол» – был использован конечный автомат Мили (рис. 10). На вход автомату подаются входные последовательности сигналов, на основе которых, а так же своего внутреннего состояния, он формирует выходные последовательности сигналов [27].

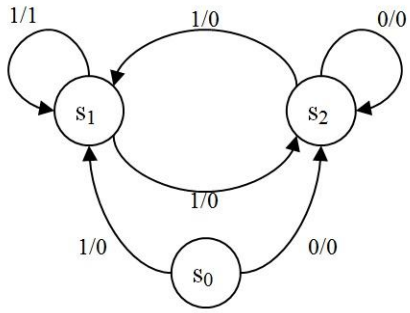


Рис. 10. Пример диаграммы переходов автомата Мили из трех состояний

Входные воздействия представляют собой закодированное в битовую строку некоторое приближение игровой ситуации. В случае «Виртуального футбола», игровая ситуация представляет собой совокупность вещественных переменных, задающих положение самого робота, мяча, соперника, а так же их скоростей.

1.1.6.2. Использование деревьев решений

Каждое состояние управляющего автомата представляет собой дерево решений [7], в листьях которого хранятся переходы автомата. Переход автомата представляет собой совокупность значения функции переходов T и значения функции выходного воздействия G.

1.1.6.3. Входные и выходные воздействия

Выбор представления входных и выходных воздействий для управляющего автомата в генетическом алгоритме сильно влияет на оптимальность получаемого решения. В результате эвристических наблюдений и проведенных экспериментов было найдено представление входных воздействий, которое позволило добиться хороших результатов за приемлемое время. В него входит набор следующих переменных:

- Скорость игрока относительно положения мяча (рис. 11), кодируемая двумя переменными. Значение первой из них определяется исходя из попадания вектора скорости игрока $\vec{v}_p$ в одну из двух областей плоскости (зоны I–II и III–IV), разделенных прямой, проходящей через две точки: центр игрока и центр мяча. Значение второй – в зависимости от направления движения игрока в сторону мяча, либо от него (соответственно зоны I–IV и II–III).

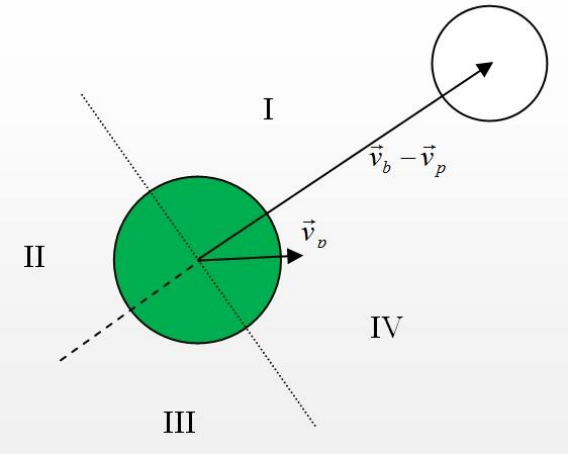


Рис. 11. Скорость игрока относительно положения мяча

- Скорость игрока относительно положения соперника. Задается двумя булевыми переменными аналогично предыдущему пункту (вместо положения мяча берется положение игрока-оппонента).
- Положение мяча относительно робота-футболиста (рис. 12).

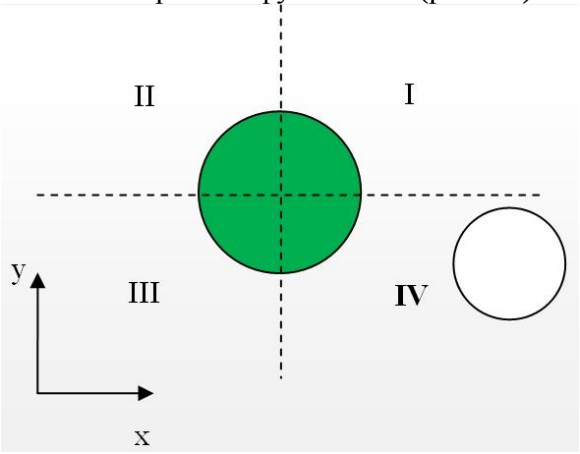


Рис. 12. Положение мяча относительно положения игрока

- Рассматриваются четыре квадранта плоскости, в качестве осей координат которой выступают прямые, параллельные сторонам поля, и проходящие через центр круга, представляющего робота. Номер квадранта, в котором расположен центр мяча, кодируется двумя булевыми переменными.
- Направление скорости игрока относительно положения центра ворот противника (рис. 13).

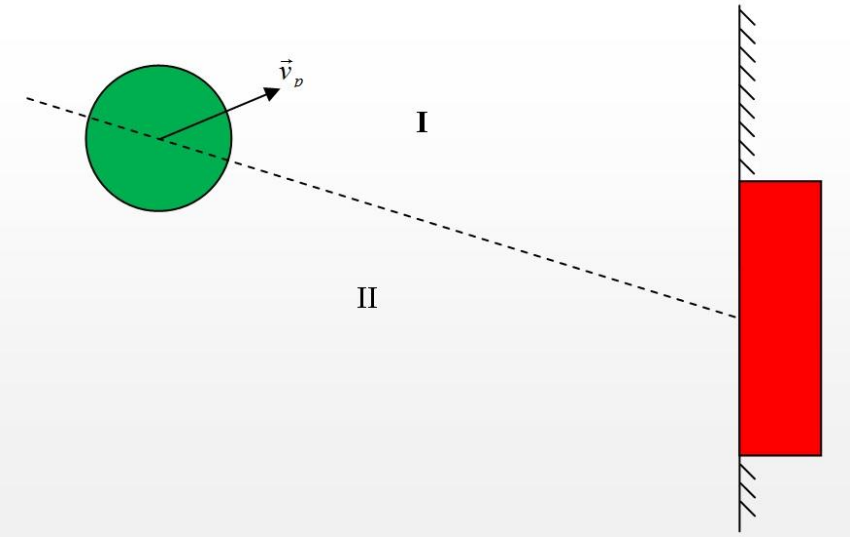


Рис. 13. Направление скорости игрока на ворота

- Направление скорости игрока относительно положения центра ворот самого игрока. Значение переменной определяется по аналогии с предыдущим пунктом.

Выходные воздействия автоматов представлены двумя типами: повернуть направо на максимально возможный угол за один тик, либо повернуть налево на то же значение. При этом предполагается, что в каждый момент игры виртуальный робот-футболист движется с максимально возможной скоростью.

1.1.7. Программная реализация

Для реализации метода генетического программирования был создан ряд классов, представляющих собой отдельную библиотеку, включающую в себя реализацию конечных автоматов Мили, деревьев решений и интерфейсов для представления используемых входных переменных.

1.1.7.1. Абстрактные деревья решений

IDecisionTree – интерфейс, описывающий обобщенное дерево решений [15]. Его реализация предоставляет пользователю набор эффективных и удобных методов для добавления вершин и меток на них, изменение информации на вершинах и ребрах, а также обхода дерева решений. В него входят два интерфейса: *IEdge*, соответствующий ребру дерева решений и *INode* – узел этого дерева.

DecisionTree – реализация дерева решений, инкапсулирующая информацию обо всех узлах и ребрах дерева и предоставляющая доступ к ней через интерфейс *IDecisionTree*.

В отдельный утилитный класс *DecisionTreeUtils* вынесены методы работы с деревьями решений. Перечислим только основные из них:

- метод *swapSubtrees* – принимает в качестве аргументов две узловые вершины различных деревьев и осуществляет замену соответствующих поддеревьев друг на друга;
- метод *cloneTree* – копирует дерево, указанное в аргументе и возвращает его пользователю;
- метод *trim* – позволяет производить обрезку дерева по заданной максимальной высоте.

1.1.7.2. Представление входных переменных

Для организации работы генетического алгоритма необходимо, в частности, задать набор входных переменных, а так же в его процессе определять значения данных переменных по вычислительному состоянию окружающей среды. В нашем случае вычислительным состоянием

является текущая игровая ситуация. Для гибкого задания переменных с последующим вычислением их значений в зависимости от имеющейся информации об окружающей среде был разработан следующий набор интерфейсов и классов:

- *IType* – интерфейс, описывающий тип переменной, которая может принимать дискретный набор значений. Он предоставляет информацию о мощности множества значений переменной. Так же с его помощью можно получить конкретное значение переменной по ее порядковому номеру.
- *IValue* – интерфейс, реализация которого представляет собой конкретное значение переменной.
- *IState* – сущность, инкапсулирующая информацию о полном наборе значений входных переменных, с помощью которой можно получить значение переменной по ее типу.
- *StateImpl* – эффективная реализация интерфейса *IState*, предполагающая использование реализации интерфейса *IType* основанной на *Java* перечислениях.

1.1.7.3. Интерфейс автомата Мили

Автомат представляет собой реализацию интерфейса *IAutomaton<I,O>*, вся функциональность которого скрыта в одном методе *nextStep*, получающем на вход сигнал generic-типа *I* и возвращающий выходной сигнал типа *O*. Соответствующий данному интерфейсу программный код, имеет следующий вид:

```
package ru.footsim.automaton;

public interface IAutomaton<I, O> {
    O nextStep(I input);
}
```

1.1.7.4. Реализация деревьев решений

Дерево принятия решений в данной работе используется в качестве структуры данных, задающей для каждого состояния функцию переходов и функцию действий в конечном автомате Мили. Класс *StateTransitionTree*, наследуемый от *DecisionTree*, реализует данную функциональность. В своих узлах он хранит либо информацию о переменной, которой помечен узел, и тогда он является внутренним узлом, либо, если это лист, пару ⟨Новое состояние, Действие⟩. Действие в нашем случае описывается перечислением *ControlSignal*, которое включает в себя две константы: *TurnLeft* и *TurnRight*. Они означают, что робот должен повернуть, соответственно, либо вправо, либо влево на максимально разрешенный угол за один тик.

1.1.7.5. Реализация входных переменных

Полный набор входных переменных задается в классе *FootsimType*, являющимся перечислением *Java*, и имплементирующим интерфейс *IType*. Гибкость этого класса обеспечивается наличием интерфейса *IVariableService*, инкапсулирующего информацию о конкретном типе переменной. Это позволяет сделать реализации типов переменных независимыми друг от друга. Для задания новой переменной необходимо унаследоваться от интерфейса *IVariableService*, реализовав методы *getValuesCount(), getValue(int), getValue(StatusEvent)*, а затем добавить новую константу в класс-перечисление *FootsimType*, указав в качестве параметра конструктора экземпляр соответствующего класса.

1.1.7.6. Автомат Мили и генетические операции

Класс *Automaton* представляет собой реализацию автомата Мили, для каждого из состояний содержащего дерево решений, описанное в предыдущем разделе. Для работы генетического алгоритма необходимы реализации генетических операций на автоматах. Данная функциональность находится в *singleton*-классе *AutomatonService*, содержащем следующие методы:

- *generateRandomAutomaton* – создание случайного автомата;
- *crossAutomatons* – операция скрещивания двух заданных автоматов;
- *mutateTree* – операция мутации автомата.

1.1.7.7. Управляющий конечный автомат робота

Генерируемый робот-футболист является объектом класса *AutomataBasedRobot*. Базовым для него, как и для любого управляющего алгоритма, является класс *Robot*. В его методе *onStatus* выполняется преобразование информации о вычислительном состоянии в набор значений переменных. Оно осуществляется посредством вызова метода *getState* специализированного класса *FootsimVariableService*, передавая в качестве единственного аргумента объект *StatusEvent*, имеющийся в распоряжении пользователя. Полученный набор значений, инкапсулированный в объект класса *FootsimState*, передается на вход управляющему автомату робота-футболиста, в зависимости от которого автомат переходит в новое состояние и генерирует одно из возможных выходных действий – *TurnLeft* или *TurnRight*. В соответствии с этим роботу подается сигнал либо повернуть налево, либо повернуть направо на максимально возможный угол. При этом робот всегда движется с максимально возможной скоростью.

1.1.8. Генетический алгоритм

Для представления автоматов в виде деревьев решений в эволюционных алгоритмах [36, 41] необходимо задать генетические операции [4]. В дальнейшем будем считать, что число состояний в автоматах фиксировано. В данной работе генетические операции для метода генетического программирования реализовывались следующим образом:

Случайная генерация автомата, выполняемая на шаге инициализации – в каждом состоянии создается случайное дерево решений.

Скрещивание автоматов – скрещиваются деревья решений в соответствующих состояниях.

Мутация автомата – в случайном дереве решений выполняется мутация.

Теперь рассмотрим операции над собственно самими деревьями решений:

Случайное порождение дерева решений – случайным образом выбирается метка: либо одно из возможных значений функции переходов, либо одна из переменных. После этого создается вершина, помеченная выбранной меткой. При этом если была выбрана переменная, то рекурсивно генерируются дети вершины, иначе вершина становится листом дерева.

Мутация – выбирается случайный узел в поддереве. После этого поддерево, соответствующее выбранному узлу, заменяется на случайно сгенерированное (рис. 14). Процесс генерации случайного поддерева аналогичен описанному порождению дерева решений в предыдущем пункте.

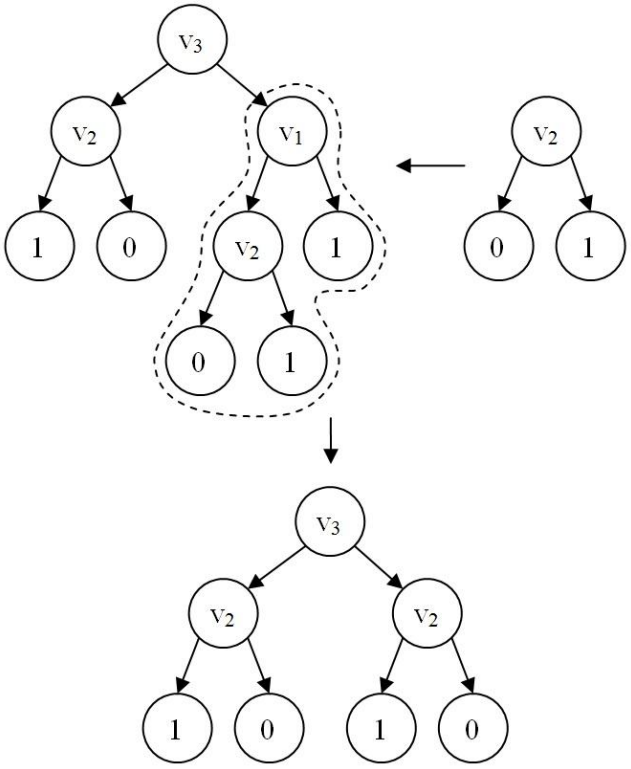


Рис. 14. Мутация дерева решения

Скрещивание – в каждом из скрещиваемых деревьев случайно выбирается по одному узлу. После этого поддеревья, соответствующие выбранным узлам в первом и втором дереве, заменяются друг на друга (рис. 15).

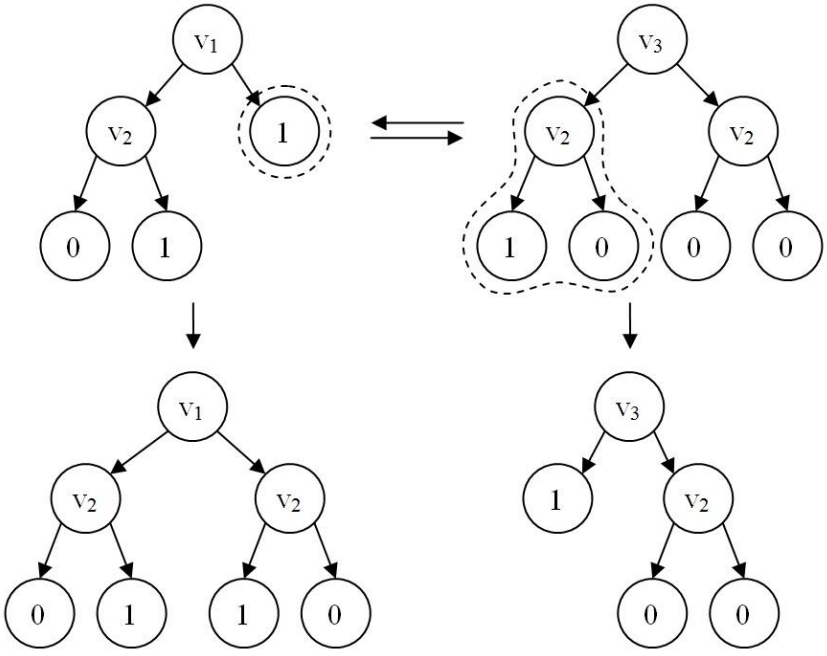


Рис. 15. Скрещивание деревьев решений

Следует заметить, что заданные таким образом операции мутации и скрещивания могут порождать деревья, в которых некоторый атрибут встречается на пути от корня до листа дважды. Такие деревья являются корректными, так как определяют функцию на любом наборе значений атрибутов единственным образом, но в то же время имеют недостижимые вершины. Если атрибут встречается на пути от корня до листа дважды, то значение его уже известно, а значит ветви, соответствующие другим значениям переменной, недостижимы ни при каких значениях остальных переменных. Очевидно отрицательное влияние недостижимых ветвей на работу генетического алгоритма. Действительно, появление данных ветвей ведет к увеличению высоты деревьев, что означает экспоненциальное увеличение необходимой памяти на хранение популяции. При этом недостижимые ветви не несут полезной информации, так как не учитываются при определении значения задаваемой деревом решения функции.

Пример дерева решений, содержащего недостижимую вершину, приведен на рис. 16. Функция, задаваемая этим деревом, в точности эквивалентна функции, описываемой деревом решений на рис. 10, но содержит две избыточные вершины: внутреннюю вершину, помеченную переменной v_3, и лист со значением 1.

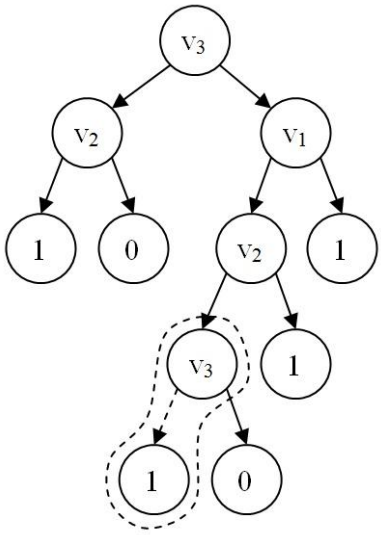


Рис. 16. Пример дерева решений с недостижимыми ветвями

Таким образом, в процесс применения генетических операций мутации и скрещивания необходимо ввести коррективы по выявлению недостижимых ветвей и их обрезке. Этого можно достичь, запоминая переменные и их значения, которыми помечены узлы и ребра на пути от корня до вершины изменяемого поддерева, а затем используя эту информацию в генетических операциях. В операции мутации множество доступных переменных для генерации поддерева будет сужено до переменных, не входящих в составленный набор. При скрещивании, в процессе замены поддерева необходимо добавлять только те внутренние узлы, которые помечены еще не использованными переменными. Если же такой узел встретился, то его необходимо заменить на того из его потомков, ребро к которому помечено соответствующим значением переменной.

1.1.9. Алгоритм имитации отжига

Алгоритм имитации отжига [10] основан на моделировании физического процесса, который происходит при кристаллизации вещества из жидкого состояния в твердое (например, при отжиге металла). В этой модели предполагается, что, во-первых, отжиг происходит при понижении

температуры, а во-вторых, атомы в веществе уже выстроились в кристаллическую решётку, но отдельные атомы ещё могут перейти из одной ячейки в другую. Вероятность таких переходов в свою очередь зависит от температуры. Устойчивая кристаллическая структура вещества соответствует минимуму энергии.

Формализуем процесс отжига. Фактически, при моделировании ищется точка (или набор точек), на которой достигается минимум некоторой числовой функции $E(\vec{x})$. Строится последовательность точек $\vec{x}_0, \vec{x}_1, ..., \vec{x}_n$, где $\vec{x}_0$ соответствует начальному значению. При достижении точки $\vec{x}_n$ алгоритм завершает свою работу. Пусть рассматривается текущая точка $\vec{x}_i$. К ней применяется некоторый оператор A_i, который зависит от температуры. После модификации получаем точку $\vec{x}'$. Вероятность выбора $\vec{x}'$ в качестве следующей точки $\vec{x}_{i+1}$ равна $P(\vec{x}', \vec{x}_i)$, где P – распределение Гиббса:

$$P(\vec{x}', \vec{x}_i) = \begin{cases} 1 & E(\vec{x}') - E(\vec{x}_i) < 0 \\ \exp\left(-\dfrac{E(\vec{x}') - E(\vec{x}_i)}{Q_i} \right) & E(\vec{x}') - E(\vec{x}_i) \geq 0 \end{cases}, \quad (1)$$

где $Q_i > 0$ – элементы произвольной убывающей сходящейся к нулю последовательности. Эта последовательность является аналогом понижающейся температуры в реальном физическом процессе.

Весь алгоритм можно разбить на шесть этапов (рис. 17):

- создание начального решения;
- оценка решения;
- изменение решения случайным образом;
- оценка нового решения;
- выбор нового решения в качестве рабочего, если оно прошло критерий допуска;
- уменьшение температуры.

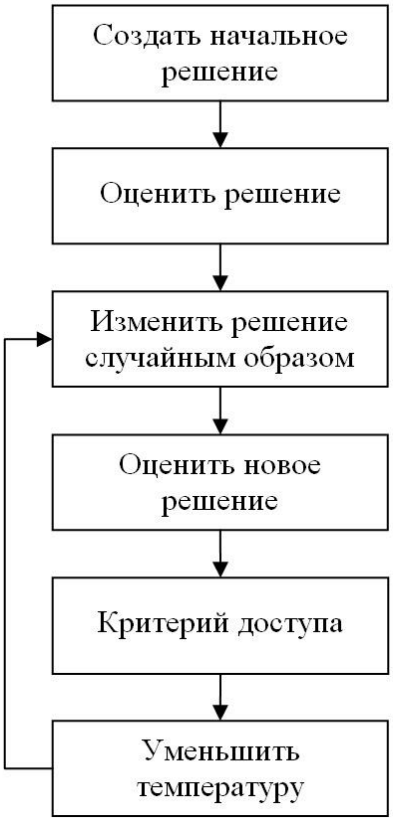


Рис. 17. Схема алгоритма имитации отжига

Рассмотрим подробнее каждый из этапов. Будем считать, что решение представляет собой битовую строку фиксированной длины. При этом любая битовая строка этой длины является решением.

Создание начального решения. На этом этапе выбирается некоторая начальная точка $\vec{x}_0$, которую принимают в качестве рабочего решения. Этой точкой может быть уже существующее решение (например, полученное другими алгоритмами) или сгенерированное случайным образом решение. В данной работе начальное решение будет генерироваться случайным образом.

Оценка начального решения. Данный этап нужен для того, чтобы в дальнейшем сравнивать новые решения с рабочим. В простейшем случае можно использовать функцию $E(\vec{x})$ для оценки решения.

Построение нового решения. На этом этапе рабочее решение изменяется случайным образом. Для этого задаётся порождающая функция $\gamma(T,\vec{x})$. В данной работе каждый бит рабочего решения может меняться с вероятностью $\frac{T}{T_{\max}}$.

Оценка нового решения. После построения решения его следует оценить, чтобы далее сравнить с рабочим. Для этих целей используется та же функция, что и во втором пункте.

Выбор нового рабочего решения. На предыдущем этапе была получена оценка нового решения $E(\vec{x}')$. По формуле (1) находим вероятность принятия нового решения в качестве рабочего. Таким образом, в качестве $\vec{x}_{i+1}$ выбирается $\vec{x}'$ или $\vec{x}_i$.

Уменьшение температуры. Перед тем, как перейти к новому шагу алгоритма, нужно уменьшить температуру по некоторому закону $T(k)$, где k – номер шага. В данной работе функция $T(k)$ задана рекурсивно:

$$T(k) = \alpha T(k-1), \quad (2)$$

где $\alpha = \sqrt[N]{\frac{T_{\min}}{T_{\max}}}$, $T(0) = T_{\max}$, N – число итераций.

1.1.10. Комбинированный метод

В рамках предложенного комбинированного метода для работы генетического алгоритма предлагается использовать генетические операции мутации и скрещивания, описанные выше, а в качестве начального приближения использовать автоматы, полученные в результате работы алгоритма имитации отжига. Применение данного подхода может способствовать увеличению шансов генетического алгоритма на достижение оптимального решения, минимизируя при этом затраты времени на свою работу.

1.1.11. Результаты экспериментов

Роботы, построенные с использованием предложенного подхода, сравнивались с автоматами, сгенерированными с помощью алгоритма имитации отжига, и автоматами, полученными в результате работы метода генетического программирования.

Запуск генетического алгоритма проводился со следующей конфигурацией:

- размер популяции – 128 особей;
- стратегия отбора – элитизм, для размножения считаются пригодными 25% особей популяции (32 особи), имеющих максимальное значение фитнес-функции;
- частота мутации – 5%.

Значение функции приспособленности автомата определялось по результатам его игры с роботом-эталоном. Для ее вычисления использовалась формула $f = \frac{testRobot.points + etalonRobot.points}{etalonRobot.points}$ – отношение суммарного количества очков, набранных обоими роботами к количеству очков, набранных эталонным роботом. Пример графика зависимости соответствующей функции от порядкового номера популяции в ходе работы генетического алгоритма приведен на рис. 18.

Рис. 18. График фитнес-функции, используемой в настоящей работе

Очки роботам начислялись на каждом тике игры по следующей формуле: $p(x,y) = x + \frac{1}{4}|y| - \frac{3}{4}x|y|$, где (x,y) – координаты мяча, причем $x \in [0,1]$, $y \in [-1,1]$. Максимальное значение $p(1,0) = 1$ данная функция принимает в центре ворот противника, а минимальное $p(0,0) = 0$ – в центре ворот игрока. Графическая интерпретация функции $p(x,y)$ представлена на рис. 19.

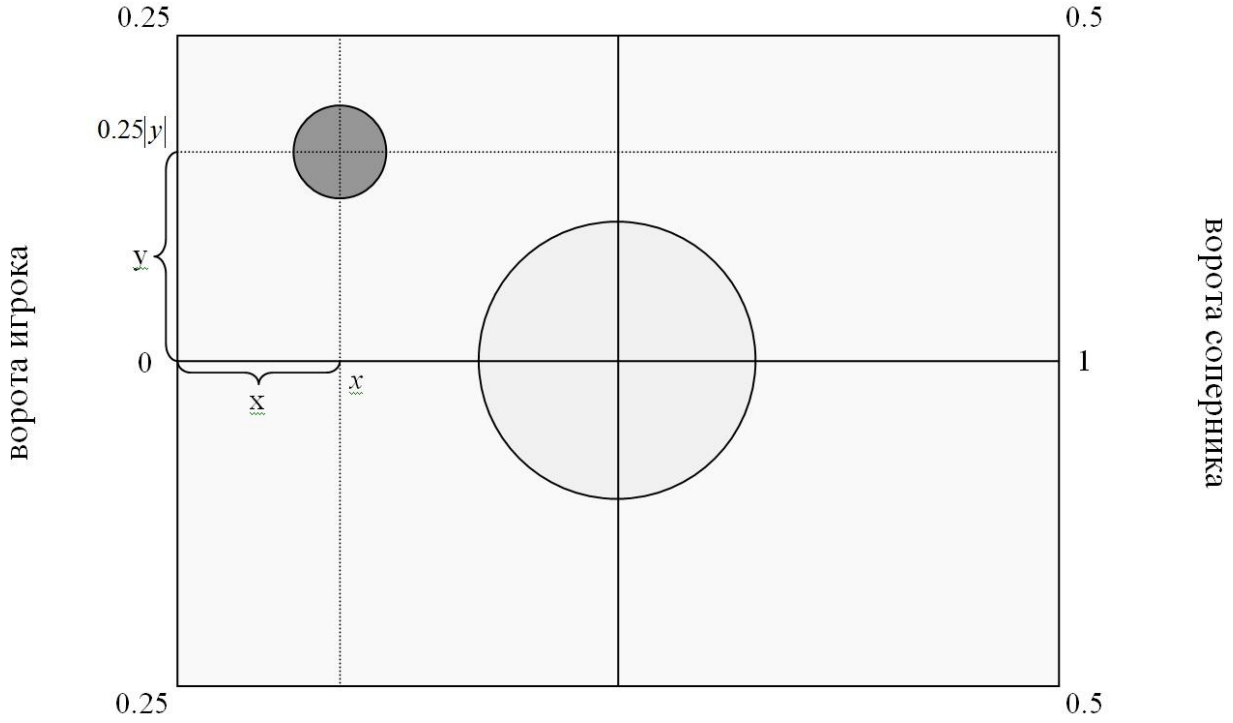


Рис. 19. Графическая интерпретация функции $p(x, y)$

Выбор данной фитнес-функции вместо функций, основанных на итоговом счете в конце каждой игры, объясняется тем, что последние имеют значительно больший разброс значений. Данный факт не может не сказаться негативным образом на работе метода генетического программирования. В то же время, используемая функция приспособленности, как показывают результаты проведенных экспериментов, довольно точно отвечает конечной цели виртуального робота-футболиста.

График функции в ходе работы генетического алгоритма приведен на рис. 20.

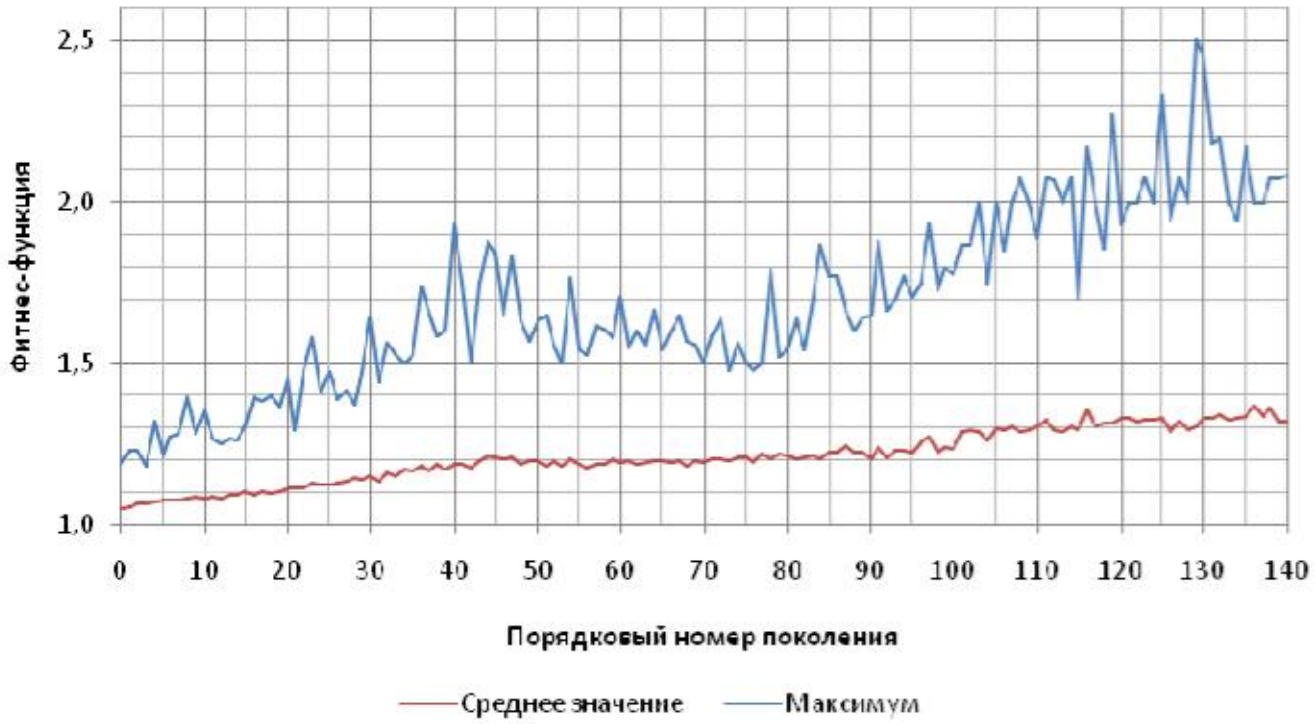


Рис. 20. График фитнес-функции, основанной на итоговом счете игры

В сериях экспериментов входные воздействия автомата были представлены восемью переменными:

- четыре переменные, определяемые направлением скорости игрока относительно положения следующих объектов: мяча, оппонента, центра своих ворот, центра ворот соперника;
- двумя переменными, определяемыми положением мяча относительно положения игрока;
- двумя переменными, задаваемыми направлением движения игрока, относительно положения мяча и положения противника.

Подробное объяснение значения каждой переменной приведено в разделе 1.1.6.3.

Количество состояний генерируемых автоматов равно 6.

В рамках каждого эксперимента время генерации двухсот поколений автоматов по 128 особей в каждом занимало порядка 8 часов. Для проведения экспериментов использовался компьютер с процессором *Intel Core 2 Duo 2.00 GHz*.

По результатам проведения 100 экспериментов по оценке эффективности работы генетического алгоритма был построен график функции распределения случайной величины – номера поколения, при котором достигается максимум значения фитнес-функции (рис. 21).

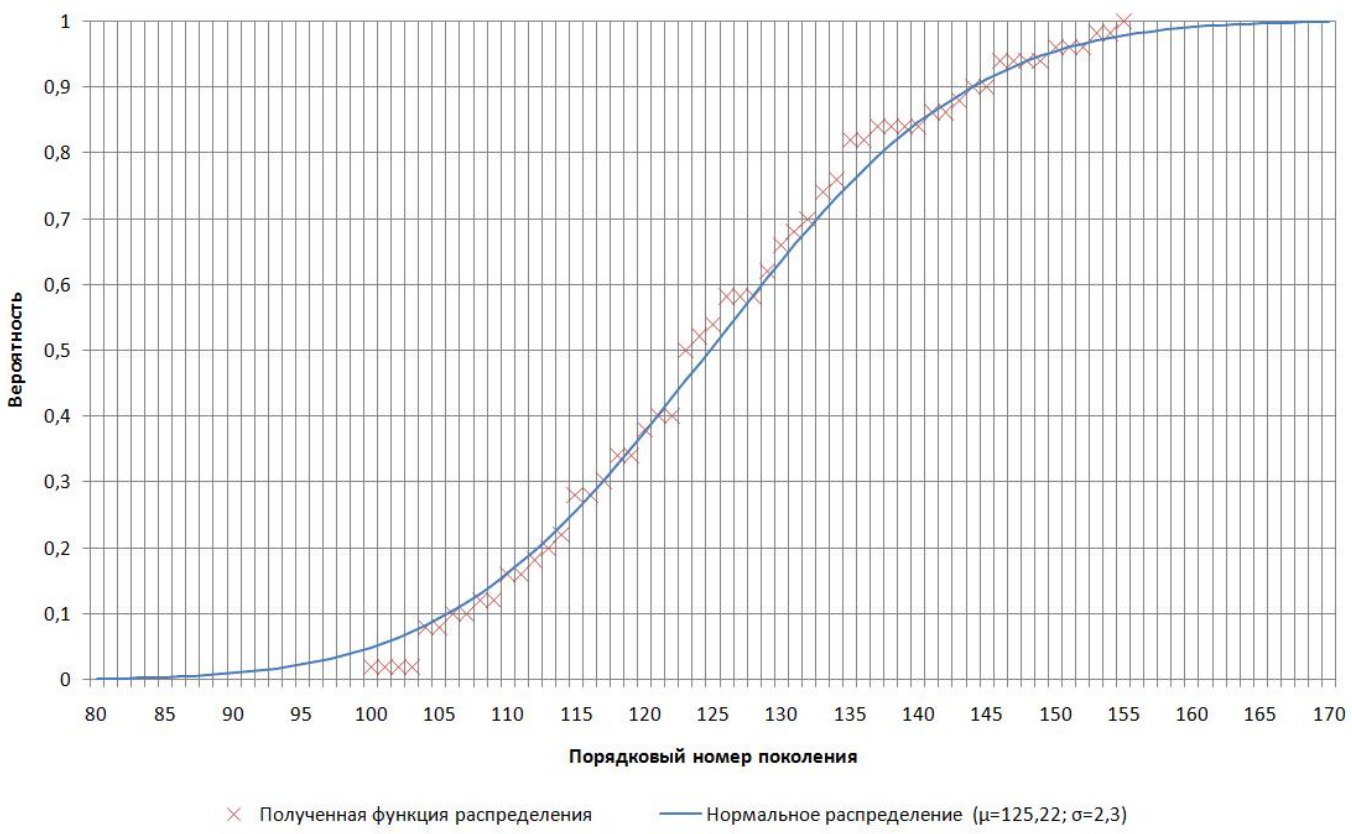


Рис. 21. График функции распределения для генетического алгоритма

С аналогичным набором входных и выходных воздействий, а также конфигурацией производимых автоматов запускался также алгоритм имитации отжига. Каждый эксперимент включал в себя генерацию серии конечных автоматов с использованием различных случайных начальных приближений, и последующий выбор наиболее оптимального автомата из набора полученных. Число автоматов в каждой серии было равно 128. Число итераций понижения температуры в ходе работы алгоритма – 200. При этом начальная температура – 1, конечная – 10^{-4}. В качестве функции энергии $E(\vec{x})$ была использована фитнес-функция, задействованная в экспериментах с генетическим алгоритмом, взятая с обратным знаком. График функции распределения для данной серии экспериментов представлен на рис. 22. График зависимости значения энергии рабочего решения от порядкового номера поколения в ходе работы алгоритма для некоторого произвольного начального приближения приведен на рис. 23.

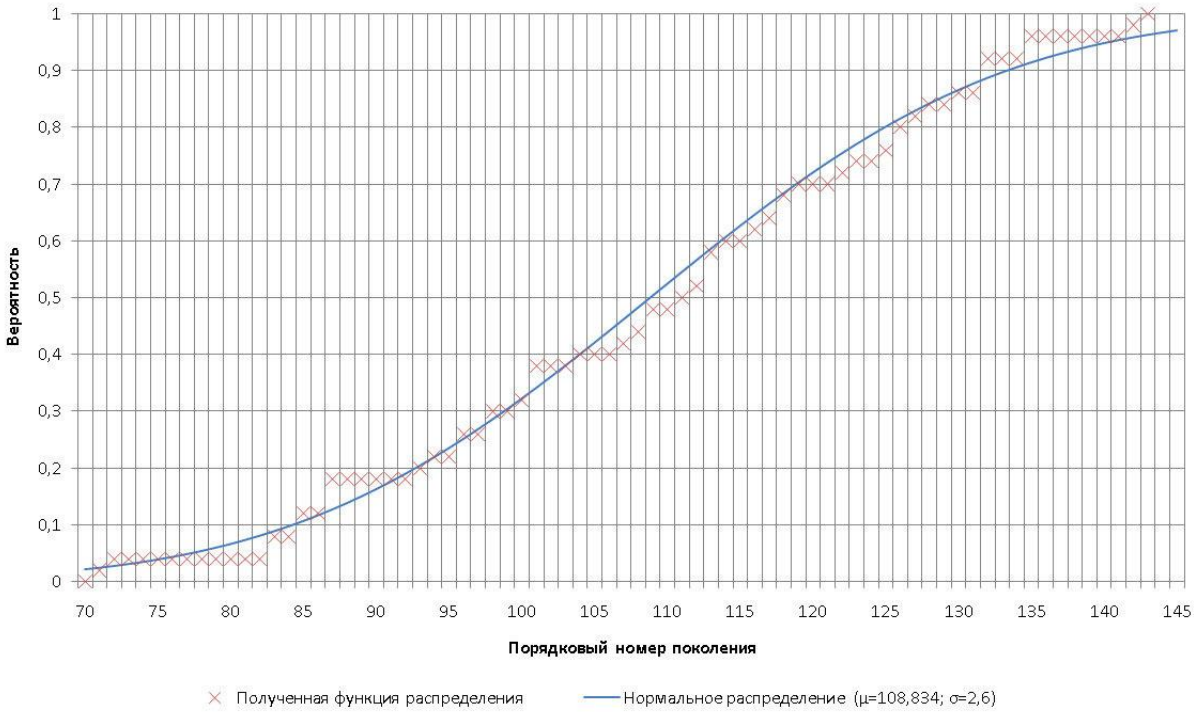


Рис. 22. График функции распределения для алгоритма отжига

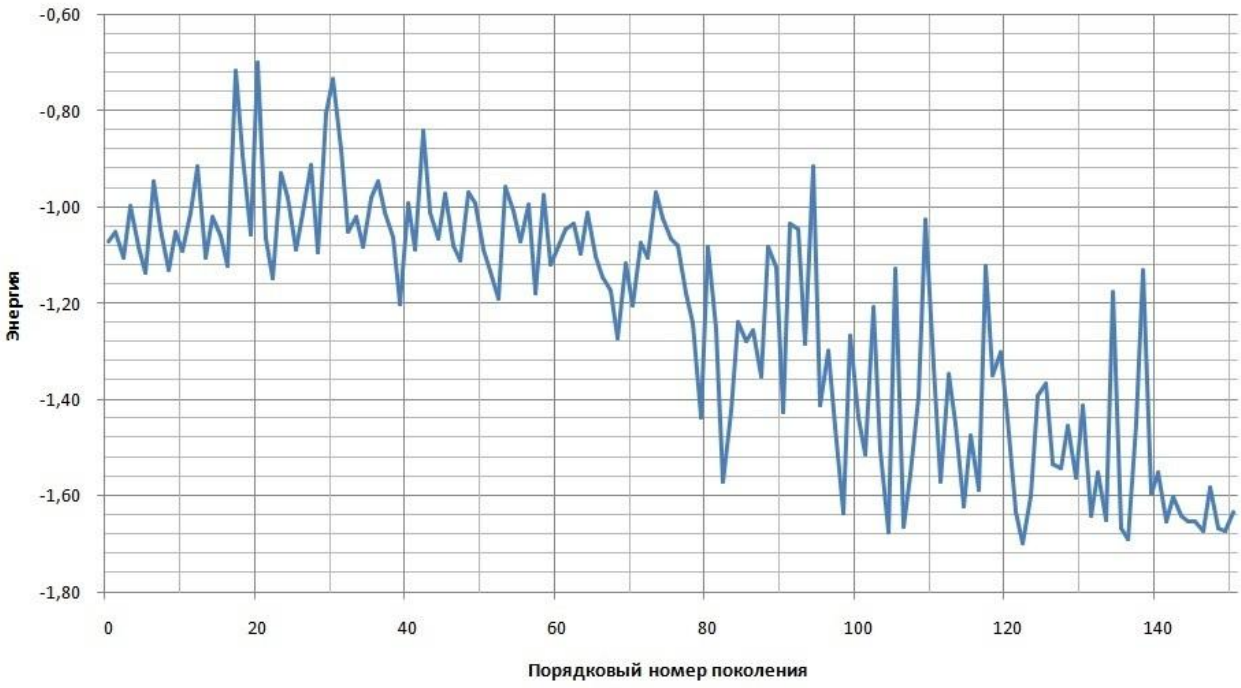


Рис. 23. График зависимости энергии решения от номера поколения

Последним из применяемых в данной работе методов генерации управляющих автоматов стал комбинированный метод. Первым шагом генерации было создание начальной популяции, состоящей из 128 управляющих автоматов, получаемых в результате применения алгоритма имитации отжига.

Число итераций понижения температуры в ходе работы алгоритма – 50. Второй шаг заключался в применении метода генетического программирования, использующего полученное начальное приближение. Функция распределения номера поколения, при котором достигается лучшее решение, с использованием комбинированного метода, представлена на рис. 24.

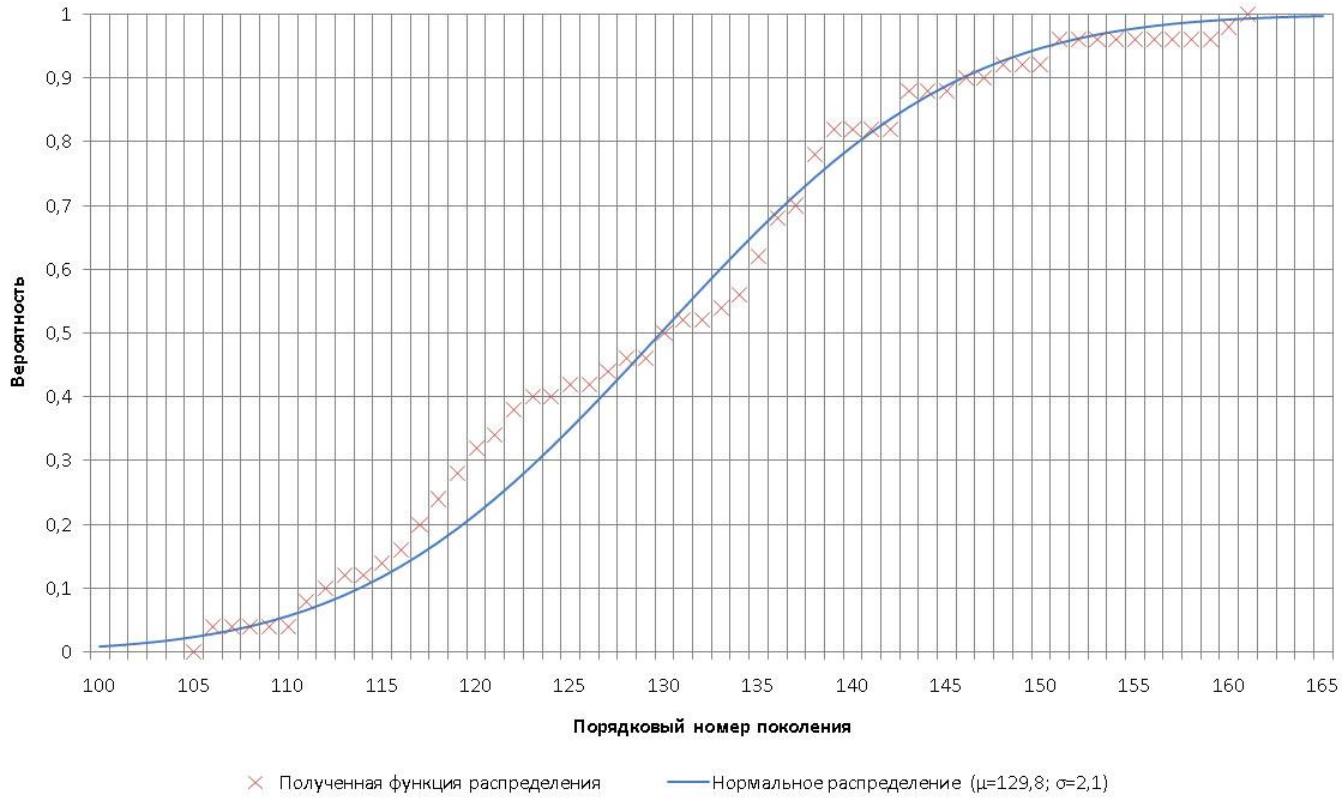


Рис. 24. График функции распределения для комбинированного метода

Заключительным этапом проведенных экспериментов стало сравнение автоматов, полученных в результате применения различных методов. Для каждой серии экспериментов был выбран автомат, обладающий наибольшим значением фитнес функции. Затем отобранные автоматы попарно приняли участие в играх друг против друга. Отношения голов, забитых автоматами в ходе проведения данных игр, представлены в таблице 1.

Таблица 1. Сравнение автоматов, полученных различными методами

Метод	ИО	ГА	КМ
Алгоритм имитации отжига (ИО)	x	0,82	0,83
Генетический алгоритм (ГА)	1,21	x	1,001
Комбинированный метод (КМ)	1,2	0,999	x

Из приведенных результатов видно, что автоматы, полученные генетическим и комбинированным методами, превосходят по эффективности автомат, созданный алгоритмом имитации отжига, в играх с которым они забивают на 9-10% больше голов, чем последний в ответ. В то же время, автоматы, полученные генетическим и комбинированным методами, обладают сходными характеристиками эффективности, каждый забивая друг другу число голов, отличающееся менее 0,1% от общего числа голов, что можно объяснить наличием статистической погрешности при проведении данного эксперимента.

1.1.12. Выводы на основе сравнения

Анализ полученных результатов игр показывает, что метод генетического программирования работает эффективнее алгоритма имитации отжига. Последний быстро достигает локального максимума, после чего найти лучшее решение не удается. Как показывают проведенные эксперименты, запуск алгоритма имитации отжига несколько раз с различными начальными решениями также не приносит значимых улучшений.

Применение комбинированного метода было нацелено на улучшение эффективности работы генетического алгоритма. Однако апробация его в рамках данной задачи не принесла желаемого результата. Как видно из результатов проведенных экспериментов, начальное поколение, созданное с помощью алгоритма имитации отжига, не вносит в результаты работы генетического алгоритма сколь либо существенных изменений, достигая аналогичных показателей эффективности рабочего решения в среднем за равное количество вычислений фитнес-функции в работе алгоритмов.

1.2. Разработка, программная реализация и экспериментальное исследование метода применения коэволюции для построения взаимодействующих управляющих конечных автоматов

В природе особи разных видов по теории Дарвина эволюционируют, лучшие выживают и оставляют потомство. Эти же принципы заложены в основу генетических алгоритмов – решение задачи выращивается на основе многих поколений эволюции. В природе особи доказывают свое превосходство при взаимодействии с другими, даже растения захватывают с помощью потомства наиболее благоприятные территории. Совместное развитие особей, при котором они друг на друга влияют, называется коэволюцией. В математике подобные отношения исследует теория игр. В работе предлагается решить математическую проблему с помощью коэволюции, подсказанной природой.

Теория игр изучает математические методы поиска оптимальных стратегий в играх. Во многих играх задействовано несколько участников, у каждого из них есть множество стратегий. При этом в игре часто каждый игрок совершает еще и некоторое число шагов. Если эти параметры игры (число игроков, число стратегий, число шагов) достаточно большие, то найти оптимальную стратегию игрока не представляется возможным. В качестве примера можно рассмотреть игру шахматы.

В обычной позиции в шахматах может существовать приблизительно 40 вариантов ходов (стратегий) и столько же ответных – каждая пара полуходов это 40 * 40 = 1600 позиций, две пары 1600 * 1600 = 2 560 000, три пары приблизительно 4 млрд позиций. Математиками подсчитано, что всего различных позиций в шахматных партиях с учетом того, что часть позиций невозможна по правилам, оценивается внушительным числом 10^{40} [1]. Для анализа подобного числа позиций и хранения результата, к сожалению, не хватает вычислительных мощностей и объема памяти даже самых современных суперкомпьютеров.

Игры с большим числом участников сегодня можно видеть на биржах. Каждый участник торгов является игроком, и естественно хочет найти оптимальную стратегию, максимально прибыльную для него. Торги идут с такой скоростью, что промедление в несколько миллисекунд может стоить большой суммы денег (рис. 25) [14].

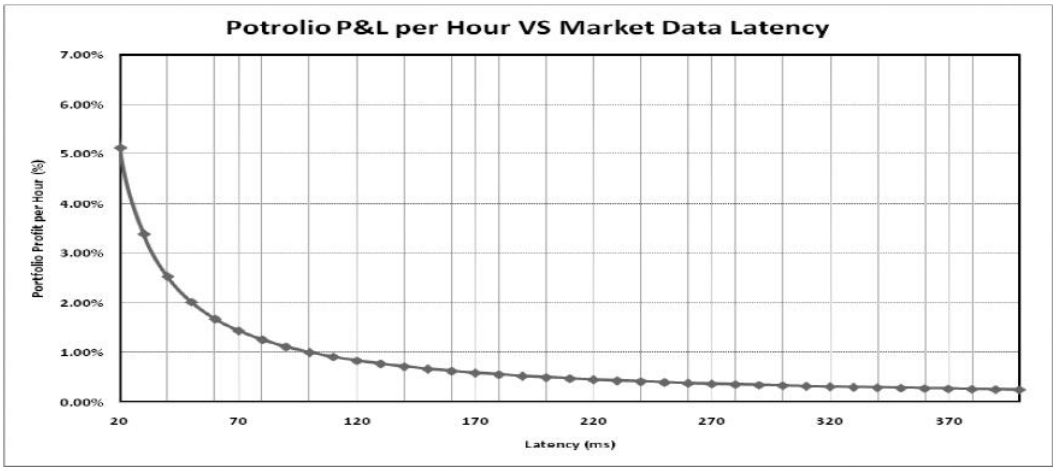


Рис. 25. Зависимость прибыли в процентах от задержки реакции по времени (мс)

Поэтому сегодня большинство крупных участников торгов имеют свои компьютерные системы аналитики рынка, быстро реагирующие на изменение ситуации.

В данной работе на частном примере многошаговой игры на поле показывается возможность искусственного построения игрока успешно справляющегося со своей задачей (соответствующей правилам игры).

Игрок рассматривается как конечный автомат, который на входные воздействия (текущая позиция), выдает выходное воздействие – следующая позиция. Например, для игры шахматы на хранение подобного автомата необходимо хранилище сопоставимое с размерами Луны [1]. Поэтому в работе рассматривается класс игр, в которых игрок видит лишь часть доски и ходит оптимально исходя из локального оптимума.

Возникает вопрос, как получить автомат, если в урезанной игре он влезает в память, но все же велик и сложен для рассмотрения человеком. В работе показывается, что построение подобных автоматов возможно с помощью генетических алгоритмов.

Идея применения эволюционного моделирования для прикладных математических и компьютерных задач появилась еще в начале 50-х годов с развитием генетики. В 1954 году была опубликована работа «Численные примеры процессов эволюции» (ит. «Esempi numerici di processi di evoluzione») норвежско-итальянского математика Нилса Алла Баричелли (Nils Aall Barricelli) [14]. Она не получила широкой известности. Начиная с 1957 года, австралийский генетик Алекс Фрейзер (Alex Fraser) опубликовал серию работ по моделированию искусственного отбора организмов [34]. Эти работы уже содержали все основные элементы современных генетических алгоритмов. В 1960 году американский ученый Лоуренс Дж. Фогель (Lawrence J. Fogel) представил Конгрессу США доклад на тему эволюционного программирования (на тот момент Фогель проводил эксперименты по выращиванию конечных автоматов) [35]. В 1964 году немецким ученым Инго Рехенбергом (Ingo Rechenberg) была разработана идея эволюционных стратегий, а в 1965 году опубликована статья «Путь кибернетического решения экспериментальной проблемы» (англ. «Cybernetic Solution Path of an Experimental Problem») [36]. Его идеи были развиты Ханс-Полом Швефелом (Hans-Paul Schwefel) [50]. Наконец, в 1975 году вышла известная книга американского ученого Джона Холланда (John Holland) «Адаптация в естественных и искусственных системах» (англ. «Adaptation in Natural and Artificial Systems») [36], в которой были даны теоретические основы генетических алгоритмов (изначально применялись для оптимизации функций, представленных битовой строкой). Позже генетические алгоритмы стали применять для выращивания деревьев в функциональном программировании [41], автоматов и других структур, при помощи которых можно представить

компьютерные программы, данная область применения генетических алгоритмов получила название генетическое программирование.

Одним из первых коэволюцию применил в 1988 году Джон Миллер для решения задачи «Дилемма заключенных» [41]. В этой задаче каждый из двух игроков-преступников решает, сдать ли ему подельника или молчать. Сколько лет сидеть каждому из них зависит от выбора обоих.

Вначале работы рассмотрены основные понятия и предыстория вопроса на кафедре компьютерных технологий СПбГУ ИТМО. Затем описываются теоретические основы изучаемой игры, конкретные применяемые операторы в генетических алгоритмах. В конце представлена практическая реализация игры с графическим интерфейсом, рассмотрены графики обучения особей игроков. Результаты экспериментов дают численное представление о возможности применения различных подходов на практике. В заключении делаются основные выводы, рассматриваются перспективы развития.

1.2.1. Основные понятия

В этом разделе рассмотрены такие фундаментальные понятия как игра, конечный автомат, генетический алгоритм. Затем рассказывается об игре, от которой произошла «Задача о муравьеде и муравьях».

1.2.1.1. Игра

На интуитивном уровне каждому понятно, что есть игра. Несколько игроков договариваются о правилах и далее в соответствии с ними делают некоторые манипуляции и в конце определяют выигравших и проигравших.

Для однозначного рассмотрения игры следует формализовать определение. Игры бывают с двумя игроками и с N игроками, антагонистические, бескоалиционные, кооперативные, одношаговые, многошаговые и другие. Для каждого типа игры можно написать свое формальное частное определение. В работе не будет даваться скорее философское наиболее общее определение игры, так как для дальнейшего рассмотрения в этом нет необходимости. Для понимания сути дадим последовательно определения частных типов игр и в конце скажем, что такое гибридная асимметричная игра.

Система $\Gamma = \left(N, \{X_i\}_{i\in N}, \{K_i\}_{i\in N}\right)$, в которой $N = \{1,2,\ldots,n\}$ – множество игроков, X_i – *множество* стратегий игрока i, K_i – функция выигрыша игрока i, определенная на декартовом произведении множеств стратегий игрока $X = \prod_{i=1}^{n} X_i$ (множество ситуаций игры), называется *бескоалиционной игрой* [14].

Поясним определение: игроки независимо друг от друга и одновременно выбирают свои стратегии x_i из множества стратегий X_i (i свое для каждого игрока), в результате формируется ситуация $x = (x_1,\ldots,x_n), x_i \in X_i$. После этого каждый игрок получает свой выигрыш $K_i(x)$.

Важным фактом является независимость выбора – никто ни с кем не общается и не договаривается.

Так как число игроков конечно, каждый может рассмотреть все множество своих стратегий, зависящих от выбора других игроков, и выбрать себе оптимальную стратегию. Определим, что такое оптимальная стратегия.

Пусть $x = (x_1,\ldots,x_{i-1},x_i,x_{i+1},\ldots,x_n)$ – ситуация в игре Г. Если игрок i заменит свою стратегию x_i на $x_i^{'}$, то получим ситуацию $x = (x_1,\ldots,x_{i-1},x_i^{'},x_{i+1},\ldots,x_n)$, которую обозначим через $\left(x \parallel x_i^{'}\right)$.

Ситуация $x^* = \left(x_1^*, \ldots, x_i^*, \ldots, x_n^*\right)$ называется *ситуацией равновесия по Нэшу*, если $\forall x_i \in X_i$ и i = 1, …, n имеет место неравенство $K_i\left(x^*\right) \geq K_i\left(x^* \parallel x_i\right)$.

Будем называть стратегии, входящие в ситуацию равновесия по Нэшу, *оптимальными*. Как известно из теории игр, в большинстве игр с конечным числом стратегий не существует ситуации равновесия по Нэшу [14].

Поэтому вводят смешанное расширение игры. Дадим основные определения.

Смешанная стратегия – случайная величина, значениями которой являются стратегии игрока. Каждой стратегии сопоставляется некоторая вероятность ее выбора (она зависит от вероятности успешного исхода игры для игрока при ее выборе).

Старые стратегии, которые игрок выбирал однозначно, будем называть *чистыми*.

Пусть $\Gamma = \left(N, \{X_i\}_{i \in N}, \{K_i\}_{i \in N}\right)$ – произвольная бескоалиционная игра, μ_i – произвольная смешанная стратегия игрока i. Через $\mu_i(x_i)$ обозначим вероятность, которую стратегия μ_i приписывает чистой стратегии $x_i \in X_i$, а через $\overline{X_i}$ – множество всех смешанных стратегий игрока i. Напомним, что выбор игроки делают независимо, поэтому вероятность появления ситуации $x = \left(x_1, \ldots, x_n\right)$ равна произведению вероятностей выбора составляющих стратегий, $\mu(x) = \mu_1(x_1) \times \ldots \times \mu_n(x_n)$. Данное соотношение определяет вероятностное распределение на множестве $X = \prod_{i=1}^{n} X_i$. Набор $\mu = \left(\mu_1, \ldots, \mu_n\right)$ называется *ситуацией в смешанных стратегиях*.

Определим функцию выигрыша в смешанных стратегиях: $H_i(\mu) = \sum_{x \in X} K_i(x)\mu(x) = \sum_{x_1 \in X_1} \ldots \sum_{x_n \in X_n} K_i(x_1, \ldots x_n) \times \mu_1(x_1) \times \ldots \times \mu_n(x_n), i \in N, x = (x_1, \ldots, x_n) \in X$.

Игра $\overline{\Gamma} = \left(N, \{\overline{X_i}\}_{i \in N}, \{H_i\}_{i \in N}\right)$, в которой N – множество игроков, $\overline{X_i}$ – множество смешанных стратегий каждого игрока i, а функция выигрыша определяется как $H_i(\mu)$, называется *смешанным расширением игры* Γ [14].

Ситуацию μ^* равновесия по Нэшу в смешанных стратегиях можно определить аналогично ситуации x^* в чистых стратегиях.

В смешанном расширении при условии непрерывности функции выигрыша и компактности множеств стратегий всегда существует ситуация равновесия по Нэшу в смешанных стратегиях [14].

Рассмотрим бескоалиционную игру $\Gamma = \left(N, \{X_i\}_{i \in N}, \{K_i\}_{i \in N}\right)$. Разрешим игрокам создавать коалиции $S \subset N$. Коалиция действует в игре как один игрок. Объединяя усилия, каждый игрок может получить больше, чем если бы действовал в одиночку. Коалиция в качестве стратегий имеет всевозможные комбинации стратегий игроков, входящих в коалицию, $X_S = \prod_{i \in S} X_i$. Исходя из общности интересов игроков, входящих в коалицию, следует ввести коалиционную функцию выигрыша – $K_S = \sum_{i \in S} K_i(x)$. Модифицированная игра называется *кооперативной* [14].

Гибридная игра включает в себя элементы бескоалиционных и кооперативных игр. Игроки могут объединяться в коалиции, преследовать общую цель, но при этом каждый игрок старается достичь личной выгоды.

Подобным образом проходит *Открытая математическая олимпиада Санкт-Петербурга среди студентов технических вузов*. Каждый вуз выставляет команды. В команде участники решают независимо задачи, это влияет на их личный рейтинг, а также на общий рейтинг команды. Они при этом не общаются. Таким образом, есть элементы бескоалиционной игры – каждый сам за себя – но, как и в кооперативной, выигрыш суммируется для всей команды.

В *симметричных* играх у участников стратегии равны – если игрок поменяется ролями с кем-то и будет придерживаться той же стратегии, что и раньше, то полученный выигрыш не изменится.

В *асимметричных* играх это правило изменяется. Выигрыш игрока зависит от роли, которую он выполняет в игре. Например, кто-то может обладать возможностями, которые существенно определяют ход игры. Например, в *Совете Безопасности Организации Объединенных Наций* пять стран обладают правом вето, что выделяет их среди остальных игроков.

Рассмотрим игру шахматы. В ней игроки делают ходы по очереди, получая на каждом ходу новую позицию. Это можно представить (с некоторыми уточнениями про повторные позиции), как дерево, где в каждой вершине определена позиция и номер игрока, которому предстоит сделать ход. Потомки вершины – это те позиции, в которых он может оказаться, сделав ход (рис. 26) [14]. Игра конечная, поэтому в конечных вершинах определены функции выигрыша (в шахматах – отображение в $\{0, 1, ½\}$).

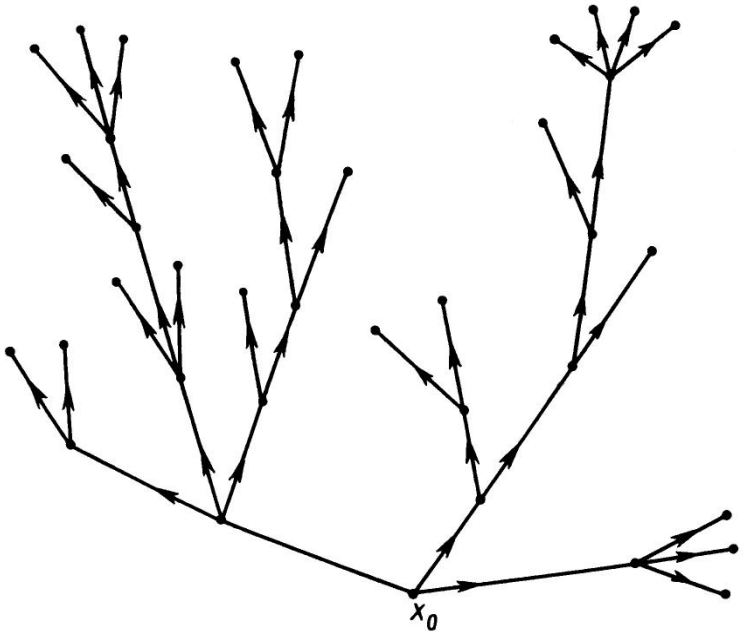


Рис. 26. Представление многошаговой игры в виде дерева, x_0 – корень дерева – начало игры

Пусть $G = (X, F)$ – древовидный граф (дерево), X – множество вершин, F – многозначное отображение из X в X, где F_x – отображение из $x \in X$ в $X' \subset X$ – множество вершин, в которые можно попасть из данной, и находящиеся на уровень в дереве ниже.

Рассмотрим разбиение множества вершин X на $n+1$ множество $X_1, \ldots, X_{n+1}, \bigcup_{i=1}^{n+1} X_i = X, X_k \bigcap X_l = 0, k \neq l, \forall x \in X_{n+1} F_x = 0$. Множество X_i называется *множеством очередности игрока* i, а множество X_{n+1} – *множеством окончательных позиций*. На множестве окончательных позиций X_{n+1} определены n вещественных функций $H_1(x), \ldots, H_n(x), x \in X_{n+1}$, H_i – выигрыш i-го игрока.

Начиная с корня дерева, игрок, множеству очередности которого принадлежит текущая вершина x, делает ход согласно отображению F_x.

Если в игре каждый игрок, находясь в определенной вершине дерева, может восстановить последовательность ходов, начиная с корня, то говорят, что игроки имеют *полную информацию*.

Однозначное отображение u_i, которое каждой вершине $x \in X_i$ ставит в соответствие некоторую вершину $y \in F_x$, называется *стратегией игрока* i. Назовем U_i множеством всевозможных стратегий игрока i. Набор $u = (u_1, \ldots, u_i, \ldots, u_n), u_i \in U_i$ называется ситуацией в игре. Ситуация однозначно определяет игру – ей соответствует набор вершин в дереве, по которым передвигаются игроки. Введем понятие функции выигрыша в каждой такой вершине, как выигрыш игрока в конечной вершине: $K_i(u_1, \ldots, u_i, \ldots, u_n) = H_i(x_l), i = 1, \ldots, n$, а x_l – конечная вершина игры, определяемой ситуацией.

Игра $\Gamma = (N, \{U_i\}_{i \in N}, \{K_i\}_{i \in N})$, в которой $N = \{1, 2, \ldots, n\}$ – множество игроков, U_i – множество стратегий игрока i, K_i – функция выигрыша игрока i, и игроки обладают полной информацией, называется *многошаговой игрой с полной информацией на древовидном графе* [14].

Построив дерево, можно начиная с конечных вершин, провести анализ лучших стратегий, идя к корню. Сопоставляя каждой вершине потомка, в которого выгодней пойти, легко определить путь, по которому игроки будут двигаться в игре, зная ответные ходы остальных. Таким образом, в многошаговых играх с полной информацией всегда существует оптимальная стратегия. Заметим, что этот тип игры является частным случаем бескоалиционных: каждой стратегии в многошаговой игре можно сопоставить чистую стратегию в бескоалиционной.

Можно представить ситуацию, когда игрок, находясь в определенной вершине дерева, знает лишь некоторое множество, к которому его вершина принадлежит, а не конкретное свое местоположение. Поэтому нельзя восстановить путь, по которому он пришел. Игроки не обладают полной информацией. Такая игра называется *многошаговой игрой с неполной информацией*.

В дальнейшем будут интересовать *многошаговые гибридные асимметричные игры с неполной информацией*.

1.2.1.2. Конечный автомат

Рассмотрим объект управления (ОУ, игрок), который подчиняется некоторому субъекту управления (СУ). Для применения генетических алгоритмов (подробно описываются ниже) необходимо, чтобы СУ легко видоизменялся. В работе используется конечный автомат типа Мили.

Данное представление полностью определяет поведение объектов, исключает неопределенные состояния. Конечным детерминированным *автоматом типа Мили* называется совокупность пяти объектов [21]:

$A = (S, X, Y, \delta, \lambda)$, где S (множество состояний автомата), X (множество входных воздействий) и Y (множество выходов) – конечные и непустые, а δ и λ – отображения вида:

$\delta: S \times X \to S$,

$\lambda: S \times X \to Y$

со связью элементов множеств S, X и Y в абстрактном времени $T = \{0, 1, 2, \ldots\}$ уравнениями:

$s(t + 1) = \delta(s(t), x(t))$,

$y(t) = \lambda(s(t), x(t))$,

где t из T.

Отображение δ называется функцией переходов, отображение λ – функцией выходов автомата A.

Таким образом, на каждое входное воздействие автомат реагирует переходом в новое состояние и выходным воздействием.

Конечный автомат типа Мили не сложно представить в виде однонаправленного графа, где множество состояний S соответствует множеству вершин, отображение δ задает множество ребер, а

значения функции λ и множество входных воздействий *X* записываются над ребрами графа. Такое представление называется диаграммой состояний или графом переходов.

1.2.1.3. Генетический алгоритм

Генетический алгоритм – эвристический алгоритм нахождения максимума многопараметрической функции, по способу приближения к решению напоминающий биологическую эволюцию. При этом используются функция приспособленности, операторы скрещивания, отбора и мутации [41, 1]. Введем основные понятия генетических алгоритмов [36, 1, 22].

Особь – специальным образом закодированное представление оптимизируемого объекта.

Популяция – конечное множество особей.

k-е поколение – популяция после *k* итераций генетического алгоритма.

Функция приспособленности принимает на вход решение и выдает его оценку.

Оператор скрещивания принимает на вход два решения, например, представленные битовой строкой, и создает два новых решения, полученных путем синтеза из входных – потомки состоят из частей решений предков. Синтез может происходить путем обмена определенными частями предков либо случайной переброской составляющих решений.

Оператор мутации принимает составляющие решения и с некоторой вероятностью их случайно изменяет, получая, таким образом, новое решение.

Элита – часть особей с лучшими результатами функции приспособленности, переходящая в следующее поколение без изменения.

Условие останова – при достижении какого значения функции приспособленности следует остановиться.

В операторе отбора генетического алгоритма выбираются *n* лучших на данный момент решений. Обычно для этого используют функцию приспособленности, определяющую может ли решение попасть во множество лучших [41, 1].

Классический генетический алгоритм выполняет процедуру оптимизации в следующем порядке:

1. Формируется начальное поколение особей, чаще всего путем их случайного генерирования.
2. Подсчитываются результаты функций приспособленности особей.
3. Происходит отбор особей в промежуточное поколение.
4. Особи из промежуточного поколения для формирования следующего поколения подвергаются изменению операторами скрещивания и мутации. При этом после оператора скрещивания потомки заменяют своих родителей в следующем поколении.
5. Если не выполняется условие останова, происходит переход к пункту 2.

При практическом исследовании интересует не обобщенное понятие генетического алгоритма, а конкретная реализация всех составляющих. Большинство этапов интуитивно понятны, но оператор отбора содержит некоторую неопределенность. Рассмотрим конкретную его разновидность, «метод рулетки» [1], он нам понадобится впоследствии.

«Метод рулетки» легко объясняется на примере барабана рулетки в казино. Каждому решению отводится площадь сектора на круге, соответствующая значению его функции приспособленности. Можно принять сумму значений функций приспособленности всех особей из поколения за единицу. Колесо рулетки можно представить в виде отрезка [0, 1]. Тогда на отрезке [0, 1] каждой особи будет отведен интервал длиной L_i, соответствующий целой части отношению значения ее функции приспособленности $f(i)$ (где i – номер особи) к сумме всех функций приспособленности M: $L_i = \left\lfloor \frac{f(i)}{M} \right\rfloor$, $\sum_{i=1}^{N} L_i = 1$, где N – число особей в поколении. Затем N раз с

помощью генератора псевдослучайных чисел получается число из заданного диапазона (аналог остановки стрелки рулетки на секторе барабана). В следующее поколение попадает та особь, в чей интервал попало это число. Таким образом, решение с большим значением функции приспособленности имеет больший сектор на круге и, соответственно, больше шансов попасть (возможно, и не в одном экземпляре) в следующее поколение (рис. 27).

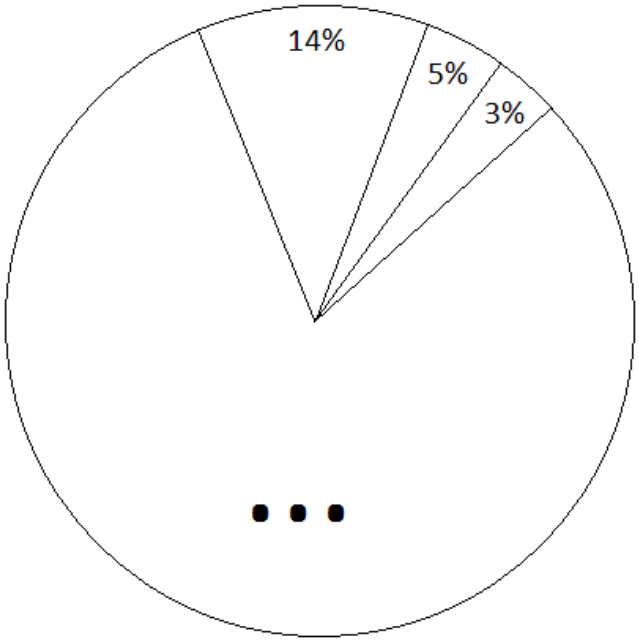


Рис. 27. Иллюстрация к «методу рулетки»

В работе «метод рулетки» применяется до оператора скрещивания и оператора мутации, не давая худшим решениям шанса оставить потомков. Этот вариант приближает генетический алгоритм к реальной биологической эволюции, где худшие особи (решения) не выживают и не дают потомства.

В работе также используется «элитизм» – сохранение лучших решений, элиты, для следующего поколения, не подвергая их изменениям.

1.2.1.4. «Задача об умном муравье-3»

На кафедре компьютерных технологий СПбГУ ИТМО исследовалась «Задача об умном муравье-3» [1]. С нее начались исследования, изложенные в данной работе. Условия задачи просты: задан двумерный тор размером 32 на 32 клетки. В верхнем левом углу тора стоит муравей. На поле случайным образом с заданным коэффициентом заполнения размещены яблоки. Муравей умеет совершать три действия (выходное воздействие автомата):

- повернуть налево;
- повернуть направо;
- пойти на одну клетку вперед, и, если там есть яблоко, съесть его.

Муравей видит восемь клеток (две слева, две справа, две спереди и по одной спереди по диагонали) (рис. 28).

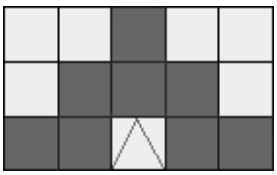


Рис. 28. Муравей и клетки, которые он видит

Решением задачи является автомат, управляющий муравьем, который генерируется с помощью генетического алгоритма. Муравей за ограниченное число ходов (200) должен съесть как можно больше яблок (рис. 29) [2].

Как показано в работе [1] с помощью генетических алгоритмов можно вырастить автомат, успешно управляющий муравьем. При этом обучение на современных компьютерах занимает приемлемое время.

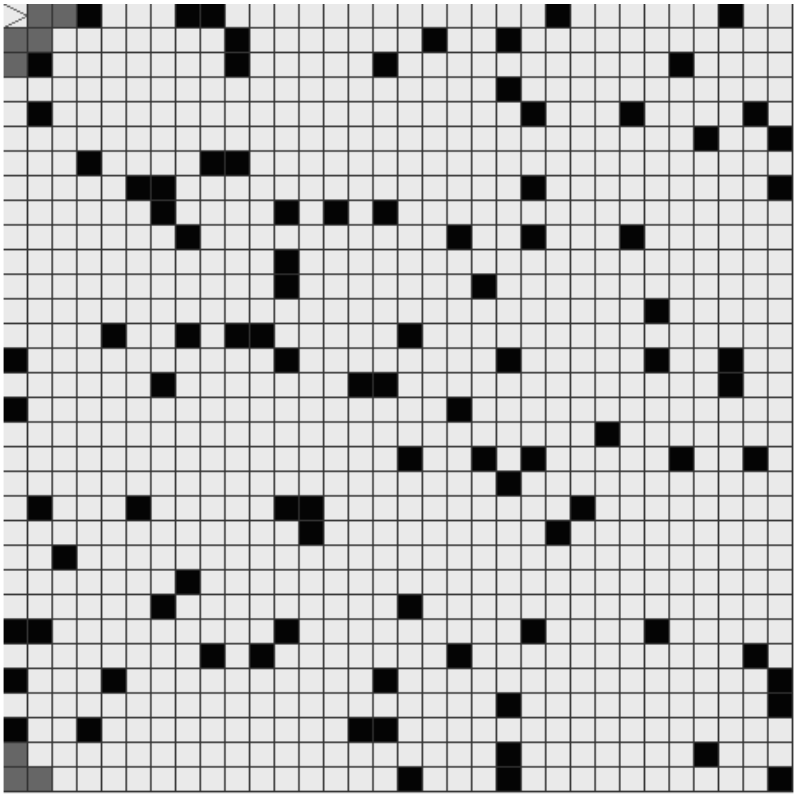


Рис. 29. Тор с муравьем и яблоками. Серая стрелка изображает муравья, серые клетки – это те клетки, которые он видит. Черные квадраты обозначают места расположения яблок.

Рассмотренные определения дают возможность приступить к изложению основной темы данной работы. Возможность решения игры с помощью генетических алгоритмов и коэволюции показывается в последующих разделах.

1.2.2. «Задача о муравьеде и муравьях»

«Задача о муравьеде и муравьях» является обобщением предыдущей задачи. Задано поле – двумерный тор произвольного размера с координатной сеткой, которая делит его поверхность на клетки. На поле расположены муравьед, муравьи (еда для муравьеда) и яблоки (еда для муравьев). Яблоки и муравьи располагаются случайным образом. На одной из клеток находится муравьед (рис. 30).

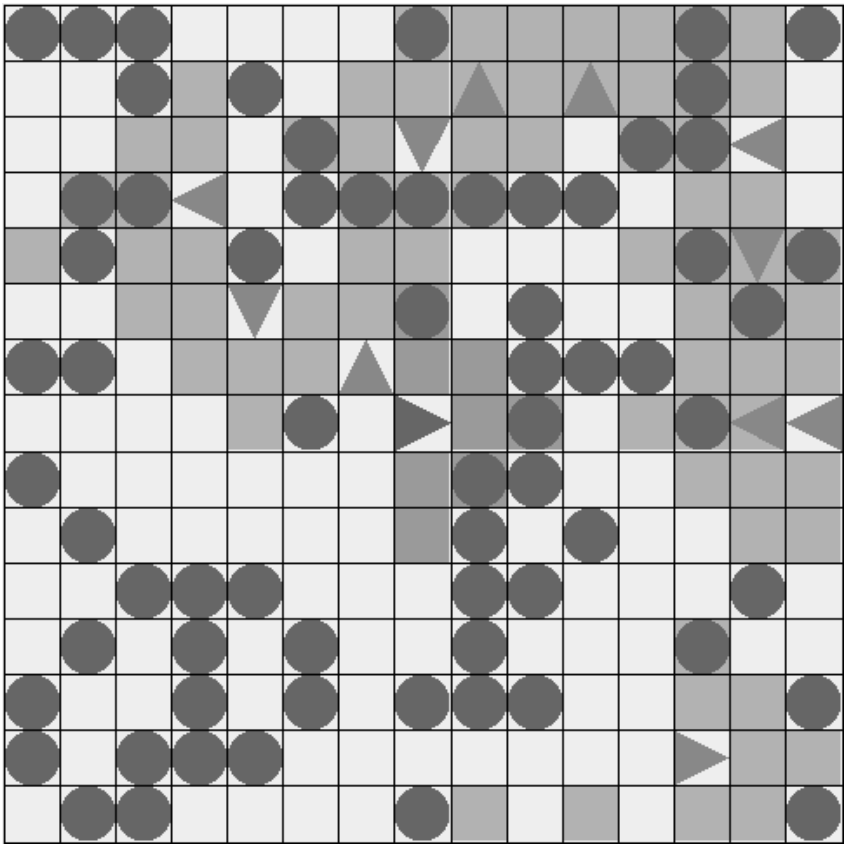


Рис. 30. Тор с муравьедом, муравьями и яблоками. Темный треугольник в центре изображает муравьеда, серые клетки рядом – это те клетки, которые он видит. Серые треугольники – муравьи, серые клетки рядом с ними – та часть поверхности, которую видят муравьи. Темные кружочки обозначают места расположения яблок.

Цель каждого муравья – за заданное число шагов съесть как можно больше яблок и «не попасться» муравьеду. Цель муравьеда – помешать им сделать это, съев за заданное число шагов максимальное число муравьев. Муравьи не должны «наступать» на других муравьев и муравьеда, так как они при этом погибают.

Яблоки лежат на поле неподвижно, а муравьед и муравьи на каждом шаге могут совершить одно из трех действий (выходные воздействия соответствующих автоматов):

- повернуть налево;
- повернуть направо;
- пойти на одну клетку вперед, и, если там есть еда (муравей для муравьеда или яблоко для муравья), то съесть ее.

Муравьед и муравьи видят перед собой по восемь клеток: две слева, две справа, две спереди и по одной спереди по диагонали [22].

При решении данной задачи будем использовать коэволюцию – выращивание решений при их взаимодействии [44, 7, 36].

Для построения генетического алгоритма необходимо разработать способы представления автоматов, выбрать функции приспособленности, операторы скрещивания, мутации и отбора.

Как видно из определения, в задаче рассматривается конечная игра (число игроков определяется правилами) на поле. Она является асимметричной, так как муравьед и муравьи находятся не в равном положении (муравьед может объявить проигравшим любого участника-муравья, если тот окажется на его пути). Игра также является гибридной, так как с одной стороны у муравьев есть общая цель – съесть наибольшее число яблок – кооперативная игра, а с другой каждый муравей действует сам за себя, при этом он даже может помешать собрату, опередив его. Каждый игрок совершает конечное число ходов, видя перед собой ограниченное число клеток – многошаговая игра с неполной информацией.

Подобные игры могут встречаться в разных областях, для них могут придумываться некоторые эвристики. В игре же с довольно сложной системой правил, картой обзора, простые эвристические методы уже не выйдет найти за приемлемое время. На примере данной простой игры постараемся показать применимость генетических алгоритмов и коэволюции, что существенно облегчает задачу поиска оптимальных стратегий.

1.2.2.1. Представление автоматов

Для представления автомата типа Мили в работе используется массив байтов, в котором первый элемент содержит номер начального состояния автомата *S*. Затем следуют *n* секторов (*n* – число состояний автомата), каждый из которых содержит *m* пар чисел (*m* – число входных воздействий), каждая из которых состоит из следующего состояния δ и выходного воздействия λ (рис. 31).

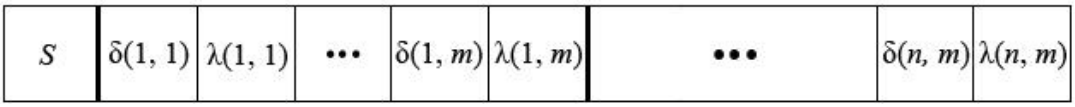


Рис. 31. Представление автомата типа Мили

В рассматриваемой задаче автоматы могут принимать большое число входных воздействий, так как каждая клетка поля может содержать муравьеда, муравья, яблоко или не содержать их.

Муравьед не различает яблоки и пустые клетки, и поэтому его автомат может принять $2^8 = 256$ входных воздействий (на клетке есть муравей либо его нет).

Муравей различает клетки с яблоками и пустые клетки. К тому же он может видеть (если они находятся в поле его зрения) других муравьев (для того, чтобы не наткнуться на них) и муравьеда (для того, чтобы вовремя убежать от него). Таким образом, автомат муравья может принять $3^8 + 8 * 3^7 = 6561 + 8 * 2187 = 24057$ входных воздействий.

1.2.2.2. Функции приспособленности

В данном разделе описываются функции приспособленности для метода коэволюции, в других методах значения вычисляются с учетом отсутствия обучения автомата муравьеда или муравья.

Функция приспособленности автомата, управляющего муравьем, равна числу съеденных им яблок. Функция приспособленности автомата, управляющего муравьедом, вычисляется аналогично, но при этом вместо яблок используются муравьи.

При решении задачи эти функции приспособленности вычисляются следующим образом: по очереди из текущего поколения выбирается один автомат, управляющий муравьедом, и для него случайным образом из текущего поколения автоматов, управляющих муравьями, выбирается заданное в эксперименте число различных автоматов k (для других методов выбирается один автомат муравья и делается k копий).

На поле размещаются муравьед и k муравьев, управляемых этими автоматами. При этом для яблок и муравьев генерируются случайные начальные координаты, а для каждого муравья также случайно указывается начальное направление движения. Все участвующие в эксперименте особи (муравьед и муравьи) совершают заданное число шагов.

Выбранные муравьед и муравьи имеют заданное число попыток для случайного размещения на поле муравьев и яблок. При этом для каждой особи в отдельности значения функции приспособленности автомата на всех полях суммируются, а в конце эксперимента сумма делится на число совершенных попыток. Если число попыток у особей разное, то последние попытки не учитываются у тех особей, для которых было задано меньшее число попыток. В результате для каждой особи получается среднее (за несколько попыток) число съеденных ею единиц еды. Если несколько муравьев хотят пойти на одну и ту же клетку, то приоритет имеет тот муравей, чей автомат обладает наибольшим значением функции приспособленности (учитываются и предыдущие попытки). При этом автоматы, управляющие муравьями, которые не смогли сделать шаг, остаются в предыдущих состояниях. После того, как сходили все муравьи, ходит муравьед, но он не знает новое расположение муравьев. Таким образом, муравьям дается шанс убежать.

Если муравей ходит на клетку, где стоит другой муравей или муравьед, а также если муравья съедают, то он погибает. При этом ему не засчитываются съеденные на текущей попытке яблоки, он оживает на следующей попытке. Стратегия наказания (не засчитывать пищу) помогает научить автоматы муравьев избегать подобных ситуаций.

После вычисления значений функций приспособленности всех автоматов муравьедов, если остаются автоматы муравьев, которые не участвовали в эксперименте, то для каждого из них выбираются автомат муравьеда и необходимое число автоматов муравьев и вычисляются значения их функций приспособленности. При этом функции приспособленности автоматов, вычисленные ранее, имеют после эксперимента предыдущие значения.

В заключение раздела отметим, что в ходе коэволюции в общем случае автоматы муравьев даже при одинаковом числе состояний отличаются друг от друга.

1.2.2.3. Оператор скрещивания

В работе был выбран один и тот же оператор скрещивания для автоматов муравьедов и муравьев. Он обеспечивает компромисс между временем работы и скоростью улучшения автоматов. Оператор скрещивания сначала копирует автомат первого предка в одного потомка, а автомат второго – в другого потомка. Затем он случайно устанавливает потомкам различные начальные состояния от предков. После этого тридцать раз для каждого из потомков выбирается фрагмент другого предка (чей копией потомок до этого не был) длиной в один процент от общей длины (напомним, что автоматы хранятся в виде байтовой строки), и этот фрагмент устанавливается на те же позиции потомка, на которых он был у предка.

1.2.2.4. Оператор мутации

В операторе мутации для муравьеда и муравья с заданной вероятностью изменяется начальное состояние автомата. Затем выполняется проход по всем состояниям (в каждом из них, в свою очередь,

выполняется проход по всем входным воздействиям) и с той же вероятностью на случайные изменяются номера состояний для переходов и выходные воздействия.

1.2.2.5. Оператор отбора автоматов в следующее поколение

Каждое поколение в генетическом алгоритме делится на элиту (автоматы с наибольшими значениями функций приспособленности) и остальные автоматы. Сначала в следующее поколение переходят автоматы из элиты. Далее для отбора автоматов используется «метод рулетки». К автоматам, которые были отобраны «методом рулетки», применяется оператор скрещивания, а к их потомкам – оператор мутации. После этого они добавляются в следующее поколение к элите.

В разделе были изложены теоретические основы «Задачи о муравьеде и муравья». Описанные операторы скрещивания, мутации, отбора и функция приспособленности используются в работе в дальнейшем для получения результатов. Практическая реализация задачи и результаты опытов будут рассмотрены в следующем разделе.

1.2.3. Экспериментальное исследование

В разделе рассмотрено инструментальное средство, позволяющее изучать «Задачу о муравьеде и муравьях». Далее приводятся результаты экспериментов, показывается применимость метода генетических алгоритмов и коэволюции в играх.

1.2.3.1. Инструментальное средство

Для решения поставленной задачи разработано инструментальное средство, поддерживающее описанные особенности рассмотренного генетического алгоритма. Это средство позволяет проследить процесс выращивания автоматов, а также визуализировать поведение особей, управляемых этими автоматами на случайном поле с муравьями и яблоками.

Главной частью интерфейса программы являются две вкладки («Лучший муравьед» (рис. 32) и «Лучший муравей» (рис. 33)). Каждая вкладка содержит диаграмму, на которой отображается график зависимости наибольшего числа съеденных объектов пищи от номера поколения. В правом верхнем углу для удобства пользователя выводятся данные о прошедшем на данный момент числе поколений и о числе съеденных лучшим муравьедом (муравьем) объектов.

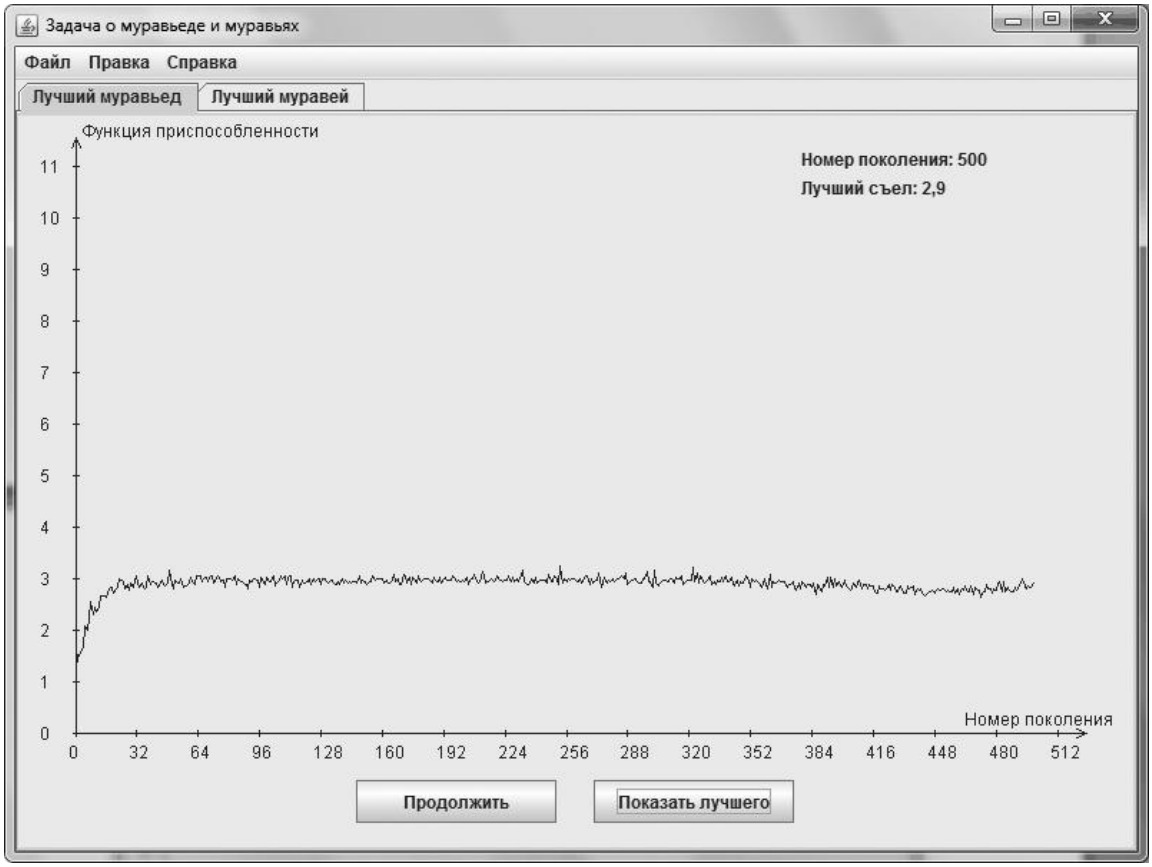


Рис. 32. Главное окно с открытой вкладкой для лучшего муравьеда

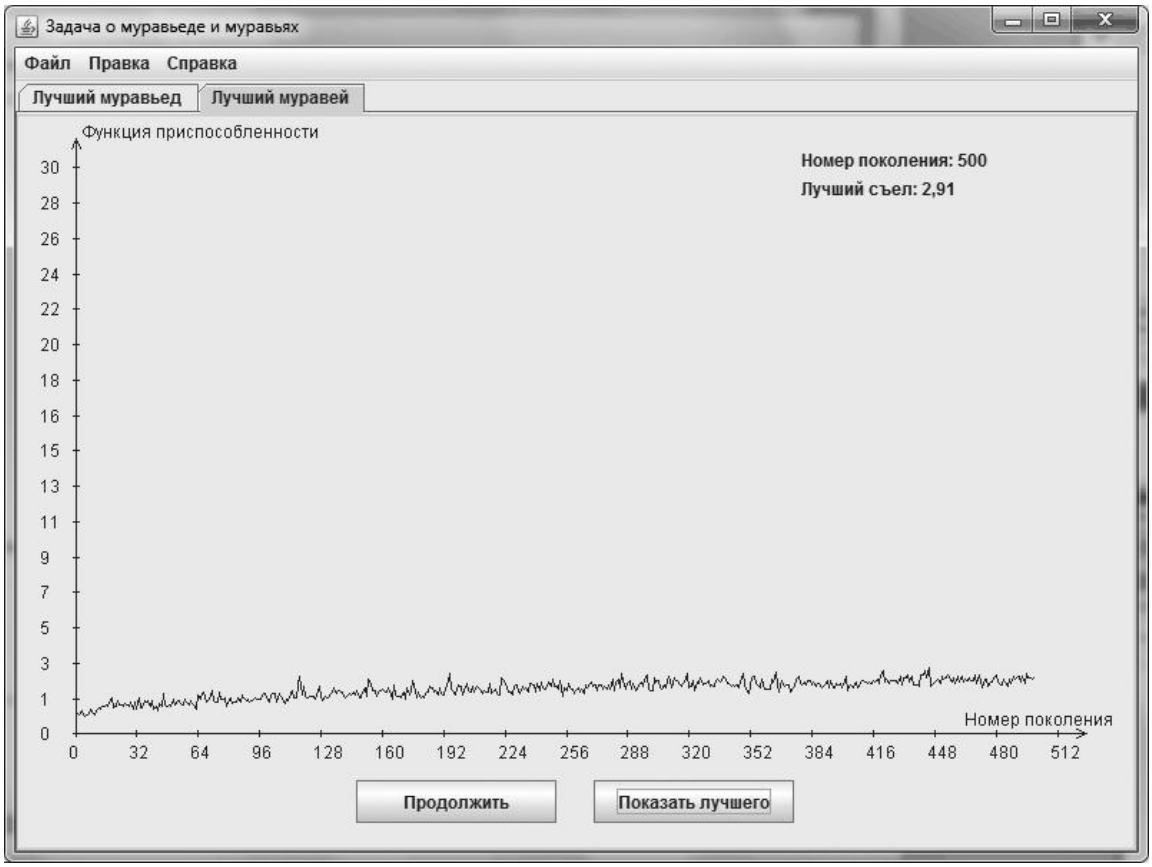
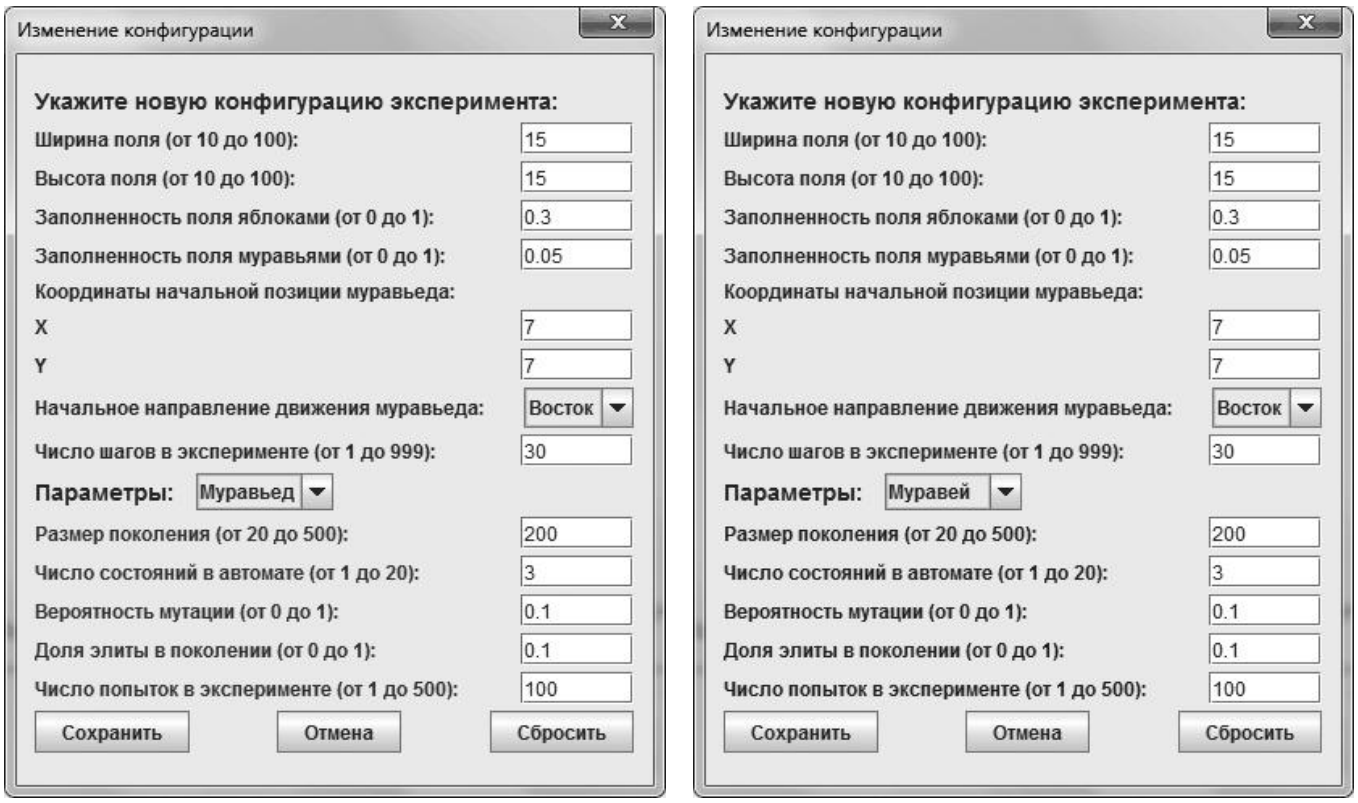


Рис. 33. Главное окно с открытой вкладкой для лучшего муравья

На каждой вкладке под диаграммой размещены две кнопки: одна для старта, приостановки и продолжения генетического алгоритма, а вторая для визуализации лучшего на данный момент муравьеда и *k* лучших муравьев, полученных на момент нажатия кнопки.

При незапущенном или приостановленном алгоритме с помощью поля меню «Файл» имеется возможность загрузить ранее сохраненное поколение и продолжить генерацию автоматов с того места, на котором она была ранее закончена. Если алгоритм приостановлен, можно сохранить текущее поколение, либо сохранить в текстовый файл лучший автомат муравьеда или лучший автомат муравья. В текстовом файле в первой строке указан номер начального состояния автомата, далее файл разбит на четыре колонки: в первой указывается номер состояния, во второй – номер входного воздействия, в третьей – выходное воздействие (L – повернуться налево, R – повернуться направо, F – пойти вперед), в четвертой – номер следующей вершины автомата.

Через поле меню «Правка» имеется возможность задавать и изменять параметры эксперимента: ширина поля, высота поля, заполненность поля яблоками, заполненность поля муравьями, начальные координаты муравьеда, начальное направление движения муравьеда, число шагов в эксперименте. Внизу в списке можно выбирать параметры муравьеда или муравья (рис. 34) (различаются цветом). Ниже списка отображаются поля, касающиеся только выбранной особи.



а б

Рис. 34. Окно «Изменение конфигурации» с параметрами муравьеда (а) и муравья (б)

Для автоматов муравьеда и муравьев (все автоматы муравьев имеют одинаковые параметры) раздельно можно задать следующие параметры: размер поколения, число состояний в автомате, вероятность мутации, доля элиты в поколении, число попыток в эксперименте (число случайных полей, на которых особь пытается добиться своей цели и улучшить значение функции приспособленности своего автомата).

При необходимости можно, нажав на кнопку «Сбросить», установить стандартные параметры, используя которые, получены результаты, которые приведены ниже. По нажатию кнопки «Сохранить» сохраняются изменения, и закрывается окно «Изменение конфигурации». Кнопка «Отмена» закрывает окно, не сохраняя изменения.

Через поле меню «Правка» можно изменять язык интерфейса, доступны русский и английский языки.

Как было изложено выше, нажав в главном окне на кнопку «Показать лучшего», можно увидеть случайно сгенерированное поле с расположенными на нем муравьями и муравьедом (рис. 35). Муравьед и муравьи являются лучшими из всех поколений.

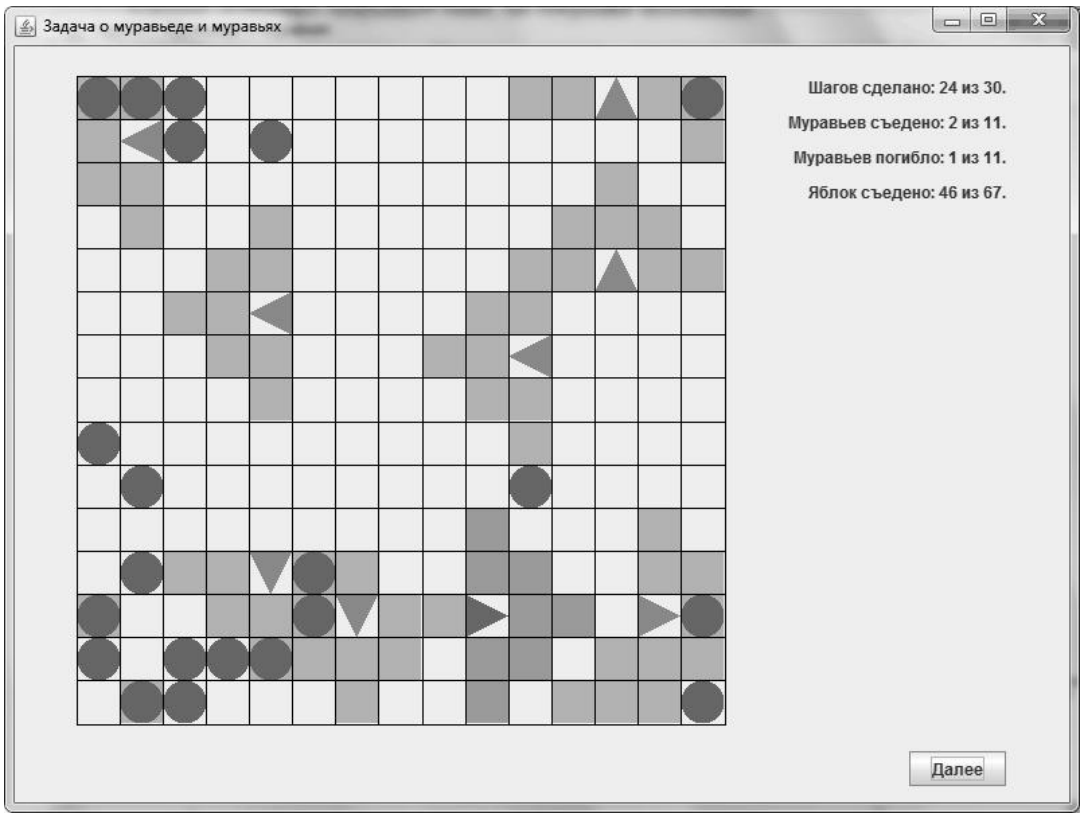


Рис. 35. Вид поля после 24 шагов

По нажатию кнопки «Далее» все особи, присутствующие на поле, делают следующий шаг. В верхнем правом углу окна расположены вспомогательные данные, сообщающие о том, сколько шагов сделали особи, сколько съедено яблок и муравьев, сколько муравьев погибло.

1.2.3.2. Результаты экспериментов

В работе для решения гибридной асимметричной игры были применены различные подходы:

1. коэволюция;
2. случайные особи;
3. выращивание на случайных.

Каждый из этих трех подходов применяется для получения автоматов, управляющих муравьедом и муравьем.

В конце из всех экспериментов выбираются средние по приспособленности особи и сравниваются между собой. Ниже расскажем о каждом из экспериментов подробнее.

В коэволюции для быстрого обучения автоматов, управляющих муравьедом, необходимо иметь большое число муравьев на поле. Аналогично для автоматов муравьев: если на поле больше яблок, то муравьи учатся есть быстрее. «Задача о муравьеде и муравьях» решалась при параметрах, указанных в табл. 2.

Таблица 2. Значения параметров в «Задаче о муравьеде и муравьях»

Параметр	Значение
Ширина поля	15
Высота поля	15
Заполненность поля яблоками	0,3
Заполненность поля муравьями	0,05
Координата X начальной позиции муравьеда	7
Координата Y начальной позиции муравьеда	7
Начальное направление движения муравьеда	Восток
Число шагов в эксперименте	30

Для автоматов муравьедов и муравьев были выбраны одинаковые значения параметров, указанные в табл. 3.

Таблица 3. Значения параметров муравьеда и муравья

Параметр	Значение
Размер поколения	200
Число состояний в автомате	3
Вероятность мутации	0,1
Доля элиты в поколении	0,1
Число попыток в эксперименте	100

Разработанное инструментальное средство при этих параметрах занимает около 170 мегабайт оперативной памяти. Для вычисления следующего поколения требуется примерно 4 с при использовании процессора *Intel Pentium* 4 с частотой 3,2 ГГц.

Из графика на рис. 36 видно, что особь постепенно учится съедать все большее число муравьев (исходно на поле был размещено 11 муравьев). При этом если вначале муравьед передвигается случайным образом и чаще всего вращается на месте (рис. 37), то, как показали эксперименты, уже к 50-му поколению прослеживается его стремление догнать жертву, хотя иногда он сворачивает не туда.

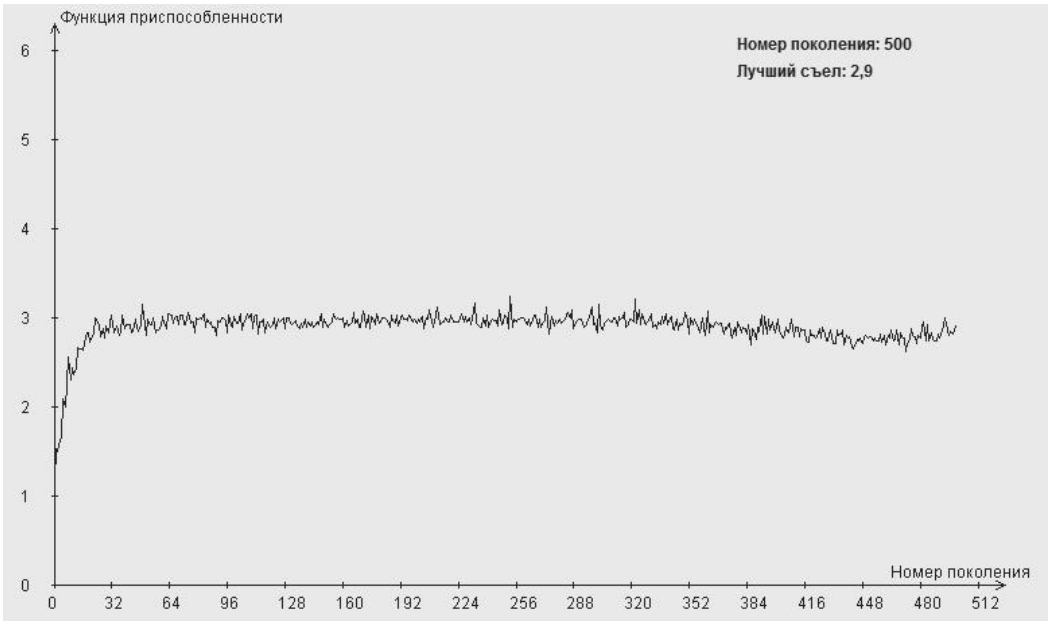


Рис. 36. График зависимости числа съеденных муравьев от номера поколения муравьеда

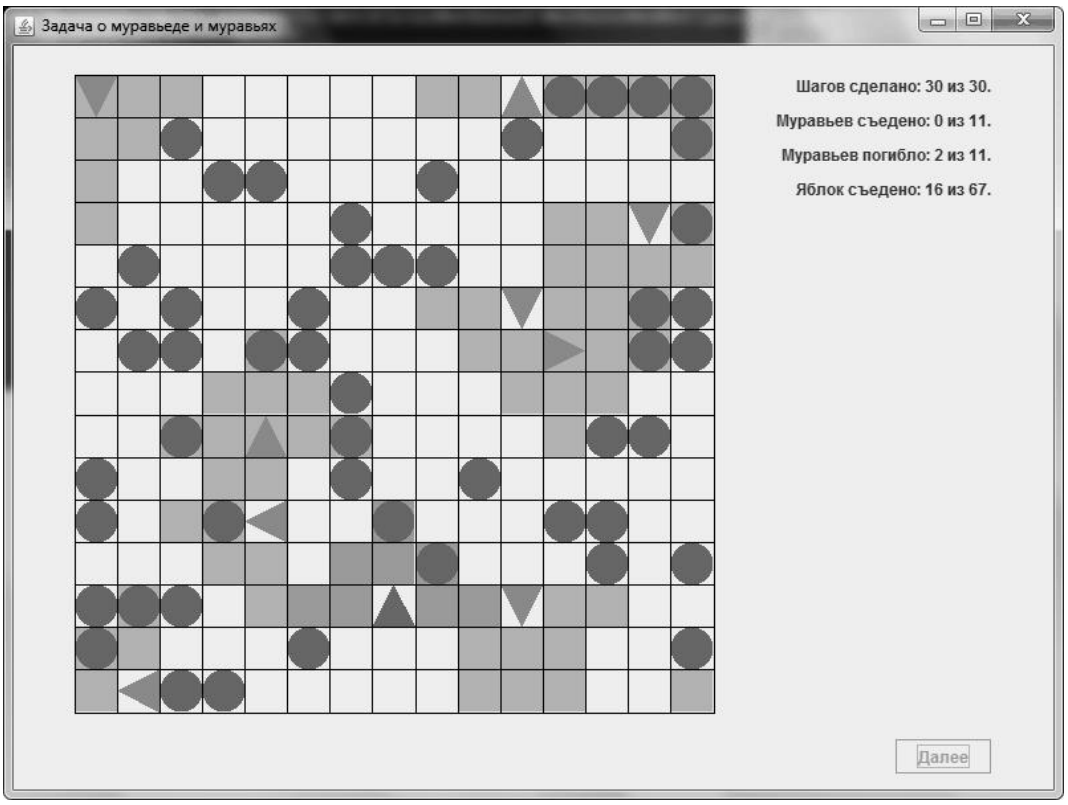


Рис. 37. Вид поля в начале генерации поколений после 30 ходов. Муравьед никого не съел, муравьи съели меньше половины яблок

Чем больше номер поколения, тем меньше подобных ошибок. Ближе к 100-му поколению муравьед уже уверенно преследует муравьев, несмотря на то, что они пытаются от него убежать (муравьи отличают муравьеда от себе подобных и яблок и постепенно учатся к нему не приближаться) (рис. 38).

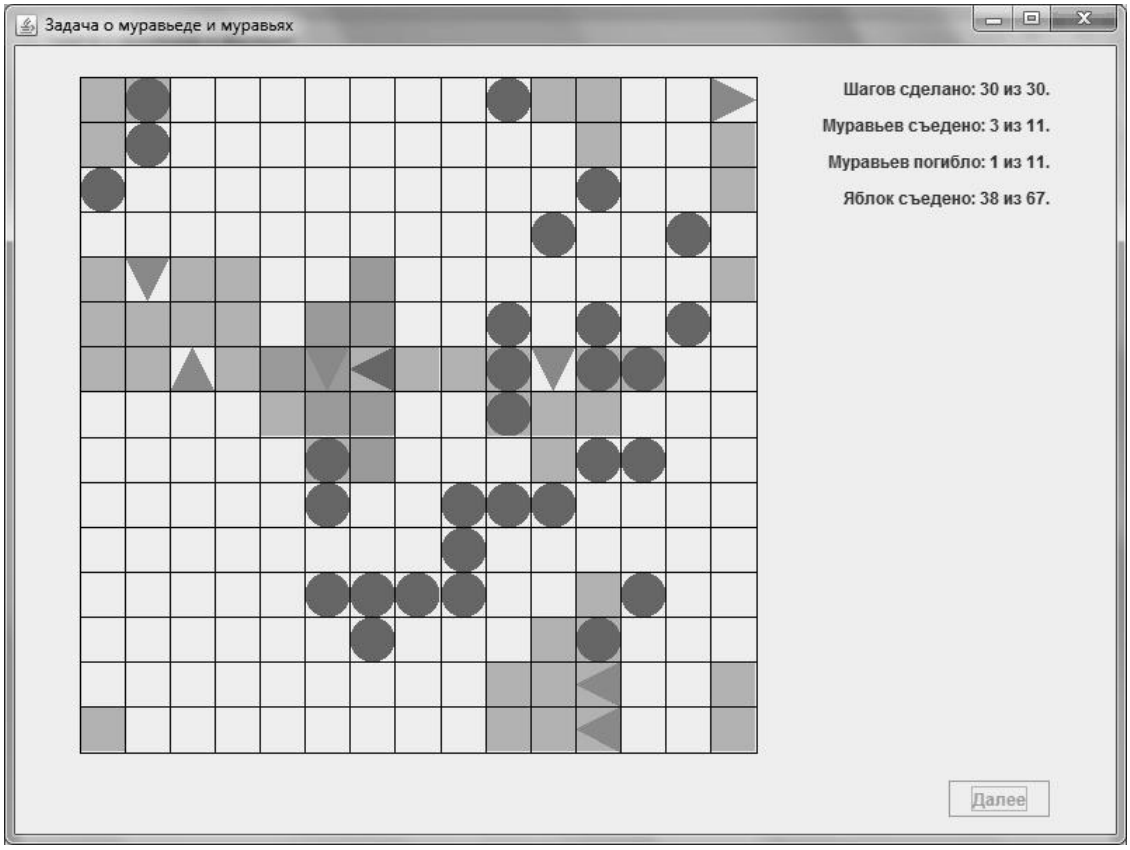


Рис. 38. Вид поля в процессе генерации поколений после 30 ходов. Муравьед съел 3 муравьев, муравьи съели больше половины яблок

При дальнейшем увеличении числа поколений муравьед поедает больше муравьев (см. рис. 36).

На втором графике (рис. 39) виден процесс роста числа съеденных яблок у муравьев от поколения к поколению (сравните рис. 37 и рис. 38). На результат повлияло то, что муравьи фактически соревнуются между собой за яблоки, число которых на поле исходно равнялось 67.

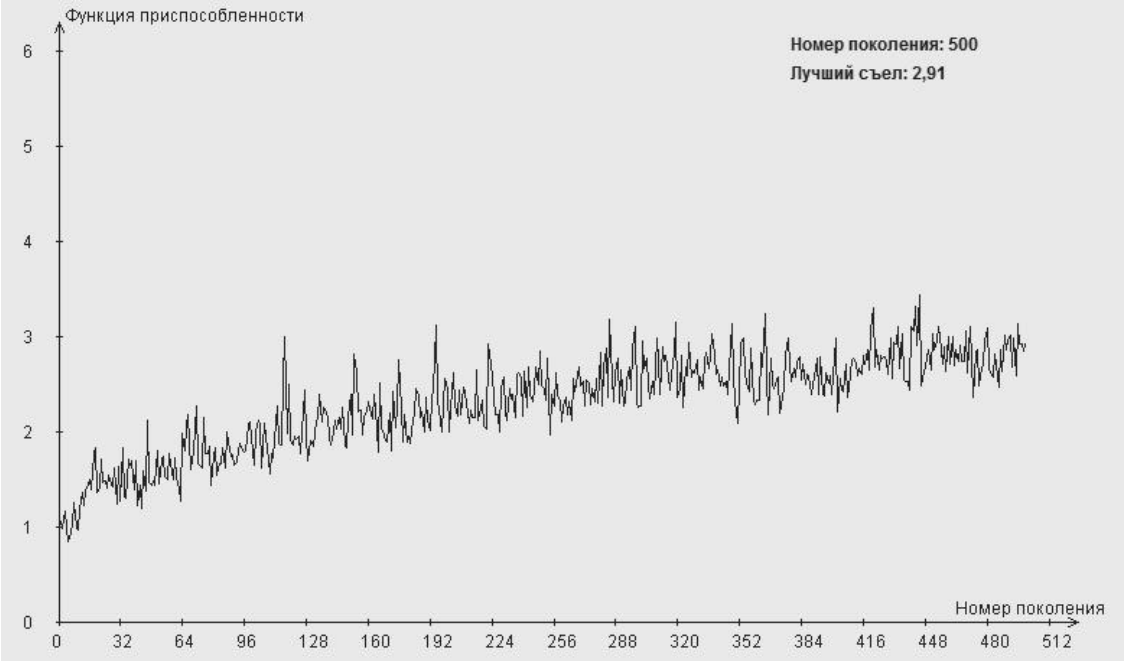


Рис. 39. График зависимости числа съеденных яблок от номера поколения муравьев

Эксперименты по выращиванию проводились до 500-го поколения, значения функции приспособленности муравьеда и муравья в последнем поколении указаны в табл. 4. Всего проводилось 10 опытов.

Таблица 4. Значения функции приспособленности в методе коэволюции

Номер опыта	Муравьед	Муравей
1	2,90	2,91
2	2,92	2,38
3	2,83	2,78
4	2,78	2,81
5	2,89	2,71
6	2,79	3,25
7	2,78	2,58
8	2,96	2,63
9	2,87	3,07
10	2,80	2,57

Необходимо выбрать среднюю пару для сравнения автоматов муравьеда и муравья с полученными иными методами решения рассматриваемой игры. Так как при выращивании с помощью коэволюции есть зависимость от противника, простой выбор со средним значением функции приспособленности представляется некорректным. Поэтому было решено выбрать пару по тому, как муравьед, управляемый автоматом, справляется с муравьями под управлением случайными автоматами. Были сгенерированы 500 случайных автоматов муравьев. На каждый случайный автомат

муравья (на поле все муравьи под управлением копии одного и того же автомата) муравьеду дается 100 попыток. Результаты, усредненные по числу попыток, представлены в табл. 5.

Таблица 5. Значения функции приспособленности автомата муравьеда в методе коэволюции на случайных автоматах муравьев

Номер опыта	Муравьед
1	2,835
2	2,836
3	2,777
4	2,766
5	2,864
6	2,811
7	2,788
8	2,945
9	2,827
10	2,876
Среднее	2,833

Ближе всех к среднему значению оказался атомат под номером 1, он и его коэволюционный соперник – автомат муравья – будут использованы при сравнении в дальнейшем.

В работе для сравнения были сгенерированы случайные автоматы для управления муравьедом и случайные автоматы для управления муравьем (по 500). Они, как и следовало ожидать, справляются со своими задачами довольно плохо, результаты показаны ниже. Эти автоматы в дальнейшем будут использованы при общем сравнении различных методов. Именно то, как полученные автоматы справляются со случайным, будет основным критерием сравнения методов решения игры.

Следующим этапом исследования является попытка вырастить автомат муравьеда при условии, что на поле располагаются муравьи под управлением случайными автоматами.

Параметры эксперимента те же, что и во всех предыдущих опытах. Перед генерацией следующего поколения проводятся эксперименты для 200 автоматов, и каждому дается по 100 попыток. При этом на поле располагаются муравьи под управлением копий одного и того же случайного автомата. В следующих 100 попытках для того же автомата муравьеда случайный автомат муравья уже другой.

Аналогично было проведено выращивание автомата муравья на муравьеде, управляемом случайным автоматом. В попытках на поле выпускались муравьи под управлением копий одного и того же автомата, значение функции приспособленности выбиралось как максимум из всех.

Выращивание, как и ранее, проводилось до 500-го поколения, результаты опытов представлены в табл. 6. Было проведено по 10 опытов.

Таблица 6. Значения функции приспособленности в методе выращивания на случайных

Номер опыта	Муравьед против случайных муравьев	Муравей против случайного муравьеда
1	3,31	4,27
2	3,56	4,51
3	3,50	4,53
4	3,57	4,67
5	3,45	4,30
6	3,47	4,58
7	3,52	4,50
8	3,36	4,49
9	3,65	4,69
10	3,44	4,19
Среднее	3,483	4,473

Для дальнейшего сравнения методов решения игры было решено выбрать особи наиболее близкие к средним значениям функции приспособленности: 6 для муравьеда и 8 для муравья.

В результате экспериментов были выращены особи при различных условиях (в зависимости от того, кто против них играет). Имеются поколения автоматов, выращенных в процессе коэволюции, на случайном сопернике, а также случайные. Заметим, что независимо от метода, муравьи выращиваются в процессе коэволюции, так как хотя не влияют на муравьеда (случайного), они активно взаимодействуют друг с другом.

Теперь проведем анализ результатов. Для сравнения брался каждый из автоматов муравьедов и особь под его управлением выставлялась на поле против каждого из автоматов муравьев (на поле все муравьи подчиняются копиям одного и того же автомата). При этом давалось по 100 попыток на каждую из особей по 500 раз (разница есть при сравнении со случайными автоматами муравьев, когда в первый раз в 100 попытках они совпадают, а в последующих 499 по 100 уже другие). Результаты всех экспериментов, усредненные по числу попыток, приведены в табл. 7.

В табл. 7 по вертикали – автомат муравьеда, по горизонтали автомат муравья, верхнее число в ячейке таблицы – значение функции приспособленности муравьеда, нижнее – значение функции приспособленности муравья.

Таблица 7. Сравнительная таблица с результатами экспериментов

Муравей / Муравьед	Коэволюция	Случайный	Выращенный на случайном
Коэволюция	2,3 2,9	2,9 0,60	2,5 2,9
Случайный	0,26 3,0	0,21 0,79	0,28 3,2
Выращенный на случайном	2,3 2,8	2,9 0,58	2,7 2,9

Первое место по таблице для автомата муравьеда занимает выращивание на случайных. Коэволюция почти такая же, и особенно важно, что значения совпадают на муравьях со случайными автоматами. Для автоматов муравьев первое место также можно отдать автомату муравья,

выращенному на случайных автоматах муравьеда. Более наглядное представление результатов изображено на рис. 40 и рис. 41.

Важно отметить, что коэволюционное выращивание автоматов муравьеда и муравья не способствовало их замыканию на успешное решение задачи только против тех, с кем они взаимодействовали в процессе выращивания. В [41] также приведена игра, при решении которой с помощью коэволюции более половины выращенных решений сводились к известному для игры равновесию Нэша. Это подтверждает уместность использования коэволюционных методов наравне с другими при решении задач.

Разработанное инструментальное средство позволило провести эксперименты по решению «Задачи о муравьеде и муравьях», результаты визуализировались графиками.

Коэволюцию следует продолжать изучать для улучшения результатов, так как исследования выявили небольшое отличие данного метода от стандартного выращивания с помощью генетических алгоритмов. Серия экспериментов показала возможность применения генетических алгоритмов и коэволюции для выращивания игроков в многошаговых асимметричных гибридных играх с неполной информацией. При этом успешному выращиванию способствовала простота представления особей – автоматы.

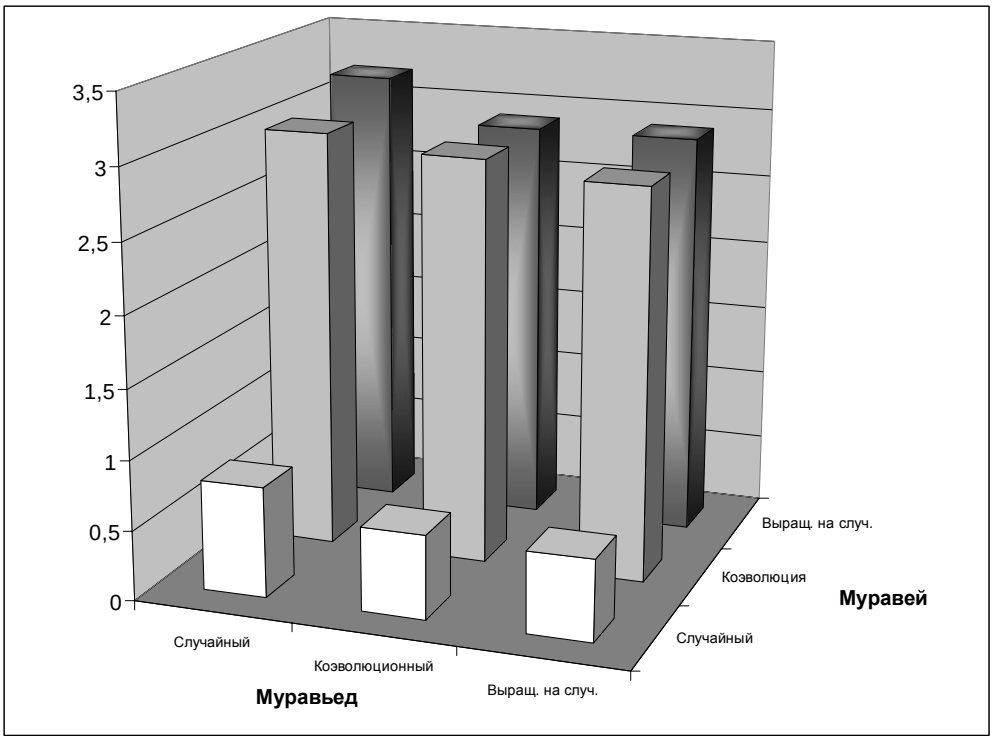


Рис. 40. Сравнение значений функции приспособленности для муравьеда

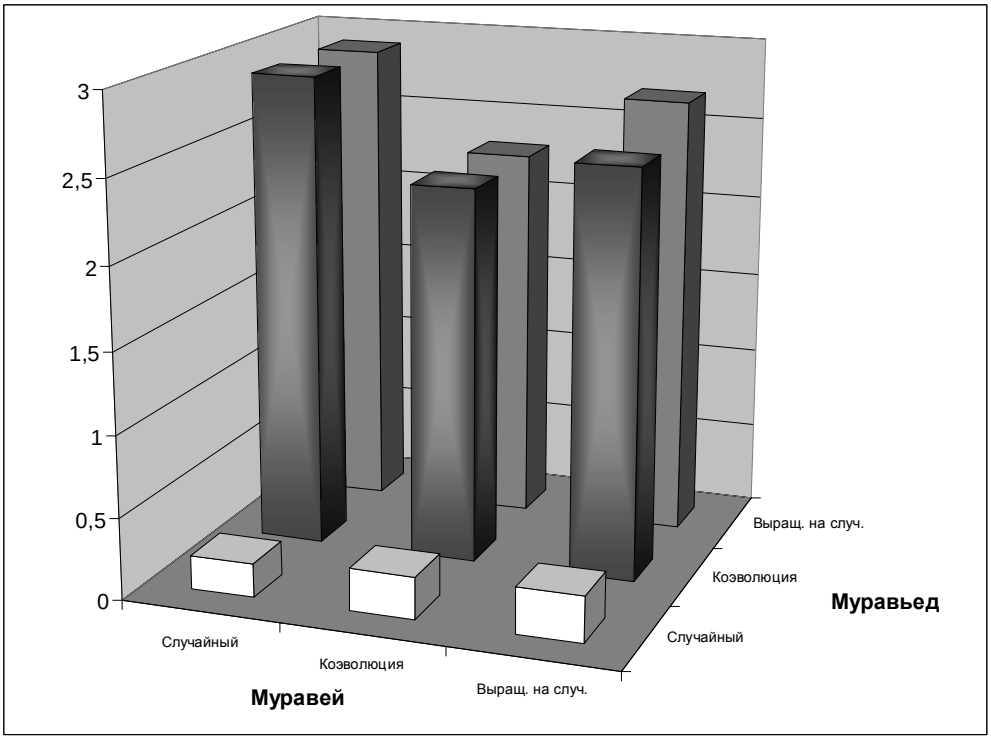


Рис. 41. Сравнение значений функции приспособленности для муравья

В работе был поставлен и решен вопрос о возможности применения генетических алгоритмов и коэволюции для выращивания игроков в многошаговых асимметричных гибридных играх с неполной информацией. В качестве примера была взята «Задача о муравьеде и муравьях» из [22].

Подобные эксперименты открывают перспективу развития генетических алгоритмов и автоматов в отраслях, где необходимо действовать быстро и с большой вероятностью успешно, но при этом разработать качественную эвристику не представляется возможным.

1.3. Разработка, программная реализация и экспериментальное исследование метода применения коэволюции для построения управляющих конечных автоматов, решающих задачи навигации

В настоящем разделе приводится описание метода применения коэволюции для построения управляющих конечных автоматов, решающих задачи навигации.

Задачи навигации возникают в различных отраслях современной науки, включая робототехнику. Так, на практике часто возникают задачи о перемещении робота из одной точки в другую в различном пространстве конфигураций – от простейшего случая передвижения робота на плоскости среди препятствий до поиска оптимальной стратегии перемещения в пространстве робота-манипулятора со множеством степеней свободы. При этом различные виды роботов в различных условиях могут хранить подробную карту местности, на которой ему предстоит работать, обладать лишь частичной информацией о местности или не знать о местности ничего, кроме положения конечной точки. Роботы также могут обладать различными средствами получения информации о внешнем мире – от простейших контактных сенсоров до систем видеонаблюдения, в том числе и в инфракрасном диапазоне. Среда работы может быть неизменной или, напротив, иметь изменяющиеся составляющие или параметры.

Среди всего множества задач навигации обычно рассматривают некоторые крайние случаи, такие как, с одной стороны, обладание полной информацией о местности и, с другой стороны, отсутствие возможности запоминать что-либо, кроме нескольких чисел. В реальности эти случаи комбинируются, и системе управления роботом необходимо использовать доступную ему информацию наиболее эффективным способом, одновременно корректно обрабатывая нештатные ситуации. Сложность требований, предъявляемых к системе управления мобильным роботом, очень высока. Использование автоматного программирования [21] в, по крайней мере, некоторых частях системы управления позволит уменьшить число ошибок, что приведет к повышению надежности системы.

Для более эффективного использования автоматного программирования, однако, необходимо исследовать возможность реализации алгоритмов, решающих задачи навигации, автоматными методами. Принимая в расчет сложность этих алгоритмов, авторы данного исследования считают целесообразным исследовать автоматическое построение автоматов, решающих некоторые задачи навигации.

1.3.1. Задача поиска цели в области с препятствиями

Рассмотрим следующую задачу навигации. Имеется некоторая двумерная область с препятствиями. Препятствия могут быть произвольной формы, однако никакие два препятствия не должны иметь общих точек. Кроме того, любой круг конечного радиуса пересекает конечное число препятствий, и любое препятствие покрывается кругом конечного радиуса.

В области, удовлетворяющей описанным требованиям, дана точка – *цель*. Агент в начальный момент времени находится в произвольной точке области, не принадлежащей ни одному препятствию. Задача агента заключается в том, чтобы добраться до цели или, если она недостижима, сообщить об этом. При этом агент знает свои координаты и координаты цели. Агент не знает, где именно находятся препятствия, но может определить, если ли препятствия в непосредственной с ним близости. Агент обладает $O(1)$ дополнительной памяти, по этой причине он не может запоминать уже посещенную часть области, но может хранить некоторый заранее определенный объем информации (например, координаты некоторого числа точек).

Известны алгоритмы, решающие данную задачу. К ним относятся алгоритмы семейства *Bug*. Общая идея этих алгоритмов такова: пока агент может двигаться по направлению к цели, он это делает. Если агент находит препятствие, которое не дает ему двигаться, он начинает обход этого препятствия. Обход прекращается и продолжается движение к цели, когда выполнено какое-либо условие, специфичное для конкретного алгоритма. В процессе этого обхода алгоритм также может прийти к выводу, что цель недостижима.

Первые алгоритмы семейства *Bug* (*Bug1*, *Bug2*) были впервые описаны в работе [44]. Ниже будут изложены принципы работы этих алгоритмов, а также алгоритмов *Distant Bug* и *Tangent Bug* [41].

1.3.2. Bug1

Идея этого алгоритма – одна из наиболее простых среди всех алгоритмов семества *Bug*. На примере этого алгоритма будет показано, каким образом можно формально описывать и доказывать корректность алгоритмов этого семейства.

Схема алгоритма *Bug1* такова:

1. Двигаться в сторону цели, пока цель не достигнута или не встретится препятствие.
2. Если цель достигнута, то сообщить об этом и остановиться.
3. Иначе встретилось препятствие. В этом случае:

3.1 Запомнить точку, в которой агент находится в данный момент (точки *A* на рисунке).

3.2 Сделать обход препятствия по часовой стрелке.

3.3 В процессе обхода запоминать точку на границе препятствия, от которой кратчайшее расстояние до цели наименьшее (точки *B* на рисунке).

3.4 Когда обход закончен (агент находится в точке *A*):

3.4.1 Если вектор от точки *B* до цели направлен в сторону препятствия, которое обходил агент, то сообщить о невозможности достигнуть цели и остановиться.

3.4.2 Иначе, продолжить обход препятствия в кратчайшем направлении до точки *B*, после чего продолжить выполнение с пункта 1.

Пример пути агента, следующего данному алгоритму, показан на рис. 42.

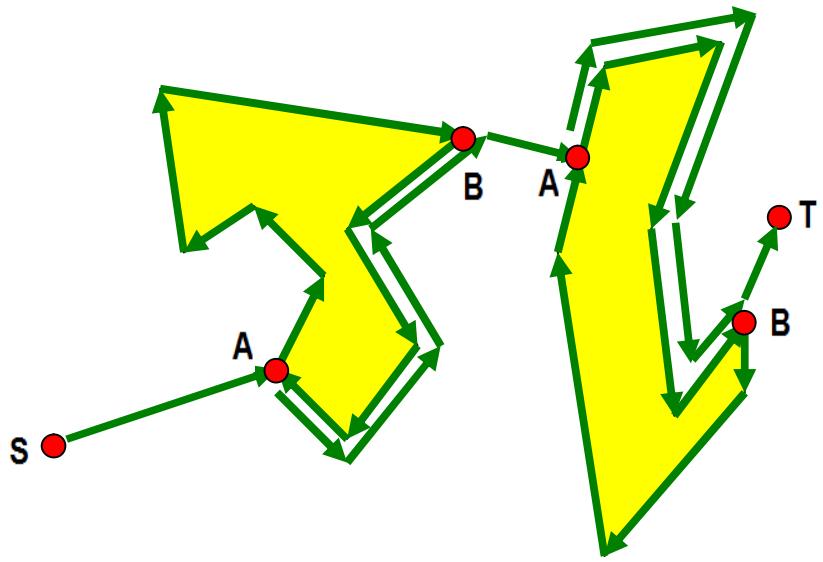


Рис. 42. Иллюстрация принципа работы алгоритма *Bug1*

Ниже изложено доказательство корректности работы алгоритма.

Теорема. Алгоритм *Bug1* всегда достигнет цели, если это возможно. В противном случае закончит работу за конечное время и сообщит, что цель недостижима.

Доказательство. Рассмотрим тривиальный случай. Пусть отрезок, соединяющий агента и цель, не пересекает внутренность ни одного из препятствий. В таком случае агент будет двигаться по этому отрезку согласно алгоритму и достигнет цели, где корректно завершит работу. Следовательно, для этого случая теорема доказана.

Теперь пусть существует препятствие *O*, внутренность которого пересекается упомянутым отрезком. Среди всех таких препятствий рассмотрим ближайшее. Пусть агент впервые обнаружил это препятствие, находясь в точке *A*, а точка *B* является ближайшей к цели точкой границы этого препятствия, достижимой агенту. Рассмотрим два случая:

1. Отрезок из точки *B* до цели не пересекает внутренность *O*. В этом случае $dist(B) < dist(A)$, где под $dist(P)$ понимается расстояние от точки *P* до цели. Таким образом, расстояние от агента до цели строго уменьшилось, кроме того, также уменьшилось число препятствий, достижимых агентом при следовании им алгоритму.
2. Отрезок из точки *B* до цели пересекает внутренность *O*. В этом случае существуют точки препятствия *O*, находящиеся ближе к цели, чем любая точка достижимой агенту границы этого препятствия. Следовательно, можно построить гомеоморфную кольцу кривую *C*, лежащую целиком внутри препятствия *O*, так чтобы цель оказалась внутри области, ограниченной *C*. Это означает, что цель недостижима агентом, о чем он и сообщает, согласно алгоритму.

Совмещая полученные результаты, получаем доказательство теоремы по индукции. Базой индукции для различных начальных случаев служат все утверждения, кроме пункта 1. Шагом индукции служит утверждение 1. Следовательно, **теорема доказана**.

Аналогичным образом можно доказать и следующее утверждение: длина пути агента в процессе поиска цели не превосходит $1{,}5\sum_{k=1}^{N} P_i + dist(S,T)$, где P_i – периметр препятствия с номером i $1 \le i \le N$, а $dist(S,T)$ – расстояние от начальной точки расположения агента до цели.

1.3.3. Bug2

Алгоритм *Bug1* удовлетворяет требованиям к решению исследуемой задачи навигации, однако во многих случаях он работает слишком долго. Действительно, однажды наткнувшись на препятствие, агент пройдет по границе препятствия путь длиной от одного до полутора периметров. Зачастую, однако, можно обойтись гораздо меньшими затратами.

Алгоритм *Bug2* запоминает не только координаты цели *T*, но и координаты своей стартовой точки *S*. От одного препятствия к другому перемещение происходит только вдоль отрезка *ST*, и только в сторону цели. Как и в случае алгоритма *Bug1*, изначально агент перемещается к цели по прямой, пока не найдет препятствие или не достигнет цели. В случае препятствия его обход производится следующим образом:

1. Запоминается точка *A*, в которой агент встретил препятствие.
2. Производится обход по часовой стрелке.
3. Если в процессе обхода найдена точка *B*, такая что $B \in ST$, $dist(B) < dist(A)$ и в сторону цели нет препятствия, то происходит «отрыв от препятствия» и продолжается движение в сторону цели.
4. Если агент пришел вновь в точку *A*, то цель считается недостижимой.

На рис. 43 приведен пример пути агента, следующего алгоритму *Bug2*.

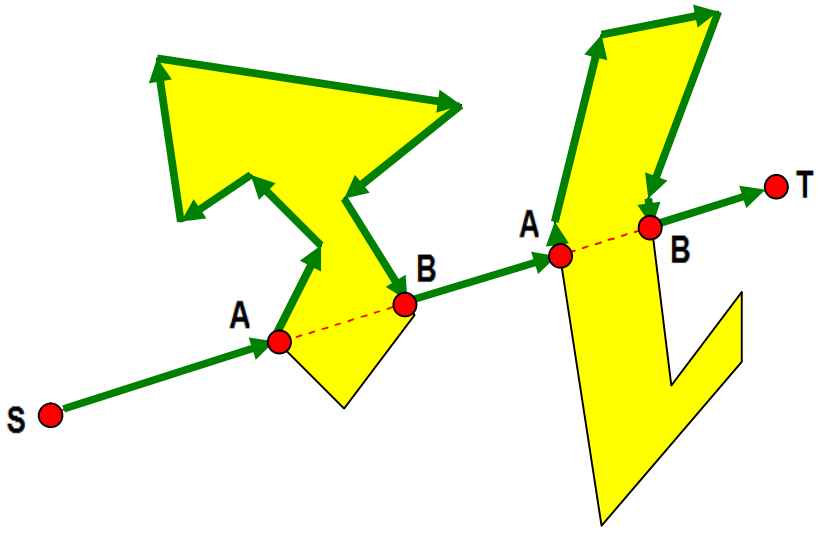


Рис. 43. Иллюстрация принципа работы алгоритма *Bug2*

Корректность этого алгоритма, как и многих других алгоритмов семейства *Bug*, доказывается по той же схеме, что и корректность алгоритма *Bug1*. Однако, данный алгоритм может обходить одно

препятствие несколько раз, как показано на рис. 44. Верхняя граница на длину пройденного пути в случае данного алгоритма определяется как $0{,}5\sum_{k=1}^{N} P_i N_i + dist(S,T)$, где N_i – число точек пересечения прямой *ST* и *i*-того препятствия.

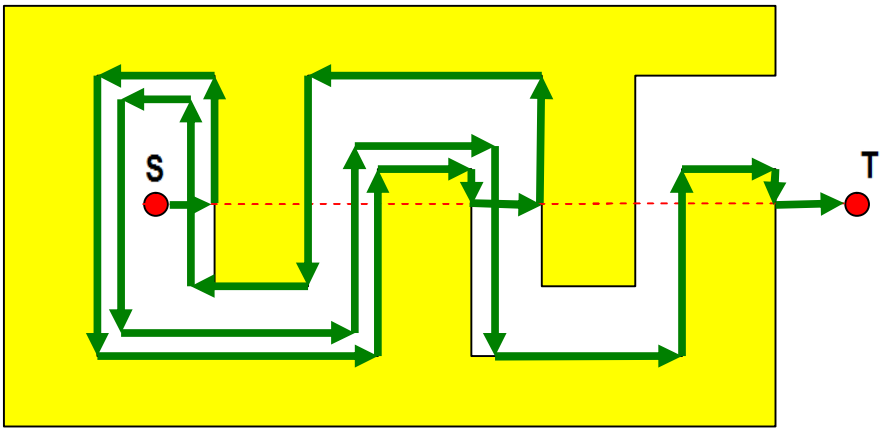


Рис. 44. Пример неэффективной работы алгоритма *Bug2*

Стоит, однако, отметить, что зачастую при следовании данному алгоритму агент обходит в среднем половину периметра тех препятствий, которые пересекают отрезок *ST*, что примерно в три раза меньше, чем в случае алгоритма *Bug1*. В частности, если и начальная, и конечная точки находятся вне выпуклой оболочки некоторого препятствия, то агент никогда не посетит одну и ту же точку границы препятствия дважды, а «в среднем» обойдет половину периметра препятствия. Если это условие верно для любого препятствия, то длину пути можно оценить как $\sum_{k=1}^{N} P_i + dist(S,T)$. В работе [44] показано, что для любого алгоритма, решающего данную задачу навигации, существует такой набор препятствий, на котором длина пути агента оценивается снизу именно этой формулой, что демонстрирует эффективность алгоритма *Bug2* на картах описанного вида.

1.3.4. Distant Bug

Алгоритм *Distant Bug* во многом похож на алгоритм *Bug2*, однако он не использует проверку на то, находится ли текущая точка на данном отрезке. «Отрыв от препятствия», имеющий место в схеме алгоритма *Bug2*, происходит в том случае, когда расстояние от текущей точки *B* удовлетворяет неравенству $dist(B) < dist(A)$, где *A* – «точка входа» в препятствие, и когда от точки *B* возможно продолжить движение в направлении цели (рис. 45). Алгоритм остается корректным, однако длительность обхода препятствия до отрыва от него может стать существенно меньше. Тем не менее, примеры, подобные рис. 44, остаются в силе и для этого алгоритма.

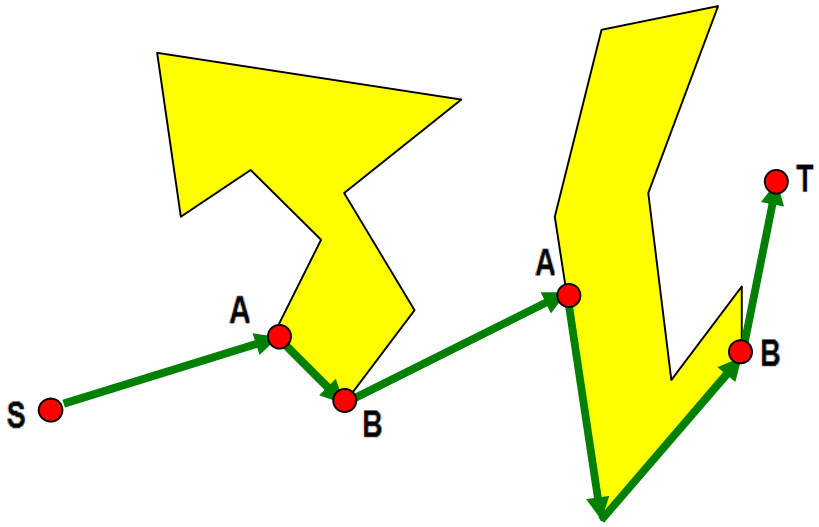


Рис. 45. Иллюстрация принципа работы алгоритма *Distant Bug*

1.3.5. Tangent Bug

Алгоритмы семейства *Bug* могут успешно обобщаться на агентов с более сложными сенсорами. Например, агент может не только «чувствовать» препятствия в непосредственной близости от себя, но и «видеть» препятствия на некотором отдалении как «недоступные отрезки» (рис. 46). В этом случае можно начать обход препятствия заранее, тем самым «срезая углы» и сокращая путь.

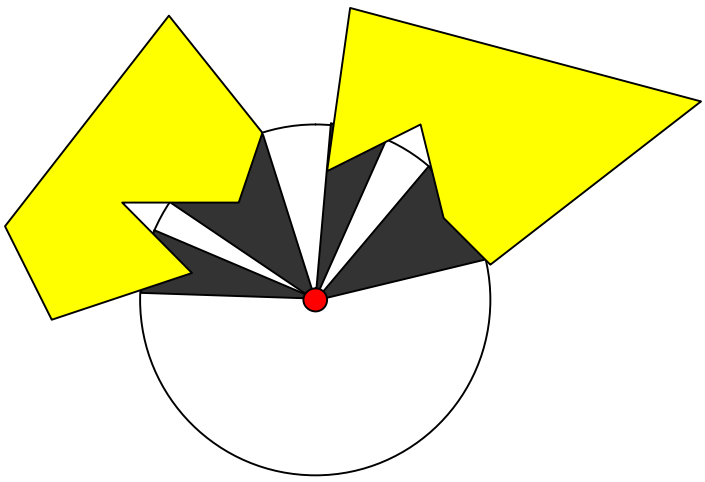


Рис. 46. Пример более сложного сенсора

Первое обобщение алгоритмов семейства Bug на случай сложных сенсоров было проведено в работе [47]. Полученный результат был существенно улучшен в работе [41], в которой был описан

алгоритм *Tangent Bug*. Этот алгоритм строит на видимых участках препятствий граф касательных [44] – упрощенный вариант графа видимости – и уже с использованием этой информации прокладывает локально оптимальный путь. Пример траекторий агента с различными настройками алгоритма приведен на рис. 47.

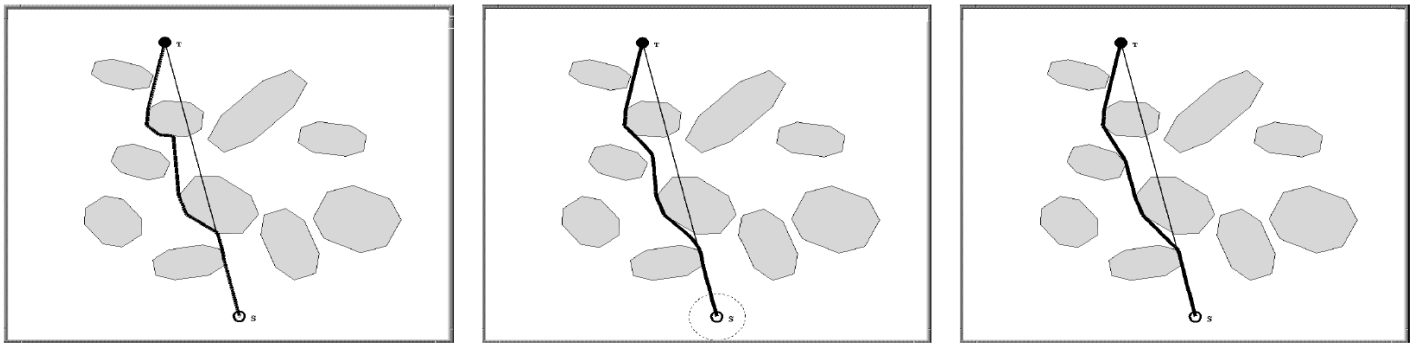


Рис. 47. Траектории алгоритма *Tangent Bug*: (а) – дистанция видимости 0, (б) – дистанция видимости 50, (в) – неограниченная видимость (иллюстрация из оригинальной статьи)

1.3.6. Постановка задачи

Несмотря на то, что описанные алгоритмы решают задачу поиска цели в области с препятствиями, представляет интерес *автоматическое построение конечного автомата*, который также решает данную задачу. Построенный конечный автомат может быть использован в качестве части системы управления робота, перемещающегося в области с препятствиями, если эта система спроектирована с использованием парадигмы автоматного программирования [21]. Кроме того, данная задача интересна с точки зрения оценки применимости эволюционных методов для построения конечных автоматов для решения задач навигации.

Упростим ранее сформулированную задачу поиска цели в области с препятствиями. Определим область как бесконечное клетчатое поле. Некоторые клетки этой области заняты препятствиями, причем таких клеток конечное число. Цель находится в одной из клеток, не занятых препятствием. Агент также занимает одну клетку. Агент может перемещаться только в клетки, смежные по стороне с текущей и не занятые препятствиями.

Агент знает свои координаты, а также координаты цели. Будем также считать, что у агента есть ориентация в пространстве и, как следствие, выделенное направление «вперед».

Агент управляется конечным автоматом. Конечный автомат принимает ряд входных воздействий и формирует на их основе ряд выходных воздействий. Также у агента есть $O(1)$ дополнительной памяти – помимо собственных координат, координат цели и информации об ориентации в пространстве, он может хранить координаты еще одной точки и число, определяющее ориентацию автомата. Доступ к координатам агента, цели и сохраненной точки осуществляется с помощью входных и выходных воздействий автомата.

1.3.7. Описание воздействий конечного автомата

Опишем входные воздействия конечного автомата.

- `x1` – верно ли, что в клетке, находящейся впереди агента, нет препятствия.
- `x2` – верно ли, что если агент перейдет на одну клетку вперед, то он станет ближе к цели (без учета препятствий).

- `x3` – верно ли, что агент находится у цели.
- `x4` – верно ли, что агент находится в сохраненной точке с сохраненной ориентацией.
- `x5` – верно ли, что агент находится ближе к цели, чем сохраненная точка.

Опишем выходные воздействия конечного автомата.

- `z1` – ничего не делать.
- `z2` – передвинуться на одну клетку вперед.
- `z3` – повернуться по часовой стрелке.
- `z4` – повернуться против часовой стрелки.
- `z5` – сохранить текущую точку и ориентацию.
- `z6` – остановиться и доложить, что агент достиг цели.
- `z7` – остановиться и доложить, что цель недостижима.

Были выбраны именно данные воздействия, так как, с одной стороны, они не оперируют с известными агенту данными на низком уровне (иначе воздействий было бы слишком много, построение автомата заняло бы чрезмерно много времени, а получившийся автомат был бы слишком сложен для восприятия), а с другой стороны, не являются слишком высокоуровневыми (что привело бы к усложнению неавтоматной части алгоритма, а также к уменьшению числа классов алгоритмов, выражаемых с помощью этих воздействий).

Прекращение работы автомата осуществляется, как было описано выше, при выполнении специфических (терминальных) выходных воздействий. Также возможен иной способ: выделяются терминальные состояния, при переходе в которые автомат останавливается. Преимущества выбранного способа заключаются в том, что при его применении семантика состояний автомата не фиксируется, чем облегчается работа алгоритма поиска автомата. Преобразование же автомата с терминальными выходными воздействиями в автомат с терминальными состояниями и обратно легко осуществляется как вручную, так и автоматически.

1.3.8. Описание генетического алгоритма с применением коэволюции

В данном исследовании для автоматического построения конечного автомата применяется генетический алгоритм [36]. Функция приспособленности каждого конкретного автомата оценивается исходя из того, как этот автомат проходит некоторый набор полей – *тестов*. В свою очередь, этот набор полей также эволюционирует параллельно с эволюцией популяции автоматов, таким образом, применяется коэволюция [36]. Цель применения коэволюции – обеспечить отсутствие сходимости генетического алгоритма к неверным решениям вследствие неполноты набора тестов, а также создание благоприятных условий для работы генетического алгоритма.

Общая схема генетического алгоритма такова. На основе имеющегося поколения создается множество потомков. Потомки создаются парами. Выбор родителей для каждой пары потомков осуществляется оператором селекции. Потомки создаются в результате действия оператора скрещивания на родителей, при этом родители остаются неизменными. Далее, для каждой из имеющихся особей создается ее измененная копия, полученная с помощью оператора мутации. Из всех имеющися особей в следующее поколение проходят лучшие особи.

Каждая особь (либо сгенерированная случайно, либо являющаяся результатом действия оператора) проходит процедуру упрощения. Эта процедура исключает доказуемо неиспользуемые ветви деревьев решений, тем самым увеличивая долю полезного кода в дереве решений и, гипотетически, ускоряя процесс нахождения решения задачи.

После обработки поколения в случае необходимости к набору тестов применятся оператор мутации тестов. Это происходит в двух случаях – когда лучшая особь в поколении прошла все тесты и когда в течение длительного времени значение приспособленности лучшей из особей не изменяется. В первом из этих случаев цель оператора мутации тестов – получить тест, который не

проходит ни одна текущая особь, если это невозможно, то считается, что генетический алгоритм достиг своей цели. Во втором из этих случаев оператор мутации тестов предназначен для того, чтобы сдвинуть генетический алгоритм с «мертвой точки».

В тех случаях, когда текущий набор тестов пройден, перед тем, как генерировать новый набор, особи, проходящие все тесты, дополнительно некоторое время оптимизируются по более жестким критериям. Это делается для того, чтобы «обобщить» алгоритмы, инкапсулируемые этими особями, путем уменьшения числа активных состояний, устранения излишних операций или упрощения логики алгоритмов.

1.3.8.1. Особь генетического алгоритма

В качестве представления состояния автомата используется дерево решений [7]. По сравнению с такими способами представления состояния, как полные таблицы переходов [14], деревья решений требуют меньше памяти для их хранения, а по сравнению с сокращенными таблицами переходов [22] деревья решения, при соразмеримой потребляемой памяти, проще в реализации. Кроме того, многие из операций над деревьями решения точнее моделируют модификацию способов рассуждения человека, чем операции над таблицами переходов.

В листьях деревьев решений находятся пары чисел – номер выходного воздействия и номер состояния, в которое нужно перейти. В каждом узле находится номер входного воздействия, на основе которого принимается очередное решение.

Автомат полностью описывается массивом деревьев решений, каждое из которых описывает соответствующее состояние. Начальным состоянием автомата является состояние с номером ноль.

1.3.8.2. Оператор селекции

В качестве оператора селекции используется турнирный отбор. Для того, чтобы выбрать особь, из имеющихся особей выбирается восемь особей. Особи разбиваются на пары, из каждой пары с вероятностью 0,9 победителем выходит более приспособленная, а с вероятностью 0,1 – менее приспособленная особь. Победители вновь разбиваются на пары, между ними вновь происходит отбор и так далее до тех пор, пока не останется ровно одна особь, которая и считается выбранной.

1.3.8.3. Операторы скрещивания

В настоящем исследовании используются два оператора скрещивания.

Первый из этих операторов для каждого из номеров состояний с вероятностью 0,2 меняет местами автоматы потомков (рис. 48).

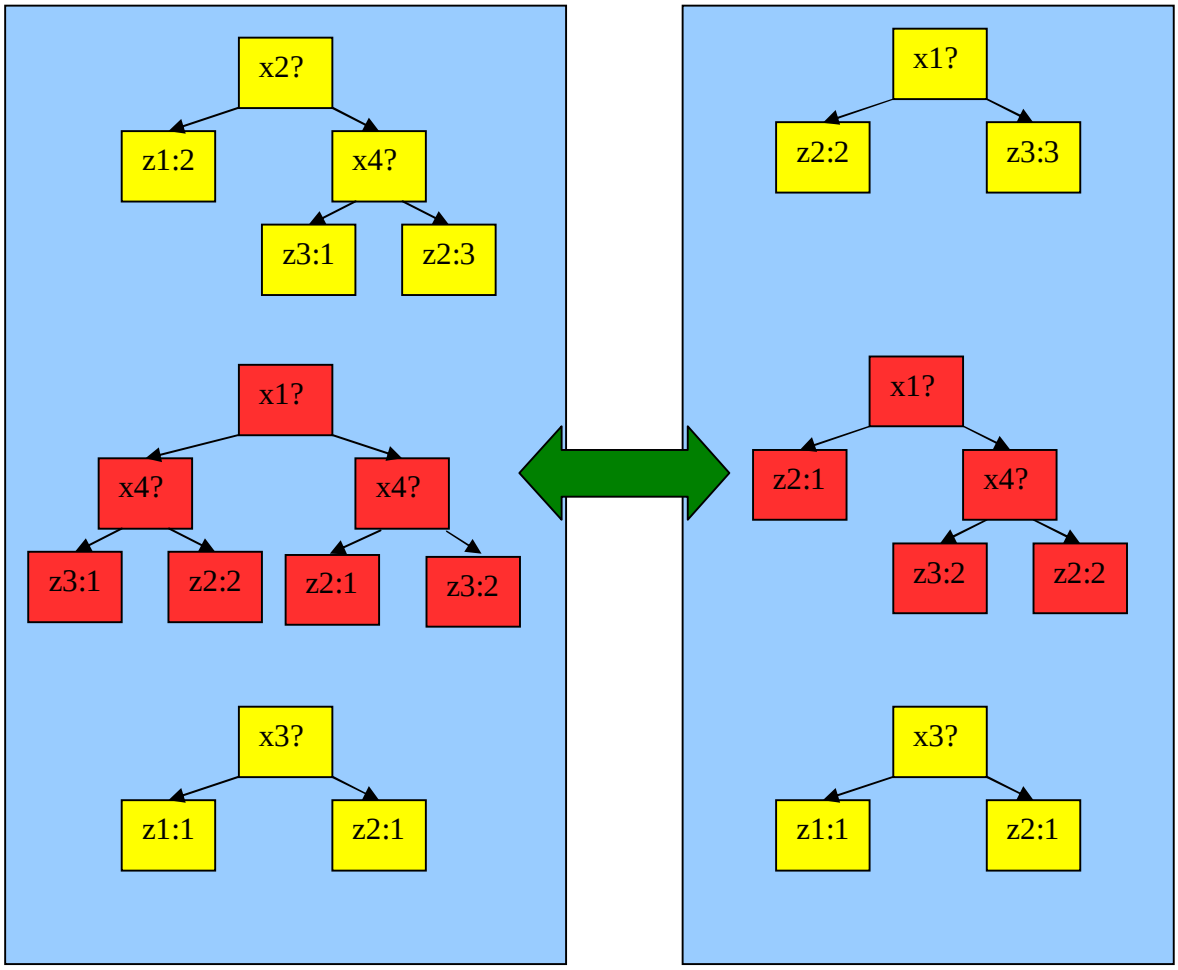


Рис. 48. Первый оператор скрещивания

Второй оператор для каждого из номеров состояний производит обмен поддеревьями в соответствующих деревьях (рис. 49).

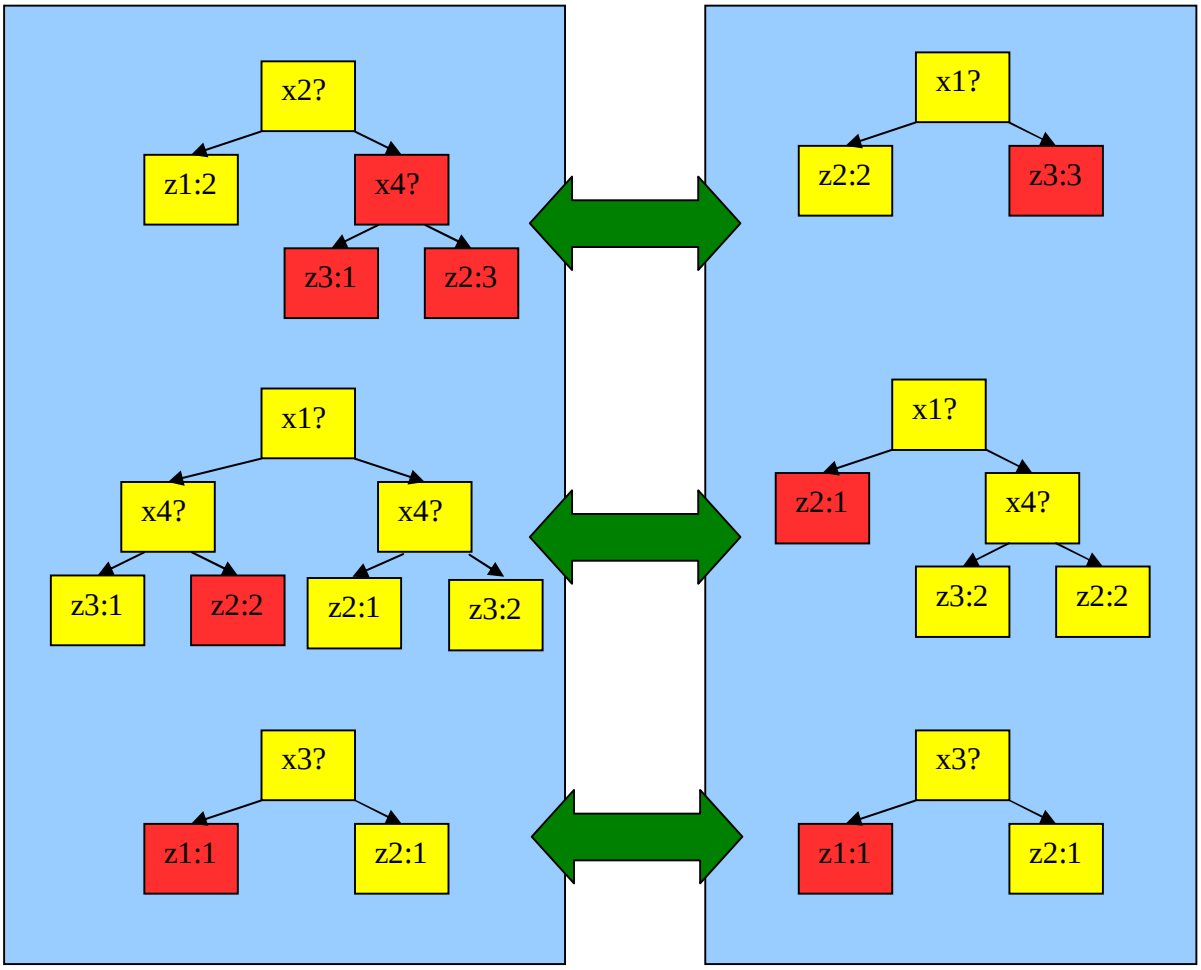


Рис. 49. Второй оператор скрещивания

1.3.8.4. Оператор мутации

Используемый в данной работе оператор мутации в каждом состоянии с вероятностью 0,1 заменяет случайно выбранное поддерево дерева решения на случайно сгенерированное дерево той же высоты (рис. 50).

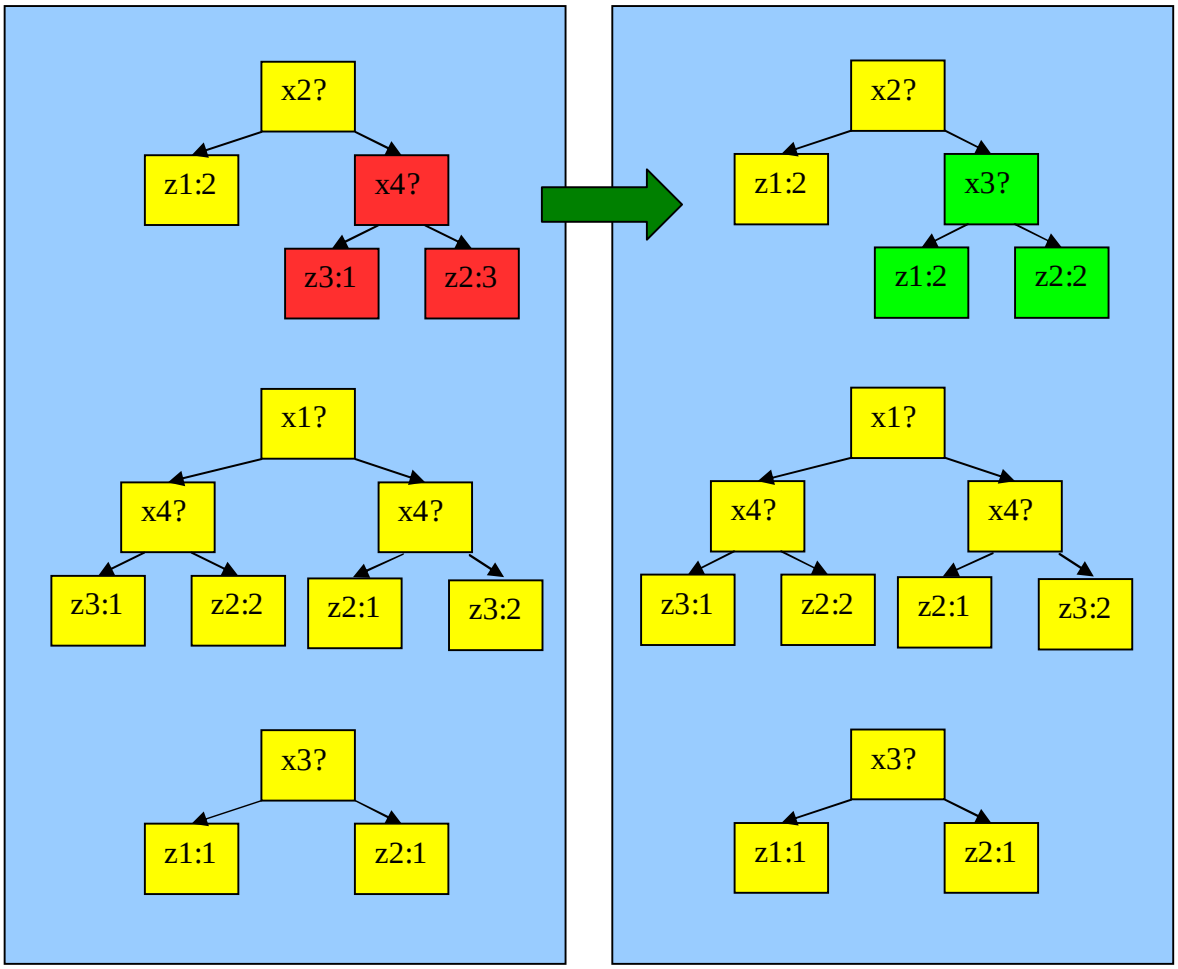


Рис. 50. Оператор мутации

1.3.8.5. Процедура упрощения

В качестве представления функции переходов автомата, дерево решений, по сравнению с полными и сокращенными таблицами переходов, обладают одним существенным недостатком. При генерации деревьев решений случайным образом или при действии операторов скрещивания или мутации возможно формирование поддеревьев, которые никогда не участвуют в работе дерева решений. Пример такого дерева приведен на рис. 51. В правом поддереве приведенного дерева предикат x3 имеет значение «ложь», поэтому поддерево A не принимает участия в формировании логики и может быть безопасным образом исключено из дерева.

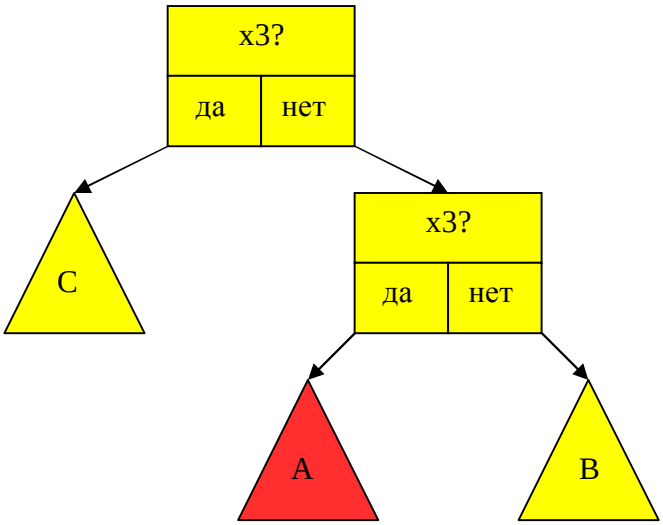


Рис. 51. Пример неиспользуемого поддерева

Более тонкий пример приведен на рис. 52. В данном примере поддерево `B` может быть безопасно устранено, потому что, согласно семантике предикатов, из предиката `x4` следует отрицание предиката `x5`.

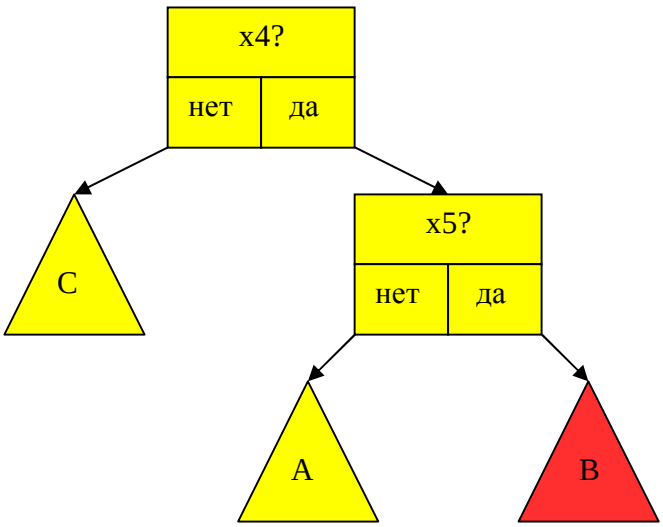


Рис. 52. Более сложный пример неиспользуемого поддерева

Для эффективного устранения неиспользуемых поддеревьев в деревьях решений на основе используемых предикатов достаточно использовать два метода, являющихся обобщениями продемонстрированных примеров:

1. Упрощение поддеревьев на основе тавтологий.

2. Упрощение поддеревьев на основе набора импликаций.

Для формализации описанных методов введем следующие обозначения. Введем на деревьях решений следующие функции:

- $isLeaf(T)$ – является ли дерево T состоящим из одной вершины (листа);
- $cond(T)$ – возвращает предикат – условие ветвления в корне дерева (определено для деревьев, состоящих более чем из одной вершины);
- $trueBranch(T)$ – поддерево дерева T, выбираемое при $cond(T)$ = «истина»;
- $falseBranch(T)$ – поддерево дерева T, выбираемое при $cond(T)$ = «ложь».

Определим $< C, T, F >$ как дерево решений, в корне которого записано условие ветвления C, T – это поддерево, выбираемое при C = «истина», F – это поддерево, выбираемое при C = «ложь».

Введем также функции предикатов, возвращающую по предикату одно из трех значений: «истина», «ложь» и «неопределенность». Пусть функция F_0 возвращает «неопределенность» для всех возможных предикатов. Также введем операцию «доопределения» функции, действующую следующим образом:

$$F \oplus (x_i = V) = \begin{cases} V, x = x_i \\ F(x), x \neq x_i \end{cases}$$

С помощью описанных терминов можно формализовать упрощение поддеревьев на основе тавтологий. А именно, определим функцию $SimplifyT$, принимающую дерево решений и функцию предикатов и возвращающую также дерево решений, следующим образом:

$$SimplifyT(T, F) = \begin{cases} T, isLeaf(T) = истина \\ SimplifyT(trueBranch(T), F), F(cond(T)) = истина \\ SimplifyT(falseBranch(T), F), F(cond(T)) = ложь \\ < cond(T), NT, NF > иначе \end{cases}$$

$$NT = SimplifyT(trueBranch(T), F \oplus (cond(T) = истина))$$

$$NF = SimplifyT(falseBranch(T), F \oplus (cond(T) = ложь))$$

При помощи данной функции операцию упрощения на основе тавтологий можно определить так: $T = SimplifyT(T, F_0)$.

Для формализации упрощения на основе импликаций необходимо на основе семантики предикатов описать множество импликаций на предикатах вида $x_i \to x_j$, $x_i \to \neg x_j$ или $\neg x_i \to x_j$. Заметим, что из $x_i \to x_j$ следует $\neg x_j \to \neg x_i$, из $x_i \to \neg x_j$ следует $x_j \to \neg x_i$, а из $\neg x_i \to x_j$ следует $\neg x_j \to x_i$. В множестве импликаций не должно быть семейств утверждений, на основе которых можно построить противоречие, например, $A \to B$ и $A \to \neg B$. Для оптимальности использования это множество необходимо замкнуть, то есть, сделать все возможные выводы указанного вида и включить их в это множество.

Пусть на основе семантики предикатов построено множество импликаций S. Определим функцию $SimplifyI(T, F, S)$, где T – дерево решений, F – функция предикатов, S – множество импликаций следующим образом:

$$SimplifyI(T,F,S)=\begin{cases} T, isLeaf(T)=истина \\ SimplifyI(trueBranch(T),F,S)\ если\ верно\ условие\ У1 \\ SimplifyI(\ falseBranch(T),F,S)\ если\ верно\ условие\ У2 \\ <cond(T),NT,NF>\ иначе \end{cases}$$

$У1 = F(cond(T)) = истина$

$\quad или\ \exists U : F(U) = истина\ и\ (U \to cond(T)) \in S$

$\quad или\ \exists U : F(U) = ложь\ и\ (\neg U \to cond(T)) \in S$

$У2 = F(cond(T)) = ложь$

$\quad или\ \exists U : F(U) = истина\ и\ (U \to \neg cond(T)) \in S$

$\quad или\ \exists U : F(U) = ложь\ и\ (\neg U \to \neg cond(T)) \in S$

$NT = SimplifyI(trueBranch(T), F \oplus (cond(T) = истина), S)$

$NF = SimplifyI(\ falseBranch(T), F \oplus (cond(T) = ложь), S)$

Данная функция применяет оба метода упрощения деревьев решений – как на основе тавтологий, так и на основе импликаций. Собственно упрощение дерева с помощью этой функции определяется так: $T = SimplifyI(T, F_0, S)$. В результате анализа имеющихся имеющихся предикатов было построено множество *S*, являющееся замыканием следующих импликаций:

- $x4 \to \neg x5$
- $x2 \to \neg x3$.

Отметим, что упрощение на основе тавтологий можно реализовать с помощью упрощения на основе импликаций – для этого достаточно добавить в множество *S* утверждения вида $x_i \to x_i$. Однако трудоемкость метода упрощения на основе тавтологии, при прочих равных, пропорциональна мощности множества *S*, поэтому реализация упрощения на основе тавтологий отдельным способом приводит к выигрышу во времени работы.

1.3.8.6. Функция приспособленности

Как отмечено в работе [36], выбор функции приспособленности является критическим для обеспечения эффективности генетического алгоритма. От вида функции приспособленности зависит то, насколько быстро и к каким экстремумам будет сходится генетический алгоритм в пространстве решений.

Так как во многих случаях вычисление функции приспособленности является наиболее трудоемким процессом в ходе работы генетического алгоритма, при разработке функции приспособленности требуется также по возможности снижать затраты процессорного времени на ее вычисление, в частности, избегать повторного вычисления одних и тех же величин.

Функция приспособленности должна принимать достаточно большое число различных значений, чтобы процесс улучшения значения приспособленности особей протекал более плавно, без значительных периодов стагнации и резких скачков. В частности, при решении поставленной задачи необходимо сравнивать по приспособленности не только автоматы, проходящие тесты, но и автоматы, их не проходящие, чтобы выделять из них лучшие и давать им развиваться дальше.

Рассмотрим прохождение одной особью (автоматом) одного теста (поля). Поведение особи на тесте характеризуется качественными и количественными характеристиками. Качественные характеристики определяют то, корректно ли пройден тест, а если некорректно, то какого рода ошибку совершил автомат. Количественные характеристики предназначены для сравнения особей, имеющих одинаковые качественные характеристики, и выявления лучшей из них.

Качественной характеристикой поведения особи является *тип результата*. Он может быть одним из следующих:

- Тест пройден корректно.
- Произошло нарушение правил (автомат «врезался» в препятствие).
- Дан неверный ответ (воздействие `z6` в том случае, когда агент не находится у цели, или воздействие `z7` в том случае, когда цель достижима).
- Автомат не останавливается. Данный результат, в свою очередь, может быть дополнительно классифицирован как:
  - Зацикливание. Граф переходов агента в пространстве состояний вошел в цикл.
  - Убегание. Агент удаляется от цели на все большее расстояние.

Заметим, что на типах результата, вообще говоря, невозможно ввести отношение полного порядка достаточно «разумным» образом. Однозначным образом можно только утверждать, что пройти тест корректно лучше, чем пройти тест некорректно. Сравнивать же некорректные прохождения теста можно различными способами, но ни один из них не является полностью верным. Так, автомат, который крутится на месте старта, определенно «хуже» автомата, который доходит до цели и «забывает» об этом доложить, хотя оба они имеют одинаковый тип результата – зацикливание. Автомат же, который в целом проходит тесты верно, но, оказавшись в «тупике», выдает сообщение о том, что цель недостижима, значительно лучше первого из «зациклившихся» автоматов, но хуже второго.

Для различения автоматов, обладающих одинаковыми или близкими по смыслу типами результата, предназначны две количественные характеристики: *минимальное достигнутое расстояние до цели* и *суммарное расстояние до цели по пути следования*. Из общих соображений следует, что если из двух автоматов первый смог приблизиться к цели на меньшее расстояние, чем второй, то первый почти наверное лучше второго. При равных же минимальных расстояниях чем меньше сумма расстояний до цели по пути следования, тем лучше считается автомат.

Описанная функция приспособленности предназначена для того, чтобы оценивать особей, если тестирование проводится на одном тесте. При использовании многих тестов уже нельзя однозначно сказать, лучше ли один автомат, чем другой, поскольку на разных тестах преимущество может быть у разных автоматов. Данная проблема носит общий характер и, в той или иной степени, относится ко всем задачам многокритериальной оптимизации.

Для дальнейшего описания функции приспособленности целесообразно выделить в описываемом генетическом алгоритме две стадии. Первая из этих стадий имеет место, когда лучшая из особей текущего поколения не проходит все тесты, и цель генетического алгоритма – добиться прохождения всех тестов. Вторая из этих стадий наступает после того, как все тесты пройдены лучшей особью, и продолжается некоторое время. По истечении этого времени к множеству тестов применяется оператор мутации тестов, и процесс снова переходит к первой стадии.

В первой стадии необходимо добиться того, чтобы все тесты были пройдены, поэтому функция приспособленности предназначена для того, чтобы давать преимущество особям, проходящим тесты лучше. Будем считать, что в данный момент времени имеется N тестов, и они упорядочены по времени их появления. Пусть автомат A_1 прошел первые k_1 тестов без ошибок и допустил ошибку на тесте с номером k_1+1 (если все тесты пройдены без ошибок, то $k_1 = N$), а автомат A_2 прошел первые k_2 тестов без ошибок и допустил ошибку на тесте с номером k_2+1. В этом случае функция приспособленности устроена следующим образом:

- если $k_1 < k_2$, то A_2 лучше A_1;
- если $k_1 > k_2$, то A_1 лучше A_2;

- если $k_1 = k_2$, то автоматы сравниваются по функции приспособленности на тесте $k_1 + 1$ (если $k_1 = N$, то автоматы считаются равными).

Во второй стадии существует по крайней мере одна особь, которая проходит все имеющиеся тесты. Однако, почти наверняка способ их прохождения неоптимален, например, особь делает много «лишних» движений. Чтобы устранить этот недостаток, требуется оптимизировать способ прохождения этих тестов. Можно сказать, что при попытке пройти новые тесты в дополнение к предыдущим особь «изобретает» новую стратегию прохождения, а с помощью оптимизации существующих стратегий происходит «обобщение» новых и старых стратегий. Будем считать, что в данный момент времени имеется N тестов, и они упорядочены по времени их появления. Пусть автомат A_1 прошел первые k_1 тестов без ошибок и допустил ошибку на тесте с номером $k_1 + 1$ (если все тесты пройдены без ошибок, то $k_1 = N$), а автомат A_2 прошел первые k_2 тестов без ошибок и допустил ошибку на тесте с номером $k_2 + 1$. В этом случае функция приспособленности устроена следующим образом:

- если $k_1 < k_2$, то A_2 лучше A_1;
- если $k_1 > k_2$, то A_1 лучше A_2;
- если $k_1 = k_2$, то для всех пройденных тестов параметры «минимальное достигнутое расстояние до цели» и «суммарное расстояние до цели по пути следования» складываются для обоих автоматов. Если первые из параметров у автоматов не равны, то лучшим считается тот автомат, у которого первый параметр меньше. Иначе лучше тот автомат, у которого меньше второй параметр.

1.3.8.7. Оператор мутации тестов и коэволюция

Как было обосновано ранее, оценка особи на фиксированном наборе тестов может привести к тому, что особь, проходящая все тесты, может не являться корректным решением задачи. С другой стороны, если ни одна особь долгое время не проходит все тесты, то такая система тестов может быть настолько «трудной», что ландшафт функции приспособленности является слишком крутым и изломанным для большинства алгоритмов направленного поиска.

Для решения этой проблемы естественным кажется введение некоторого способа изменения набора тестов. Тесты должны усложняться вместе с улучшением среднего или лучшего решения, и, возможно, упрощаться, если решения ухудшаются или долгое время остаются неизменными по качеству.

В области генетических алгоритмов реализацией этой идеи является коэволюция. Впервые коэволюция была применена в работе [36], где с ее помощью была частично решена проблема преждевременной сходимости генетического алгоритма к локальному, но не глобальному оптимуму. Сущность коэволюции в этой работе заключается в том, что функция приспособленности определяется не только для особи (на основе тестов), но и для теста (на основе особей), и операторы генетических алгоритмов введены не только для особей, но и для тестов (возможно, эти операторы различны или действуют независимо друг от друга). Тем самым, если особи генетического алгоритма стремятся к некоторому локальному оптимуму, не являющемуся глобальным, то тесты, «вытесняющие» алгоритм из этого оптимума, получают большее преимущество, и этот оптимум с течением времени покидается.

В данном исследовании была проведена попытка применить эту идею без значительных изменений. Однако, на практике выяснилось, что тесты за сравнительно небольшое число поколений чрезмерно усложняются, тем самым делая значения функции приспособленности особей неразличимо низкими и останавливая их развитие. Искусственное замедление генетического алгоритма, выращивающего тесты, привело к тому, что вначале особи развиваются до некоторого предела, что

провоцирует чрезмерное усложнение тестов и либо повторения ранее описанного сценария, либо стагнацию особей на достигнутом этапе развития.

По этой причине было решено упростить алгоритм, изменяющий набор тестов, с тем, чтобы развитие тестов не доминировало над развитием особей, и дополнить его возможностью упрощения тестов при прекращении развития поколения особей.

В данном исследовании тесты упорядочены в список. Функции приспособленности, описанные ранее, являются «префиксными» – если одна особь проходит корректно больший префикс тестов, чем другая, то она является лучшей. Следовательно, манипуляция сложностью тестов может происходить путем добавления, удаления или изменения последнего теста.

В начале работы генетического алгоритма набор тестов состоит из одного теста, сгенерированного случайным образом. В ситуациях, когда текущий набор тестов проходится хотя бы одной особью, и проведена предварительная оптимизация таких особей, на основе одного из уже существующих тестов случайными мутациями генерируется тест, который не проходится ни одной из имеющихся особей. Этот тест добавляется в конец текущего списка тестов, и работа генетического алгоритма продолжается.

Если за некоторое, заранее фиксированное число поколений ни одна особь не сумела пройти имеющиеся тесты, то делается вывод о том, что последний из этих тестов является «слишком сложным». Заметим, что именно этот вывод мы делаем по причине того, что в генетическом алгоритме на особях, как было изложено ранее, используется стратегия элитизма. Из этого следует, что по крайней мере лучшая особь проходит все тесты, за исключением последнего. Следовательно, чтобы продолжить работу генетического алгоритма на более простых тестовых данных, необходимо сгенерировать способом, описанным ранее, новый тест, предназначенный для замены последнего из имеющихся тестов. Мы можем добиться того, чтобы новый тест был проще того, который он заменяет, путем измерения функции приспособленности и поддержания очевидного неравенства. Однако, можно обойтись и без этого, оставив единственным условием то, чтобы ни одна из имеющихся особей не проходила новый тест.

Данная стратегия коэволюции позволяет в некотором смысле обеспечить полноту тестов, в то же время избегая долговременных периодов стагнации генетического алгоритма.

1.3.9. Полученные результаты

С помощью описанного генетического алгоритма и метода генерации тестов, при котором цель всегда была достижима, был сгенерирован конечный автомат всего из трех состояний, решающий задачу при условии, что цель всегда достижима. Его граф переходов изображен на рис. 53.

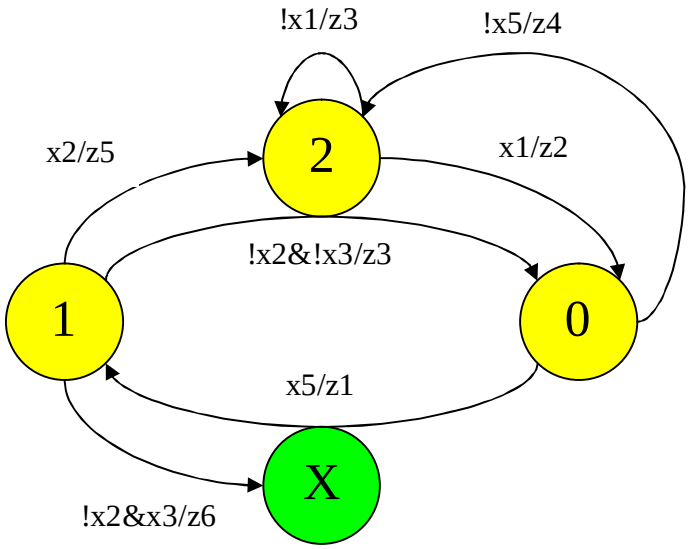


Рис. 53. Автомат для случая достижимой цели

Данный автомат весьма элегантно производит проверку всех необходимых условий, движение в сторону цели, когда это возможно, и обход препятствия. На рис. 54 продемонстрирована траектория движения агента, управляемого данным автоматом. На основе анализа траекторий агента, а также графа переходов управляющего автомата, можно доказать корректность реализованного алгоритма. Полученный алгоритм напоминает алгоритм *Distant Bug* при том условии, что в сведении задачи на клетчатое поле используется манхэттэнская метрика, в отличие от более традиционной евклидовой.

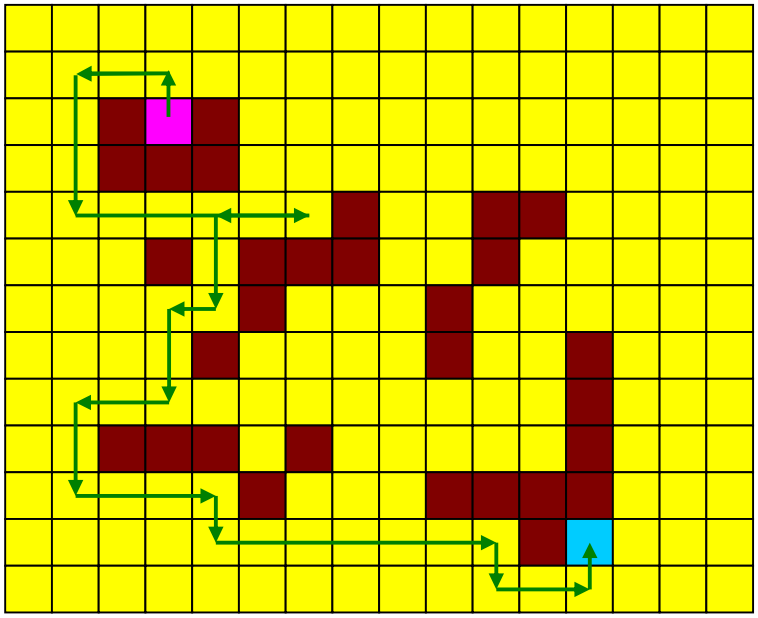


Рис. 54. Траектория движения агента, управляемого автоматом

Полученный автомат достаточно просто дополнить одним переходом и немного модифицировать условия в состоянии 2, чтобы получить полное решение задачи. Этот автомат изображен на рис. 55.

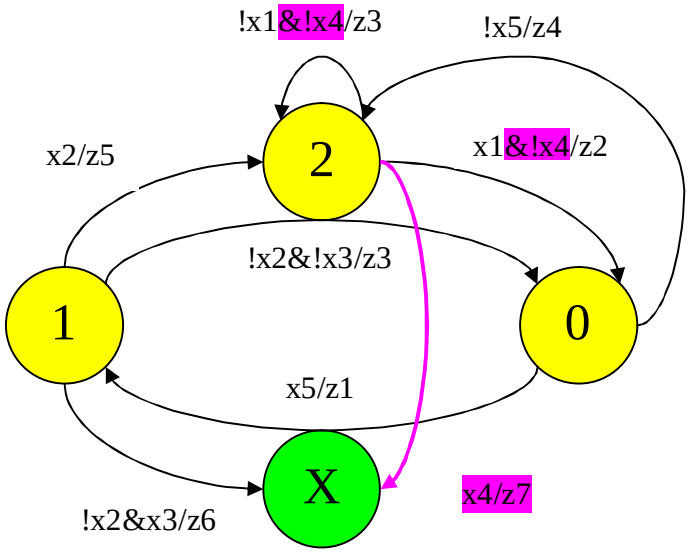


Рис. 55. Автомат для произвольного случая

Исходя из изложенного, можно сделать следующие выводы. Генетические алгоритмы с использованием коэволюции, а также других нестандартных методик возможно применять для генерации конечных автоматов, используемых при решении тех задач навигации, где применима парадигма автоматного программирования. При этом полученные решения обладают аналитически доказуемой корректностью, а в процессе их поиска компьютер находит новые нетривиальные методы решения встречающихся задач.

1.4. Разработка, программная реализация и экспериментальное исследование метода совместного применения конечных автоматов и нейронных сетей

В настоящем разделе приводится описание метода совместного применения конечных автоматов и нейронных сетей.

1.4.1. Краткий обзор существующих методов

Использование методов искусственного интеллекта получает все большее распространение. Одним из перспективных направлений в области создания вычислительных систем для обработки информации является применение нейронных сетей, как в программном, так и в аппаратном (нейрокомпьютерные сети) исполнении.

Нейронные сети применяются для решения задач классификации или кластеризации многомерных данных. Основная идея, лежащая в основе нейронных сетей – это последовательное преобразование сигнала параллельно работающими элементарными функциональными элементами. Таким образом, решения на основе нейронных сетей позволяют производить параллельную обработку данных, позволяя тем самым увеличить производительность вычислительных систем.

Основой нейронной сети является нейрон. Нейрон состоит из трех логических блоков: входы, функция преобразования, выход. На каждый вариант входа (вектор) функция преобразования нейрона вырабатывает определённый сигнал (выход, обычно скаляр) и передает его на входы другим нейронам сети. Подавая на входы некоторым нейронам сигналы извне, и, отметив выходы части нейронов как выходы сети в целом, мы получим систему, осуществляющую отображение $R^n \rightarrow R^k$, где n - размерность входа (информации извне), а k - размерность выхода. Нейронные сети различаются функцией преобразования в нейронах, внутренней архитектурой связей между нейронами и методами настройки (обучения).

Наиболее известными нейросетевыми архитектурами являются:

- Персептрон;
- Сеть Хопфилда;
- Рекурсивные сети;
- Вероятностные сети.

В контексте данной главы рассмотрим совместное применение нейронных сетей и конечных автоматов при решении различных задач, в частности, задачи контроля работы автоматных программ.

Технологию применения нейронных сетей можно условно поделить на две основные категории:

- Программные технологии;
- Аппаратные технологии.

Аппаратные технологии получают свое воплощение в виде нейро-БИС, нейрокомпьютеров и т.д.

Программные технологии используются при различных экспериментах в области разработки стратегий обучения и самообучения, а также при решении неформализованных задач: распознавание, классификация, предсказание.

Существуют также brain-технологии, в которых на базе семантических моделей и моделей искусственного интеллекта рассматривается решение практических задач в нейронных сетях.

1.4.2. Аппаратные технологии нейронных сетей

Впервые формальная модель нейрона была предложена в 1943г. американскими математиками У.Мак-Каллоком и У.Питтсом, затем она была уточнена и развита С.Клини, Й. Фон Нейманом и др.

Прямая аппаратная реализация нейрона на основе умножителей и сумматора требует высокой точности изготовления компонент. По этой причине более эффективной является цифровая реализация на основе применения однородных структур.

Западный вариант такой реализации основан на использовании однородных структур тетраидального типа (С.Мурога – 1979г.), российский вариант – гексагонального сотового типа (В.Л.Беляевский, В.А,Горбатов – 1970г.). Гексагональная структура является на два-три порядка более эффективной по сравнению с тетраидальной в смысле объема аппаратурных затрат [2].

1.4.3. Программные технологии нейронных сетей и автоматы

В области программной технологии применение нейронных сетей совместно с конечными автоматами представлено многочисленными примерами, одним из которых является статья [14].

В этой работе описывается совместное применение указанных технологий для управления жидкостным реактивным двигателем РД-0124 (год разработки 2006). При помощи нейронной сети регистрируются и распознаются параметры вращения вала турбонасосного агрегата, а полученные значения передаются на вход конечного автомата (рис. 56).

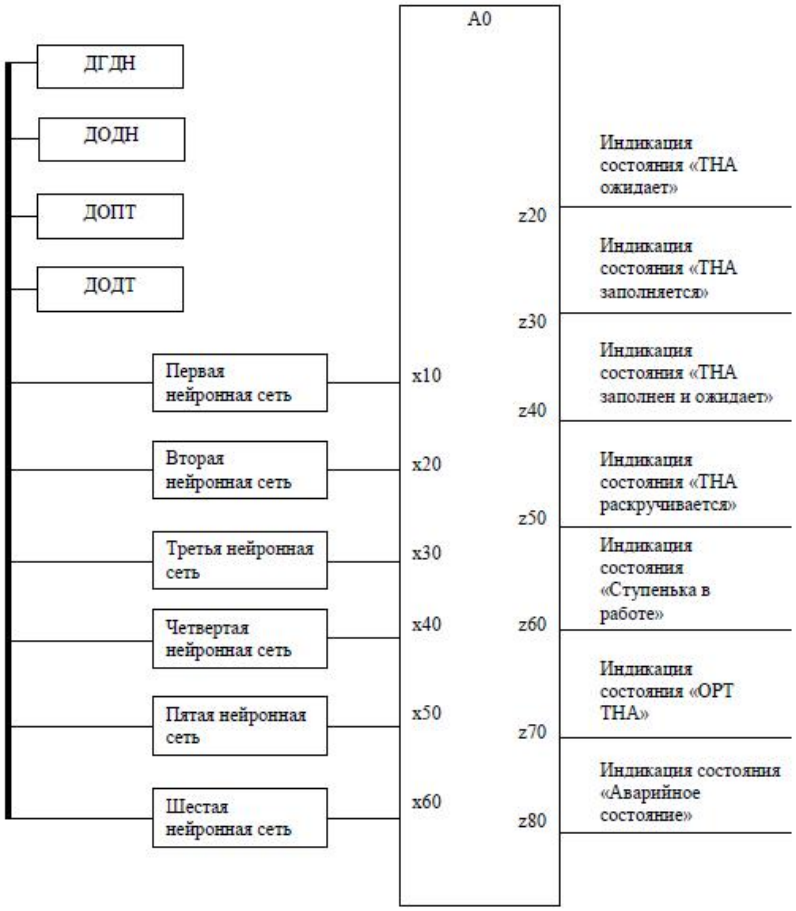


Рис. 56. Схема связей конечного автомата и нейронных сетей

1.4.4. Постановка задачи динамического контроля

Практически во всех работах, посвященных применению нейронных сетей совместно с конечными автоматами, нейронная сеть рассматривается как промежуточное звено, позволяющее преобразовать числовые значения в бинарные переменные.

В данной главе предлагается несколько иной подход, который предусматривает организацию контроля работы конечного автомата при помощи нейронной сети.

В настоящее время получает распространение проверка свойств программ на моделях. Однако такая мера не является достаточным средством для обеспечения надежности программного обеспечения, в силу того, что это не более чем проверка выполнения лишь *некоторых* свойств программы. Полноту такой проверки следует также доказать, что является также нетривиальной задачей. Таким образом, программа остается бесконтрольной *в процессе работы*. Кроме того утверждение о текущем состоянии программы может зависеть от всей истории исполнения – описания последовательности всех предшествующих переходов и всех промежуточных состояний. Это означает, что доказательство свойств такой программы не может быть статическим, и его следует вести динамически – параллельно с ее исполнением при конкретных исходных данных.

Можно поставить задачу динамического контроля работы автоматных программ. Для решения этой задачи предлагается динамически измерять значения некоторых параметров работающей автоматной программы. Измеренные значения параметров могут использоваться, к примеру, как

аргументы правил поведения контролирующей системы, для выработки управляющих воздействий по результатам измерений и т.д. Разработанный метод предоставляет возможность осуществлять контроль без введения каких-либо специальных состояний в автоматную программу, добавления (удаления) переходов, то есть фактически без изменения самой автоматной программы.

На практике это означает, что при контроле эффективности работы программы возможно использовать не только выходные параметры, генерируемые самой программой, но и некоторые показатели «поведения» программы.

1.4.5. Метод решения задачи динамического контроля

Метод состоит в следующем: процесс работы автоматной программы представляется в виде дискретной функции, пригодной для анализа средствами цифровой обработки сигналов.

Рассмотрим алгоритм преобразования процесса работы автоматной программы в дискретную функцию (рис. 57):

- перенумеруем состояния автоматной программы;
- будем осуществлять запись состояний в ходе работы программы;
- отложив по оси ординат номера состояний, а по оси абсцисс время (или такты работы автомата), получим дискретную функцию.

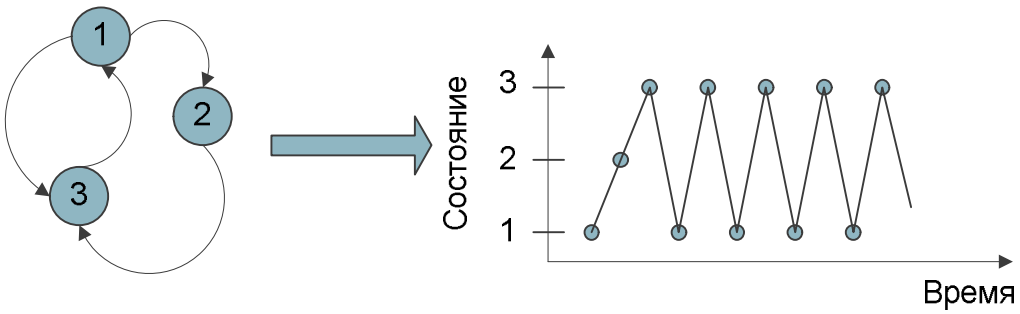


Рис. 57. Преобразование процесса работы автоматной программы в дискретную функцию

Можно рассмотреть полную четырехмерную модель: время, состояние, вход и выход.

Полученная дискретная функция – является *траекторией автомата* и может быть проанализирована средствами цифровой обработки сигналов.

Следует отметить, что для наблюдателя не всегда становится возможным накопление полной траектории автомата, например, в силу того, что подконтрольный автомат работает существенно быстрее контролирующего автомата. В этом случае могут наблюдаться явления, сходные с наложением спектров при дискретизации непрерывных сигналов.

1.4.6. Пример работы системы предупреждения

Одним из наиболее интересных подходов к анализу является построение нейросетевой, либо иной обучающейся системы классификации. Классификатор обучается с помощью тех функций, которые были получены при штатной работе автоматной программы (либо использует

самообучение). Обученный таким образом классификатор способен с некоторой вероятностью обнаруживать нештатную работу программы.

Отметим, что данный подход позволяет не только контролировать работу автоматных программ, но и сравнивать автоматы между собой по поведению, а не по графу переходов, что необходимо для уменьшения пространства поиска при решении задач выращивания автоматных программ методами генетического программирования.

Рассмотрим автомат принимающий на вход отношения вида *a<>b;* в виде строк (рис. 58), где *a* и *b* – числа, <> – знаки отношения и символ окончания *;* (например “`10>9;`”, “`6<12;`”).

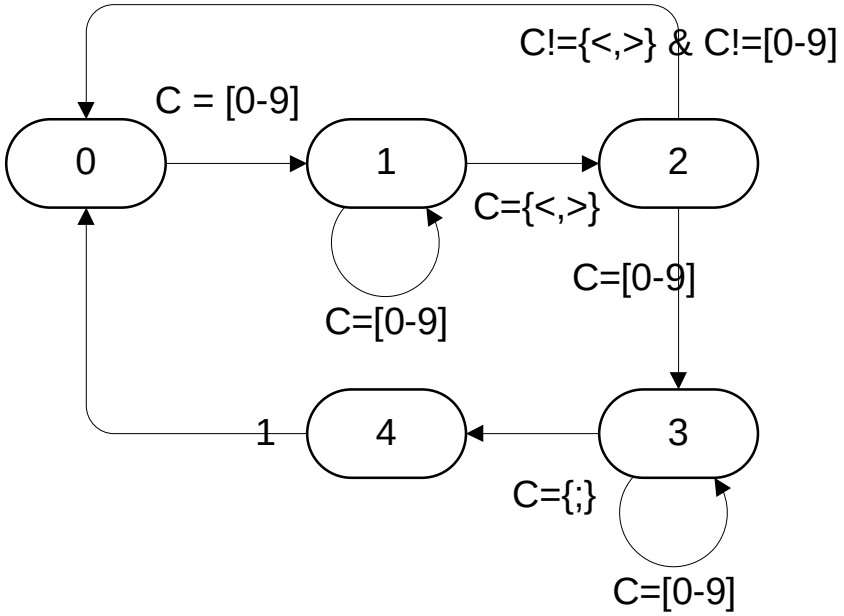


Рис. 58. Автомат, обрабатывающий строки вида a<>b

На данном этапе функция, выполняемая автоматом, не имеет значения, достаточно лишь того, чтобы результаты ее работы выглядели правильно со стороны наблюдателя.

Согласно предложенной концепции, сформируем дискретную функцию, описывающую работу автомата, и подадим ее на вход нейронной сети.

В качестве нейронной сети используется персептрон (рис. 59) с одним скрытым слоем состоящим из 21 нейрона и выходным слоем состоящим из пяти нейронов. Обучение производится методом обратного распространения в ручном или автоматическом режиме.

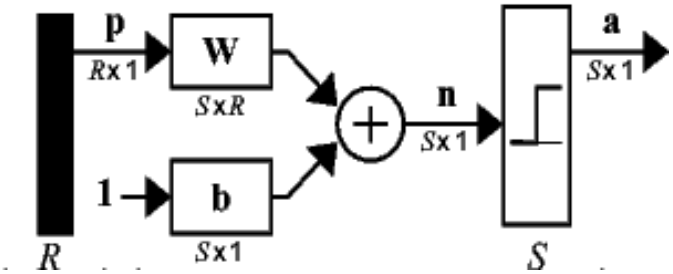


Рис. 59. Архитектура используемой нейронной сети

В ручном режиме в качестве учителя выступает пользователь, который оценивает правильность работы по пятибалльной шкале.

В автоматическом режиме система обучается в течение некоторого времени. При этом гарантируется, что в данный отрезок времени контролируемый автомат работает правильно.

Зависимость ошибки обучения нейронной сети от номера шага приведена на рис. 60. Из графика видно, что нейронная сеть обучилась всего за шесть шагов.

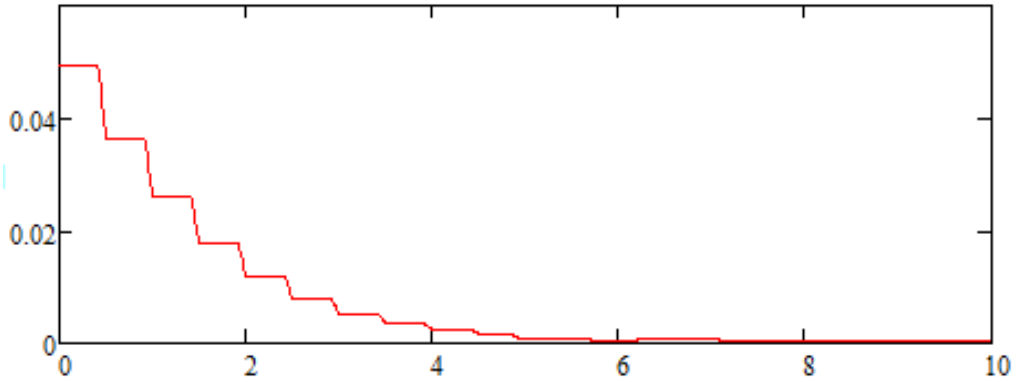


Рис. 60. Процесс обучения нейронной сети

В дальнейшем на обученную нейронную сеть оказывались следующие возмущающие воздействия (порядок соблюден):

- на вход поступают правильные данные, и работает правильный автомат;
- на вход поступают зашумленные данные;
- на вход поступают правильные данные, и работает правильный автомат;
- в наблюдаем автомате удален один из переходов.

Реакция нейронной сети представлена на рис. 61.

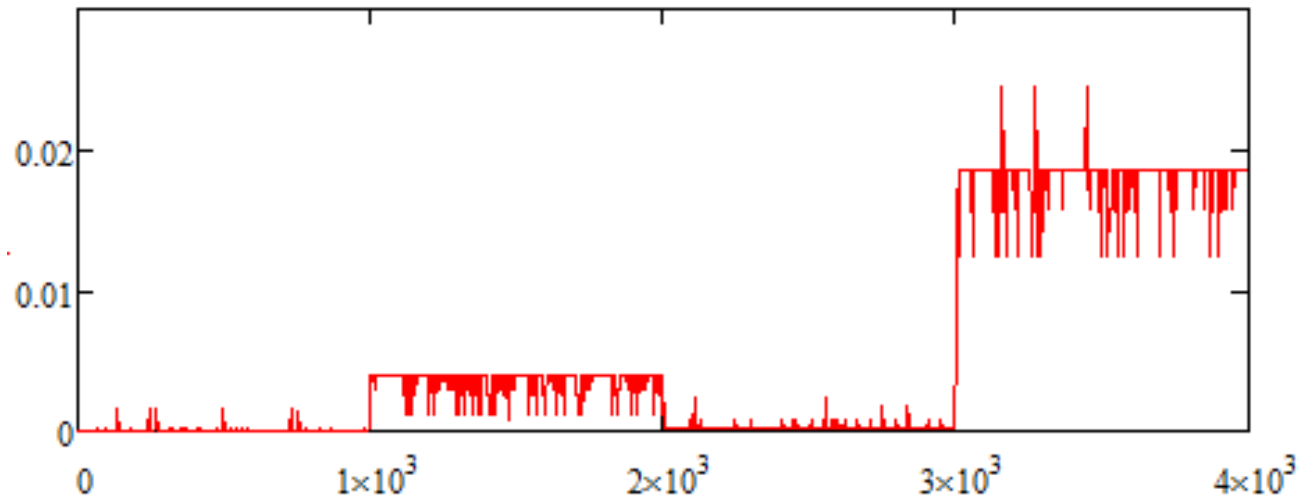


Рис. 61. Реакция нейронной сети на возмущение

Из графика видно, что нейронная сеть правильно сигнализировала об изменениях в поведении автомата. Безусловно, теоретические вопросы, связанные с настройкой сети и схемой ее построения, а также иные, связанные с предложенным методом требуют дальнейшей проработки.

1.5. Разработка метода применения верификации моделей программ при построении управляющих конечных автоматов с помощью генетического программирования

В настоящем разделе приводится описание метода применения верификации моделей программ при построении управляющих конечных автоматов с помощью генетического программирования.

1.5.1. Верификация автоматных программ

Метод проверки того, что программная система соответствует заявленной спецификации (обладает необходимыми свойствами или удовлетворяет определенным требованиям (утверждениям)), называется *верификацией*. К сожалению, верифицировать систему обычно намного сложнее, чем ее создать. Это также является одним из факторов того, что использование верификации в процессе создания самих автоматов позволит не только генерировать автоматы с заранее заданным поведением, но и в какой-то степени избавляет нас от необходимости верифицировать систему после окончания ее построения. При этом отметим, что аккуратное и точное описание свойств автомата на языке верификатора также является непростой задачей и требует определенных навыков и умений обращения с темпоральными свойствами.

Наиболее практичным в настоящее время является метод верификации, называемый *Model Checking* [39, 14]. При его использовании процесс верификации состоит из трех этапов. Первый из них, моделирование программы состоит в преобразовании программы в формальную модель с конечным числом состояний для последующей верификации. Второй этап, спецификация – формальная запись утверждений, которые требуется проверить. На третьем этапе, выполняется собственно верификация – алгоритмическая проверка выполнения спецификации для модели.

Сложность такого подхода заключается в том, что после построения модели и ее верификации, необходимо обратное преобразование ошибки в модели в ошибку в программе. Причем не всегда

корректность модели означает соответствие программы спецификации, так как при построении модели мы переходим на другой уровень абстракции, теряя определенные данные и связи в программе. Данный процесс представлен на рис. 62.

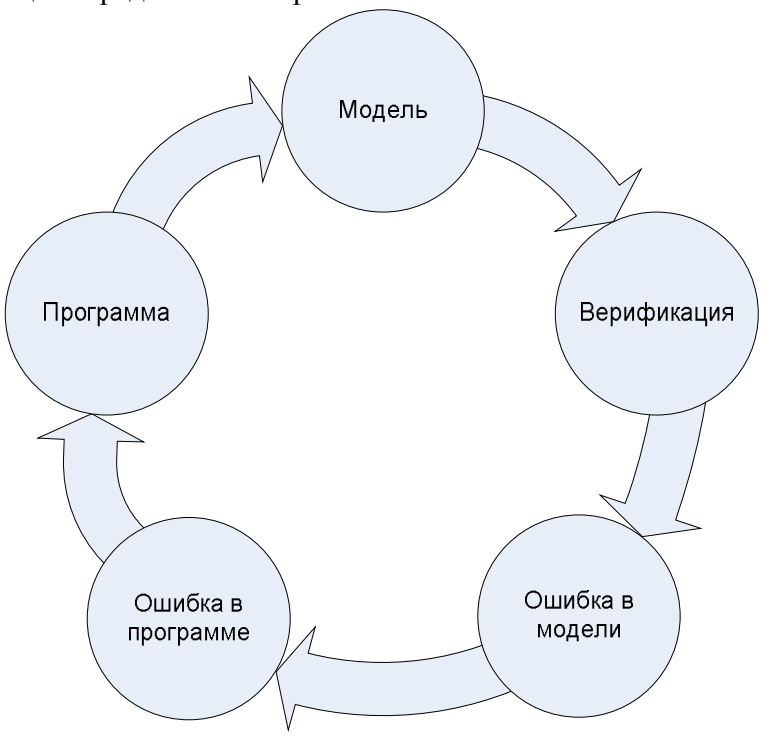


Рис. 62. Стандартный процесс верификации программы

Однако, метод *Model Checking* достаточно хорошо подходит для случая автоматных программ, так как нет необходимости преобразовывать программу в модель, и после верификации, в случае обнаружения контрпримера, совершать обратное преобразование. Это объясняется тем, что автомат уже является моделью, пригодной для верификации, и не требует никаких дополнительных преобразований и упрощений. Такая особенность автоматов позволяет строить утверждения о программе в терминах автоматов, что упрощает их верификацию по сравнению с программами, написанными традиционным путем (без явного выделения состояний).

В предлагаемом методе будет проводиться верификация не всей автоматной программы, а только ее модель (конечный автомат). Это немного упрощает задачу, но так же, как и при создании автоматной программы на основе тестов, мы считаем поставщиков событий и объекты управления достаточно простыми, а вся сложная логика вынесена в автоматную модель. При верификации будем рассматривать поставщиков событий и объекты управления в качестве «внешней среды», которая ничего не помнит о последовательности переходов рассматриваемого автомата. Таким образом, в любой момент времени может быть получено любое событие, и любое условие на переходе может быть как истинным, так и ложным. Это приводит к тому, что автомат может совершить любой переход из данного состояния. Такой подход уже был рассмотрен в работе [24]. Но это не является упрощением модели в случае вынесения всей логики в автоматную.

В настоящей работе требования к программе формулируются в виде формул темпоральной логики линейного времени (*Linear Temporal Logic, LTL*). Далее кратко опишем синтаксис и семантику этого языка. Сразу заметим, что как выбор языка темпоральной логики, так и выбор верификатора не влияет на предложенный метод построения автоматных программ генетическими алгоритмами. Это позволяет применять его с другими программными средствами, реализующими верификацию методом *Model Checking*.

1.5.2. Язык логики линейного времени

Как уже отмечалось ранее, для описания поведения автомата будем применять утверждения, написанные на языке *LTL*. Синтаксис *LTL* включает в себя пропозициональные переменные *Prop*, булевы связки ($\neg$, Λ, V) и темпоральные операторы. Последние применяются для составления утверждений о событиях в будущем и интерпретируются как $I: Prop \rightarrow .\{True, False\}$.

Логика линейного времени расширяет классическую логику, добавляя временные операторы. В нем время линейно и дискретно, и в каждый момент времени любая пропозициональная переменная может быть истиной или ложной.

Будем использовать следующие темпоральные операторы:

- **X** (ne**X**t) – «Xp» – в следующий момент выполнено p;
- **F** (in the **F**uture) – «Fp» – в некоторый момент в будущем будет выполнено p;
- **G** (**G**lobally in the future) – «Gp» – всегда в будущем выполняется p;
- **U** (**U**ntil) – «pUq» – существует состояние, в котором выполнено q и во всех предыдущих выполняется p;
- **R** (**R**elease) – «pRq» – либо во всех состояниях выполняется q, либо существует состояние, в котором выполняется p, а во всех предыдущих выполнено q.

Множество *LTL* формул таково:

- пропозициональные переменные *Prop*;
- *True, False*;
- φ и ψ – формулы, то
  - $\neg\varphi$, $\varphi\Lambda\psi$, $\varphi V\psi$ – формулы;
  - $X\varphi$, $F\varphi$, $G\varphi$, $\varphi U\psi$, $\varphi R\psi$ – формулы.

Логика линейного времени говорит о всех путях. Таким образом, логика *LTL* предполагает, что некоторое утверждение будет выполняться для всех путей. Поэтому можно строить доказательство от противного и проверять существование пути, на котором будет выполняться отрицание данной формулы. Если такой путь не будет найден, то формула выполнима.

1.5.3. Описание предлагаемого метода

Исходными данными для построения конечного автомата управления системой со сложным поведением являются:

- список событий;
- список входных переменных;
- список выходных воздействий;
- набор тестов, каждый из которых содержит последовательность Input[i] событий, поступающих на вход конечному автомату, и соответствующую ей эталонную последовательность Answer[i] выходных воздействий;
- набор темпоральных свойств, записанных на языке логики *LTL*.

Отметим, что метод, описанный в этом разделе, основан на методе из построения конечных автоматов на основе обучающих примеров из работы [19], однако предлагаемый подход использует результат верификации в генетическом алгоритме на стадии вычисления функции приспособленности и на стадии мутации. Такая интеграция верификации в уже разработанный метод автоматического создания программ на основе только тестов, по сути, позволяет использовать любое средство для верификации модели с любым входным языком.

Так же, как при создании автоматной модели только на основе тестов, запись *LTL*-формул не предполагает реализации объектов управления и поставщиков событий заранее – они могут быть созданы и после создания модели. Однако, можно создавать модель по уже готовой реализации поставщиков событий и объектов управления. Первый случай может возникнуть, когда мы создаем

программу с нуля и предоставляем только интерфейс поставщиков событий и объектов управления, откладывая их реализацию. Второй вариант – когда у нас уже есть программа с неправильной моделью, или реализация этих объектов заранее известна.

Стоит отметить, что заранее мы знаем только входные воздействия, входные условия и выходные воздействия, то есть мы можем строить утверждения только о них. Это означает, что мы не можем использовать в *LTL*-формулах предикаты о состояниях, так как мы не знаем заранее ничего о структуре конечного автомата. С другой стороны, это можно считать и преимуществом, так как мы заранее не ограничиваем себя априорными знаниями о состояниях будущего автомата. Тем более, создавая программу таким образом (на основе тестов и темпоральных свойств), мы не знаем какая модель должна получиться, а только можем заранее описать ее требуемое поведение. Ведь иначе можно было бы создать ее вручную, а только затем верифицировать.

1.5.3.1. Представление конечного автомата в виде хромосомы генетического алгоритма

В настоящей работе используется такое же представление конечных автоматов в виде хромосом генетического алгоритма, как и в работе [19].

Конечный автомат в алгоритме генетического программирования представляется в виде объекта, который содержит описания переходов для каждого из состояний и номер начального состояния. Для каждого из состояний хранится список переходов. Каждый переход описывается событием, при поступлении которого этот переход выполняется, и числом выходных воздействий, которые должны быть сгенерированы при выборе этого перехода.

Таким образом, в особи кодируется только «скелет» (рис. 63) управляющего конечного автомата, а конкретные выходные воздействия, вырабатываемые на переходах, определяются с помощью алгоритма расстановки пометок.

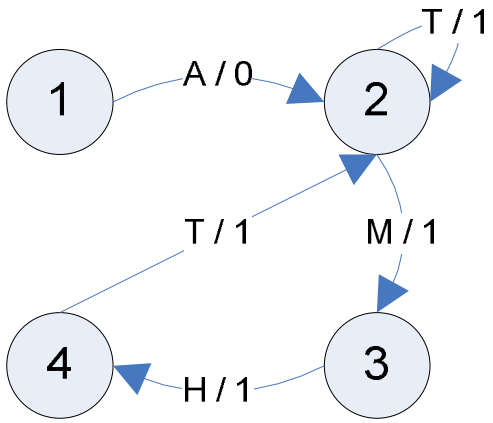


Рис. 63. Некоторые переходы «скелета» автомата

В настоящей работе генетический алгоритм остался таким же, как и в работе [19], то есть он имеет все те же стадии: создание начальной популяции, вычисление функции приспособленности, скрещивание, мутация. Также сохранилась стратегия эллитизма. Однако предлагается учитывать результат верификации при вычислении функции приспособленности и мутации, о чем будет написано ниже. Для начала отметим отличие в обработки входных переменных при генерации на основе тестов и при верификации.

1.5.3.2. Обработка входных переменных

Каждый переход может иметь условие, при котором он совершается. Такое условие записывается в виде логической формулы, которая задает ограничения на значения входных

переменных. Например, можно записывать «*T [!x1 && x2]*», что означает, что переход совершится по событию *T* при условии невыполнимости *x1* и выполнимости *x2*. При создании автоматной модели только на основе тестов, такой переход считается отличным от перехода просто по событию *T* без всяких условий – можно считать, что это два разных события.

Однако, оба перехода идентичны для предиката *wasEvent(T)* («*переход совершен по событию T*»). При верификации, если переход был совершен по событию *T* или этому же событию, но с некоторыми условиями, то в обоих случаях данный предикат, естественно, будет выполнен. Заметим, что можно вводить предикаты и на условия на переходах, тогда такие переходы будут отличаться и для верификатора. Например, можно добавить предикат *wasTrue(x1)* (условие x1 верно), тогда он не будет выполняться на переходе «*T [!x1 && x2]*», но будет верен для перехода «*T [x1 && x2]*».

1.5.3.3. Вычисление функции приспособленности

В любом генетическом алгоритме ключевую роль играет вычисление функции приспособленности. Она позволяет количественно оценивать особи: чем больше функция приспособленности, тем лучше особь. Таким образом, независимо от выбора стратегии генетического алгоритма, функция приспособленности лучшей особи в популяции, как правило, должна расти. В любом случае поиск автоматной модели считается завершенным, когда найдена особь, для которой значение функции приспособленности превышает некоторое целевой значение, заданное заранее.

Особь, проходящая все тесты и удовлетворяющая всем темпоральным свойствам, является той, которую мы ищем. Это означает, что ее функция приспособленности должна быть больше, чем у особи, не проходящей определенное число тестов или не удовлетворяющей неким формулам.

Сразу заметим, что, так же как и в работе [19], в функции приспособленности должно учитываться общее число переходов в конечном автомате, однако вклад числа переходов должен быть меньше, чем формул или тестов. Ведь нам важнее найти «правильную» модель, а не модель с наименьшим числом состояний.

В настоящей работе функция приспособленности является суммой трех частей. Каждая часть представляет собой вклад одного из критериев: успешность прохождения тестов, выполнимость *LTL*-формул, число переходов в конечном автомате.

Напомним, что функция приспособленности для тестов основана на редакционном расстоянии (расстоянии Левенштейна) [17]. Для ее вычисления выполняются следующие действия: на вход автомату подается каждая из последовательностей Input[*i*]. Обозначим последовательность выходных воздействий, которую сгенерировал автомат на входе Input[*i*] как Output[*i*]. После этого вычисляется величина $FF_1 = \frac{\sum_{i=1}^{n}(1 - \frac{ED(Output[i], Answer[i])}{\max(|Output[i]|, |Answer[i]|)})}{n}$, где как ED*(A, B)* обозначено редакционное расстояние между строками *A* и *B*. Отметим, что значения этой функции лежат в пределах от 0 до 1, при этом, чем «лучше» автомат соответствует тестам, тем больше значение функции приспособленности.

Для того чтобы выделять особи, проходящие все тесты, и те, которые проходят только часть, предлагалось вычислять функцию приспособленности для тестов по формуле: $FF_{test} = \begin{cases} 0.5 \cdot T \cdot FF_1, FF_1 < 1 \\ T, FF_1 = 1 \end{cases}$, где T – «стоимость» прохождения всех тестов.

Вклад *LTL*-формул в общую функцию приспособленности предлагается оценивать как доля верных формул рассматриваемой особи. Этот вклад можно оценить как $FF_{LTL} = F \cdot \frac{n_1}{n_2}$, где F – «стоимость» выполнения всех формул, n_1 – число успешно выполненных *LTL*-формул, а n_2 – общее

число формул. Таким образом, чем больше число верных формул, тем больше вклад темпоральных свойств в функцию приспособленности, а, при выполнимости всех свойств, их вклад становится равным F.

Функция приспособленности зависит не только от того, насколько «хорошо» автомат работает на тестах и удовлетворяет формулам, но и числа переходов, которые он содержит. Таким образов, функцию приспособленности можно вычислить по формуле: $FF = FF_{test} + FF_{LTL} + \frac{(C - cnt)}{C}$, где как cnt обозначено число переходов в автомате, C – число, больше чем число переходов, а FF_{test} и FF_{LTL} – вклады тестов и формул соответственно. Эта функция приспособленности устроена таким образом, что при одинаковом значении $FF_{test} + FF_{LTL}$, отражающем «прохождение» тестов и *LTL*-формул автоматом, преимущество имеет модель, содержащая меньше переходов.

В тоже время, мы хотим построить модель, проходящую все тесты и удовлетворяющую всем *LTL*-формулам. Поэтому генетический алгоритм сначала строит такую особь, а потом уже пытается уменьшить число переходов. Для обеспечения такого поведения требуется подбирать значение *C* из формулы, приведенной выше, таким образом, что бы вклад успешно пройденного теста и удовлетворение темпоральному свойству был больше, чем, например, уменьшение числа переходов на единицу.

Также в настоящей работе предлагается не вычислять функцию приспособленности для всех тестов и формул одновременно, а разбивать их на группы и вычислять для каждой группы отдельно. Каждая такая группа включает набор тестов и набор *LTL*-формул, которые описывают определенное поведение модели, а значит особь, которая полностью проходит все тесты конкретной функциональности и удовлетворяет всем ее темпоральным свойствам, будет лучше, чем особь, проходящая только часть тестов из разных групп. Тогда можно записать функцию приспособленности следующим образом: $FF = \sum_{i=1}^{N} \left(FF_{test,i} + FF_{LTL,i}\right) + \frac{(C - cnt)}{C}$, где $FF_{test,i}$ и $FF_{LTL,i}$ – функции приспособленности для тестов и для темпоральных свойств, вычисленные для *i*-ой группы, а как *N* обозначено число групп. Вычисление функций приспособленности для каждой группы выполняется так же, как описано выше.

Оценим время вычисления функции приспособленности. Время вычисления редакционного расстояния пропорционально произведению длин последовательностей, для которых оно вычисляется. Таким образом, время вычисления функции приспособленности для тестов есть $O(\sum_{i=1}^{n} |\,\mathrm{Output}[i]\,| \cdot |\,\mathrm{Answer}[i]\,|)$. Заметим также, что добавление в набор тестов «префиксов» тестов не увеличивает время вычисления функции приспособленности, так как достаточно вычислить редакционное расстояние только для «самых больших» тестов, а для их префиксов редакционное расстояние взять из вычисленной таблицы динамического программирования.

Оценить время верификации одной *LTL*-формулы можно только примерно. Если n – длина формулы, то в худшем случае будет построен автомат Бюхи с $n \times 2^n$ состояниями [36]. Значит, автомат-пересечение будет содержать $n \times 2^n \times V$ состояний, где V – число вершин в модели. Заметим, что верификатор использует средство *LTL2BA* [45] для построения автомата Бюхи по *LTL*-формуле, которое преобразует формулу перед построением автомата и модифицирует автомат после. В итоге построенный автомат не будет экспоненциально расти от длины формулы. Как правило, верифицируемые формулы достаточно компактные, что не будет приводить к автоматам с большим числом состояний.

Так как формулы задаются заранее, то их преобразование в автоматы Бюхи производится один раз. Однако пересечения этих автоматов с автоматом модели придется выполнять каждый раз при вычислении функции приспособленности новой полученной особи.

Также заметим, что при верификации не всегда требуется целиком обходить автомат пересечения. Если модель не удовлетворяет темпоральному свойству, то контрпример может быть найден задолго до обхода всего автомата. Но даже, когда формула выполняется, автомат пересечения может быть меньше, чем $n \times 2^n \times V$, так как многие вершины могут оказаться недостижимы.

Из сказанного следует, что процесс верификации может занимать много времени и, чем больше формул мы хотим использовать при построении автомата, тем дольше будет вычисляться функция приспособленности конкретной особи. Так как в каждом поколении могут быть тысячи особей, то выбор верификатора и его эффективность в плане производительности крайне важны.

1.5.3.4. Учет результата верификации при вычислении функции приспособленности

Вклад каждого теста оценивается по редакционному расстоянию между эталонной выходной последовательностью и выходной последовательностью, сгенерированной особью. Тем самым, каждый тест вносит не просто «0» или «1», показывая, что тест пройден на конечном автомате или нет, а некоторое вещественное число из отрезка [0; 1] (оба конца включительно).

Предлагается оценивать вклад каждой *LTL*-формулы аналогичным образом – сделать вклад каждой формулы не дискретным, а вещественным (из отрезка [0; 1]). Однако, используемый верификатор умеет только сообщать, что формула верна или приводить контрпример. В настоящей работе верификатор был расширен таким образом, чтобы можно было помечать переходы в процессе двойного обхода в глубину. В результате, когда во время первого обхода в глубину состояние конечного автомата покидается, все переходы, ведущие из него, помечаются как *проверенные*. Это означает, что они точно не лежат на пути, опровергающем *LTL*-формулу.

Предлагается в качестве вклада *LTL*-формул в функцию приспособленности брать отношение числа проверенных переходов к числу достижимых переходов. Формулу можно записать как $\mathrm{FF}_{\mathrm{LTL}} = \sum_{i=1}^{n} \frac{\mathrm{t}_i}{\mathrm{T}} \Big/ \mathrm{n}$, где n – число *LTL*-формул, T – число достижимых переходов в особи, t_i – число переходов, помеченных как проверенные при верифкации i-ой формулы. Чем больше переходов было отмечено в процессе верификации, тем больше вклад формулы в функцию приспособленности, а значит и особь является более приспособленной.

Приведем пример вычисления вклада одной из *LTL*-формул. Пусть мы верифицируем конечный автомат из шести состояний, представленный на рис. 64. Цифрой «1» выделена часть автомата, на которой не удалось обнаружить контрпример, а цифрой «2» – часть автомата, на которой формула опровергается.

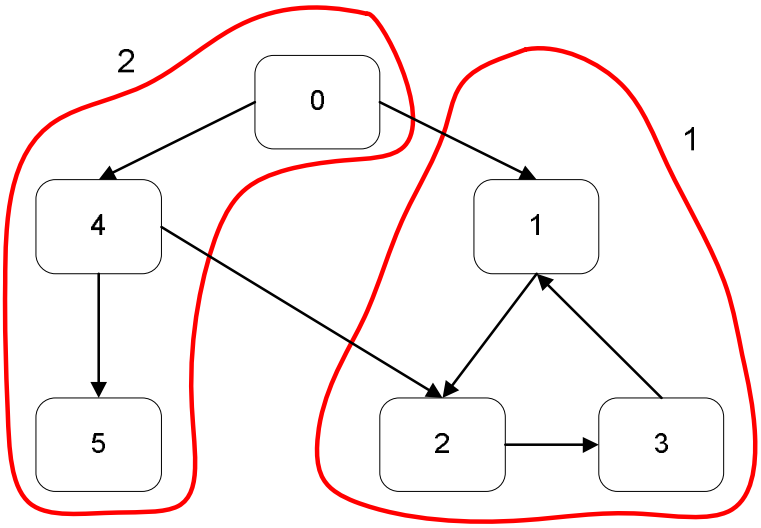


Рис. 64. Пример вычисления вклада LTL-формулы в функцию приспособленности

Таким образом, вклад данной формулы будет $FF_i = \frac{3}{7}$, где 3 – число помеченных переходов из первого подмножества, а 7 – общее число переходов в конечном автомате.

Заметим, что порядок обхода в глубину состояний и переходов автомата не гарантируется, поэтому могло получиться так, что контрпример (помеченный цифрой 2 на рис. 64) мог быть найден сразу. Тогда вклад формулы в функцию приспособленности оказался бы нулевым, так как ни один из переходов не был бы помечен.

1.5.3.5. Операция мутации

Выполнимость или невыполнимость *LTL*-формул позволяет только отбирать «лучшие» особи, но не позволяет «улучшать» популяцию путем скрещивания или мутации, так как этот процесс был бы случайным и не гарантировал бы увеличения числа верных утверждений в следующем поколении. То есть, учитывая только относительное число выполненных темпоральных свойств, мы не можем влиять на результат скрещивания или мутации, однако именно эти операции и приводят к росту значения функции приспособленности.

Для преодоления описанного недостатка при выполнении операции мутации предлагается учитывать контрпример, построенный верификатором. Такой контрпример представляет собой список вершин и переходов автомата, которые опровергают *LTL*-формулу. По сути, это путь в автоматной модели, на котором выполняется отрицание формулы. Имея информацию о таком пути, мы можем увеличить вероятность скрещивания или мутации для одного или нескольких переходов из контрпримера. Например, при выполнении мутации мы можем заменить конечное состояние перехода (входящего в контрпример), входное событие и выходные воздействия. Благодаря предложенным действиям перестанет существовать данный контрпример для *LTL*-формулы, и увеличатся шансы новой особи соответствовать большему числу *LTL*-формул.

Если писать конкретнее, то при верификации выбирается самый длинный по числу переходов контрпример. Затем при создании новой особи с переходами из этого контрпримера могут происходить следующие действия:

- при копировании перехода в новую особь с некоторой вероятностью у него одновременно меняются: входное событие, число выходных воздействий и конечное состояние;

- при обычной мутации, если она заключается в удалении перехода у вершины, удаляется тот переход, который лежит на пути, опровергающем формулу.

Конечно, такие действия не гарантируют увеличение значения функции приспособленности у особи в следующем поколении. Но они хотя бы позволяют убрать путь, на котором выполняется отрицание *LTL*-формулы, а тогда, вероятно, исходная формула будет выполняться на всех возможных путях.

Мы не можем автоматически проанализировать семантику формулы и понять, как надо исправить переход, чтобы формула стала выполняться. Поэтому одновременно меняется входное событие, число выходных воздействий и конечное состояние перехода, так как мы не знаем, какой из предикатов в *LTL*-формуле выполняется или не выполняется на данном пути. Предикат может «говорить» о событии, может «говорить» о вызванном действии, а, может быть, рассматриваемый переход ведет в состояние, для которого некоторая подформула *LTL*-формулы является неверной.

Приведем пример верификации модели и операции мутации. Пусть одна из особей в поколении оказалась такой, как представлена на рис. 65.

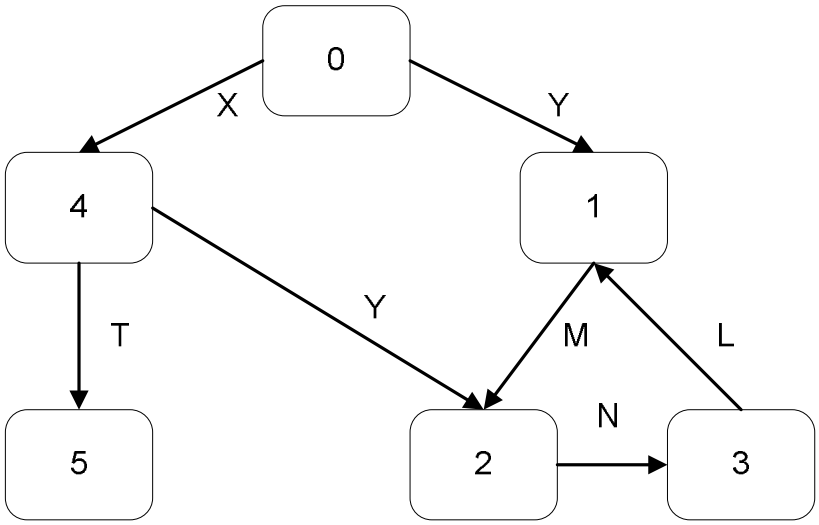


Рис. 65. Пример конечного автомата

Пусть необходимо проверить утверждение «G(!wasEvent(T))», которое означает, что никогда не будет обработано событие T. Так как алгоритм верификации заключается в поиске контрпримера, опровергающего формулу, то мы пытаемся найти путь, на котором выполняется отрицание формулы: «!G(!wasEvent(T))». По отрицанию формулы строится автомат Бюхи, представленный на рис. 66.

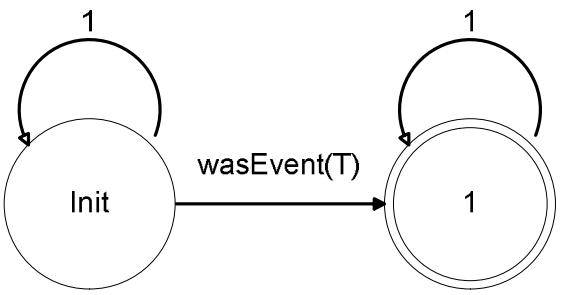


Рис. 66. Автомат Бюхи для формулы «!G(!wasEvent(T))»

Построенный автомат перейдет в допускающее состояние «1» в том и только в том случае, когда автомат модели совершит переход по событию T, так как только тогда предикат «wasEvent(T)»

будет верен. Если же автомат модели не будет содержать такого перехода, или же он будет недостижим из начального состояния, то автомат Бюхи никогда не сможет перейти в состояние «1» и темпоральное свойство будет выполняться.

Не будем явно строить пересечение двух автоматов. Автомат Бюхи, построенный по отрицанию формулы, будет совершать переходы по петле из состояния «Init», пока будет обходиться автомат модели (состояния особи с номерами 0, 1, 2, 3, 4). Только после того, как автомат модели перейдет по событию T из состояния «4» в состояние «5», автомат, допускающий отрицание *LTL*-формулы, сможет сделать переход из состояния «Init» в состояние «1». Так как из состояния модели с номером 5 переходов больше нет, то автомат пересечения запустит второй обход в глубину. В результате работы верификатора будет найден путь в модели, опровергающий формулу (рис. 67).

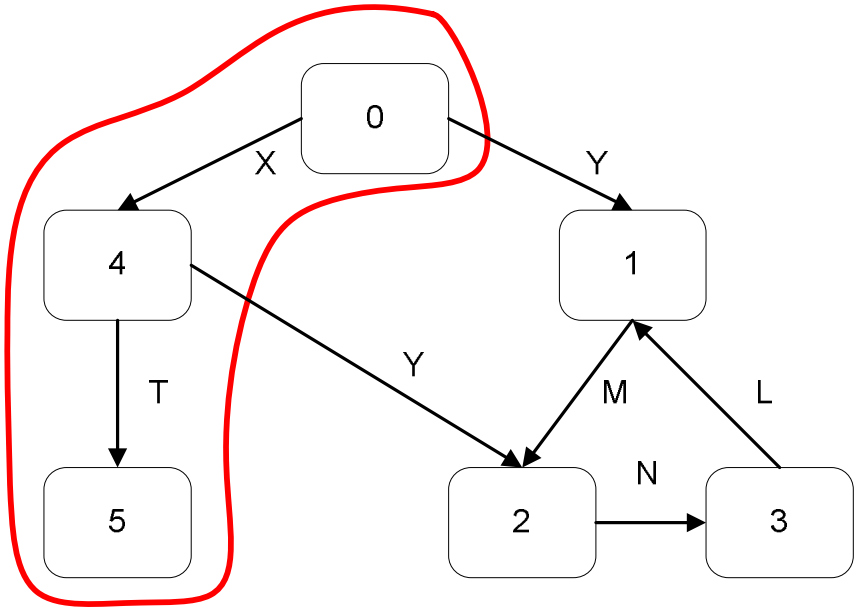


Рис. 67. Путь в модели, опровергающий LTL-формулу

Таким образом, если мы удалим любой переход в найденном контрпимере, переход по событию T перестанет существовать или быть достижимым. В то же время, может помочь и мутация, изменяющая событие на переходе, но, в нашем случае, мутировать должен именно переход по событию T.

Конечно, модель может оказаться не такой простой, и мутация окажется не такой эффективной. Например, если бы был переход из состояния модели «2» в состояние «5» по событию T, то сначала мог быть найден путь, проходящий через состояния с номерами 0, 1, 2, 5 (рис. 68). В таком случае мутация устранила бы данный контрпример, а формула все равно не стала бы выполняться. Но в следующем поколении, верификатор найдет путь, проходящий по состояниям с номерами 0, 4, 5 и устранит второй контрпример, тем самым за два поколения найдется особь, удовлетворяющая заявленной формуле.

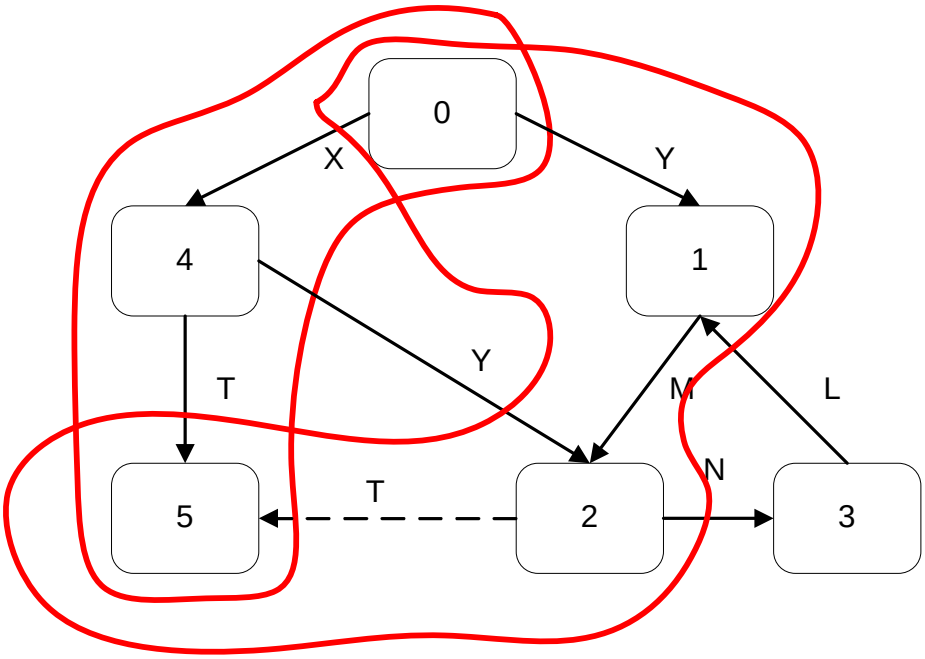


Рис. 68. Модель с двумя контрпримерами

В общем случае мутация остается случайным процессом, просто вероятность мутации переходов из контрпримера выше, чем остальных. Не стоит забывать, что, устраняя контрпример, мы можем нарушить прохождение какого-нибудь теста.

1.5.3.6. Операция скрещивания

Операция скрещивания может выполняться одним из трех способов: случайное, по тестам и с учетом результата верификации. Первые два способа описаны в работе [19].

Случайное скрещивание самое простое – выбираются две особи и их списки переходов «смешиваются». В результате получается особь, в которой часть переходов от одного родителя, а часть от другого.

Скрещивание по тестам основано на том, что часть конечного автомата, на которой «хорошо» проходятся тесты, переходит в новую особь без изменений, а остальные переходы «смешиваются», так же как при случайном. Такое скрещивание позволяет сохранить часть автомата, на которой проходятся тесты, тем самым не уменьшив функцию приспособленности.

1.5.3.7. Скрещивание с учетом результата верификации

Как было изложено выше, алгоритм верификации конечных автоматов основан на двойном обходе в глубину. Таким образом, когда первый обход в глубину покидает состояние, то подграф, образованный просмотренными состояниями и переходами, соответствует *LTL*-формуле.

Напомним, что если состояние в автомате-пересечении является допускающим, то второй обход в глубину проверяет, лежит ли данное допускающее состояние в сильной компоненте связанности. Если состояние не допускающее или цикл, проходящий через него, не был обнаружен, то состояние покидается. Так как алгоритм верификации использует обход в глубину, то все достижимые состояния из покидаемого так же просмотрены и мы можем пометить все исходящие переходы как проверенные.

Приведем псевдокод предлагаемого метода, жирным шрифтом выделен фрагмент, отличающий его от обычного двойного обхода в глубину, помечающий исходящие переходы как верифицированные:

```
procedure emptiness
    for all q0 ∈ Q0 do
        dfs1(q0);
    terminate(False);
end procedure

procedure dfs1(q)
    local q’;
    hash(q);
    for all последователей q’ вершины q do
        if q’ не содержится в хэш-таблице then dfs1(q’);
    if accept(q) then dfs2(q);
    for all переходы t из вершины q do
        markAsVerified(t);
end procedure

procedure dfs2(q)
    local q’;
    flag(q);
    for all последователей q’ вершины q do
        if q’ в стеке dfs1 then terminate(True);
        else if q’ не является помеченной then dfs2(q’);
        end if;
end procedure
```

После пометки переходов, как соответствующих темпоральному свойству, можно проводить скрещивание таким образом, чтобы помеченные переходы перешли в новую особь без изменений.

Так как обычно проверяются несколько формул, то при скрещивании можно брать либо объединение помеченных переходов для разных формул, либо пересечение. Возможно, что какая-то формула выполняется и все переходы помечены, тем самым объединением помеченных переходов будут все переходы. В то же время, мы не знаем в какой последовательности проверяются переходы, и контрпример может быть найден сразу для неверной формулы. Тем самым, ни один переход не будет помечен, и пересечение будет пусто. Что бы избежать обе эти проблемы предлагается брать объединение (или же пересечение) не для всех формул, а только для случайной их выборки.

Приведем пример такого скрещивания. Предположим, что каждая особь популяции содержит по шесть состояний, они отмечены номерами от 0 до 5 (рис. 69).

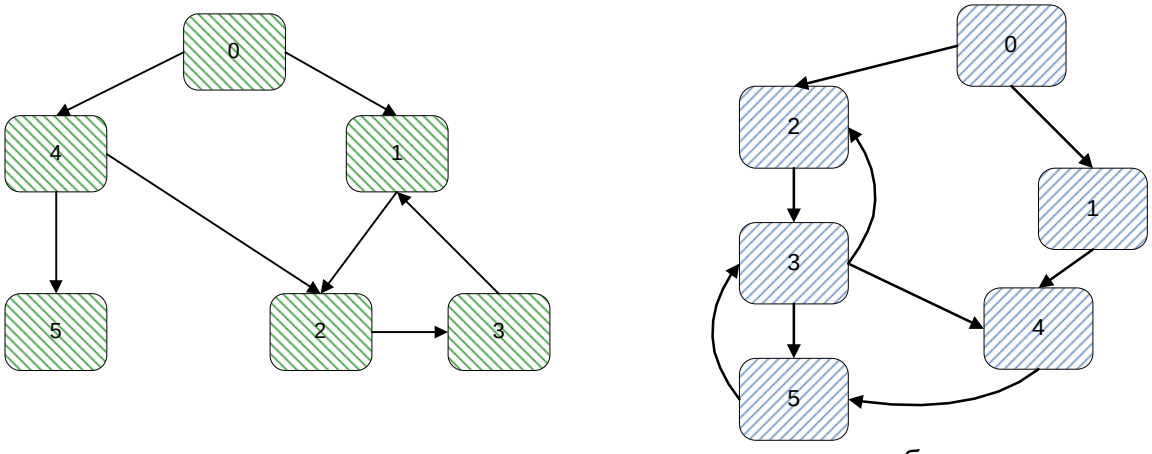


Рис. 69. Две особи, построенные в ходе генетического алгоритма

Каждая из таких особей не удовлетворяет всем *LTL*-формулам, но, помечая переходы в процессе верификации, можно получить некий подграф из переходов и состояний, на котором часть формул выполняется. Данный подграф перейдет в новую особь без изменений, а остальные переходы случайным образом перемешаются.

Такое скрещивание приведено на рис. 70, выделенная часть подграфа, образована помеченными переходами.

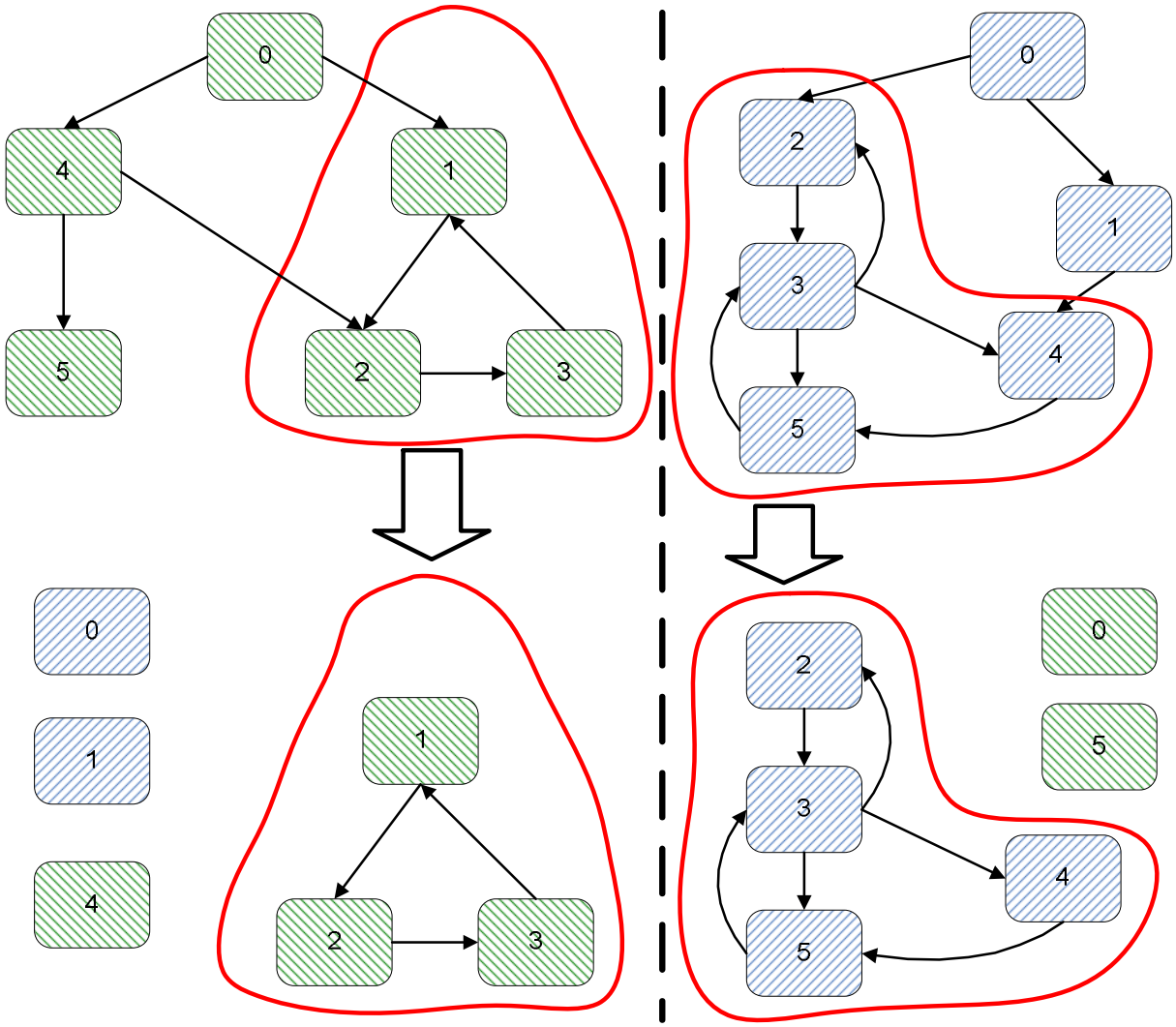


Рис. 70. Скрещивание с учетом результата верификации

В результате такого скрещивания, часть конечного автомата, на которой часть формул выполняется, сохранится, что позволит увеличить функцию приспособленности.

1.5.4. Методика построения автоматных программ

Для сравнения методов построения модели на основе тестов и на основе одновременно тестов и темпоральных свойств, сначала приведем методику построения автоматных программ только на основе тестов (рис. 71).

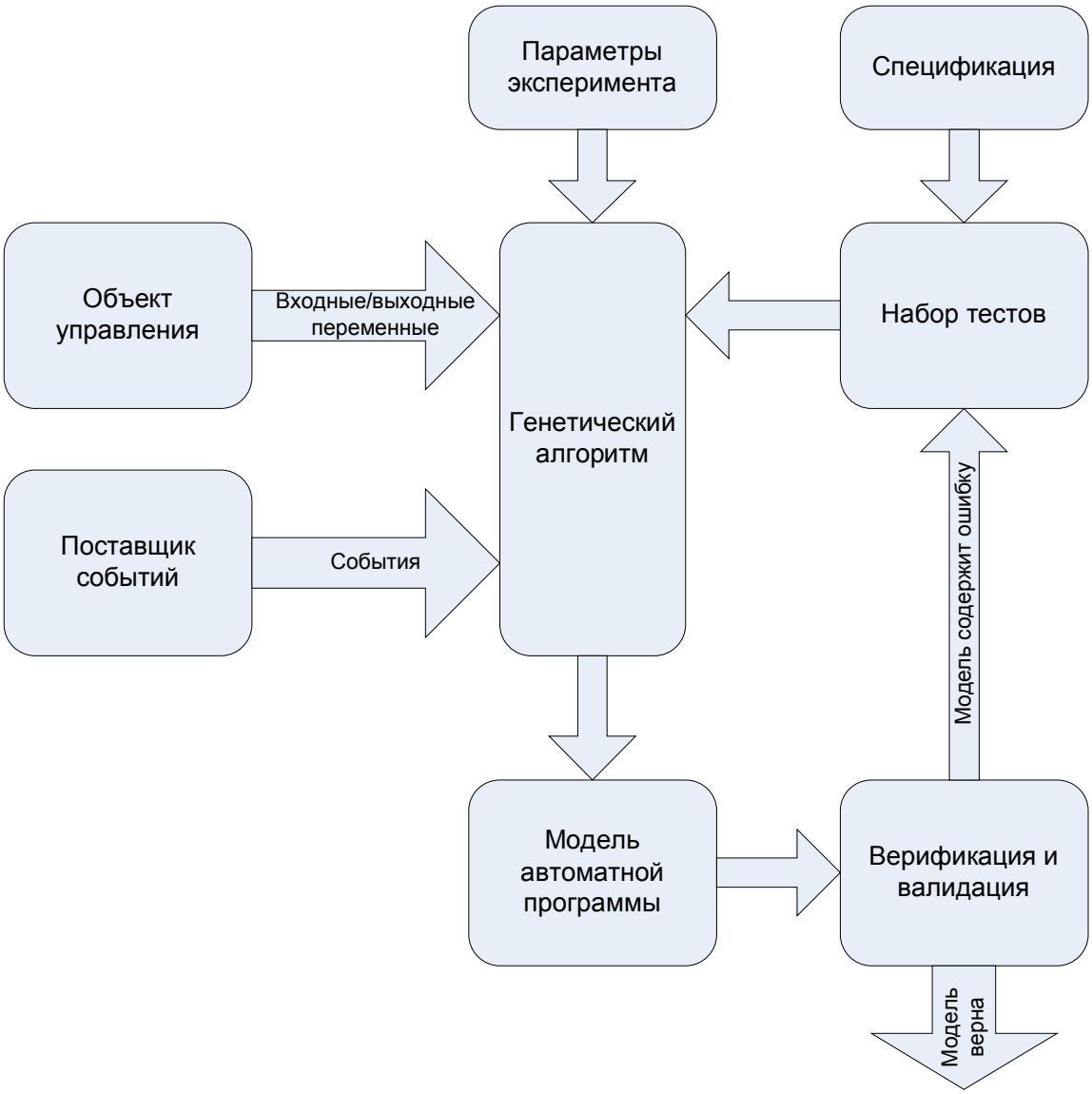


Рис. 71. Методика построения автоматных программ на основе тестов с помощью генетических алгоритмов

На вход генетическому алгоритму подаются списки входных переменных, выходных переменных и событий. Входные и выходные переменные берутся из «объекта управления», который либо реализован заранее, либо известны его методы. События, которые обрабатываются моделью, берутся из «поставщика событий», который также либо реализован заранее, либо известны только возможные события. В общем случае можно использовать несколько объектов управления и несколько поставщиков событий.

Далее по спецификации программы или из других априорных знаний о ее поведении на основе известных переменных и событий строится набор тестов. При записи тестов используются события и входные переменные в качестве входных данных теста (Input[i]), а выходные переменные, как эталонная последовательность выходных данных (Answer[i]).

На вход генетическому алгоритму также подаются параметры эксперимента, такие как размер популяции, вероятность мутации, число поколений до «большой» и «малой» мутации и т.д.

Далее генетический алгоритм строит автоматную модель. Отметим, что в этом методе генетический алгоритм одинаков для всех задач – для каждой новой задачи необходим только новый набор тестов, входных и выходных воздействий и, возможно, параметров алгоритма. Построенная модель проходит все тесты, однако мы не можем быть уверены в ее правильности, поэтому необходима дополнительная верификация и валидация.

Если модель оказалась верной, то ее можно использовать в реальной системе. Однако, если верификатор обнаружил несоответствие модели и спецификации, то необходимо добавлять новые тесты во входной набор и выполнять построение заново. В худшем случае, мы никогда не сможем построить корректную модель, так как нет гарантии, что такой цикл разработки модели когда-нибудь завершится.

Заметим, что мы можем вручную модифицировать модель, чтобы она стала соответствовать спецификации, однако это может оказаться сложнее, чем просто создание модели вручную.

В настоящей работе предлагается использовать верификацию на стадии создания программы. Методика работы предлагаемого метода представлена на рис. 72.

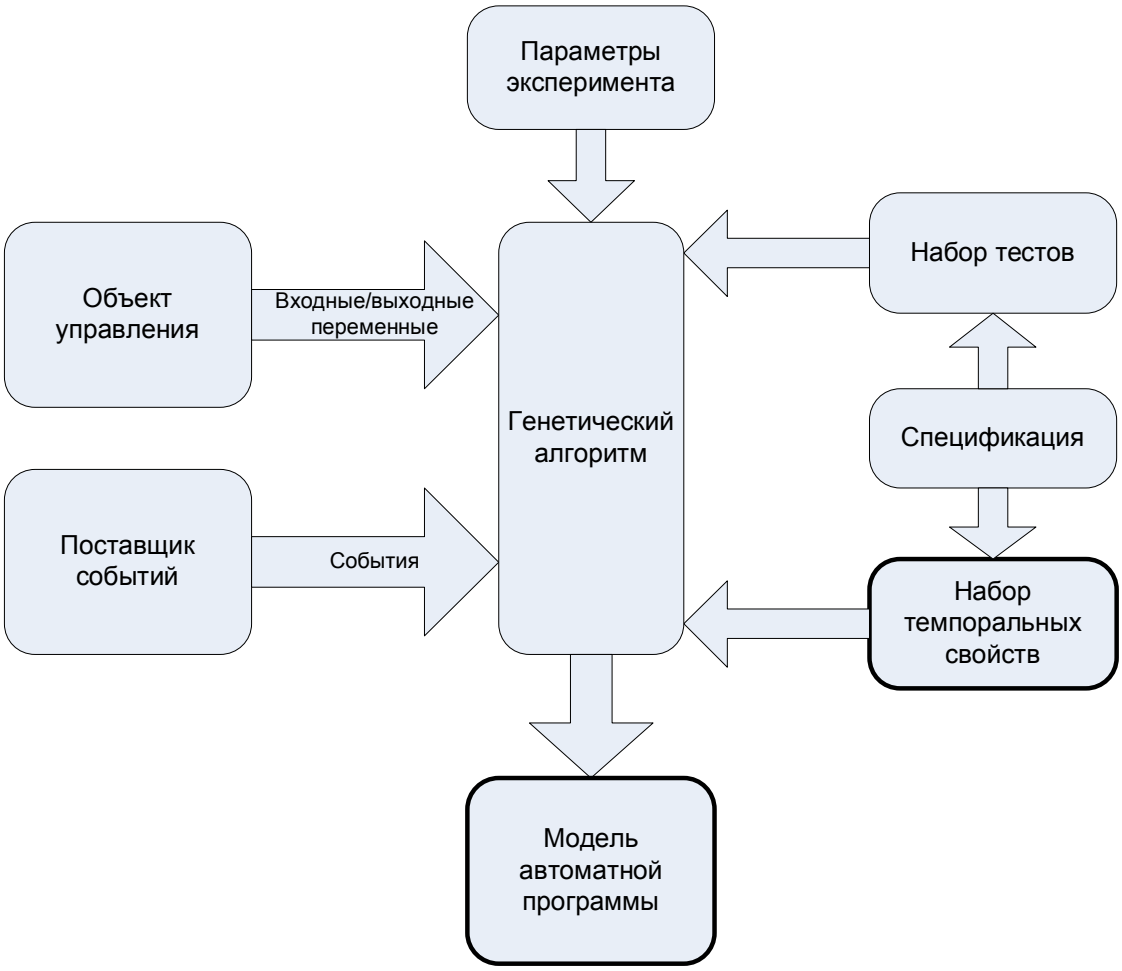


Рис. 72. Методика создания автоматных программ на основе тестов и темпоральных свойств

В отличие от метода, основанного только на тестах, входными данными являются также темпоральные свойства. Темпоральные свойства формулируются на основе спецификации и записываются в виде *LTL*-формул. Как видно из схемы, представленной на рис. 72, построенная

модель автоматной программы не требует дополнительной верификации, так как каждая особь в каждом поколении верифицировалась, и мы выбрали в качестве результата работы метода ту, у которой была максимальная функция приспособленности. Это означает, что она проходит все тесты и удовлетворяет всем темпоральным свойствам, а значит она соответствует спецификации и дополнительные проверки не требуются.

Таким образом, если набор тестов и набор теморальных свойств непротиворечивы, то автоматная модель будет построена, и она будет соответствовать заявленной спецификации.

1.5.5. Программная реализация метода и экспериментальное исследование

В настоящей главе описана программная реализация предлагаемого метода, и приводятся примеры построения управляющих конечных автоматов. Данный метод тестировался на двух примерах. Первый заключался в построении конечного автомата, управляющего часами с будильником. Во время второго эксперимента выполнялось построение конечного автомата, управляющего дверьми лифта.

1.5.5.1. Построение конечного автомата управления дверьми лифта

Предложенный метод также исследовался на задаче построения конечного автомата управления дверьми лифта [24]. Двери лифта умеют открываться и закрываться. Если при закрытии дверей появилось препятствие, то требуется прекратить их закрывание и открыть двери. Как и любой настоящий лифт, рассматриваемый лифт может сломаться в процессе открывания или закрывания дверей, и тогда необходим звонок в аварийную службу.

Заметим, что после поломки лифта не предусмотрено возвращение автомата в работоспособное состояние, то есть в модели не предусмотрено такое событие, как «ремонт окончен» (в этом случае предполагается, что программа просто будет перезапущена).

Конечный автомат лифта имеет пять входных событий:

- *e11* – открыть двери (нажата кнопка «открыть двери»);
- *e12* – закрыть двери (нажата кнопка «закрыть двери»);
- *e2* – двери успешно открыты или закрыты;
- *e3* – препятствие мешает закрытию дверей;
- *e4* – двери лифта сломались.

Кроме этого автомат имеет три выходных воздействия:

- *z1* – начать открывание дверей;
- *z2* – начать закрывание дверей
- *z3* – звонок в аварийную службу.

Поведение дверей лифта может быть описано конечным автоматом, взятым из работы [24] и построенным вручную (рис. 73).

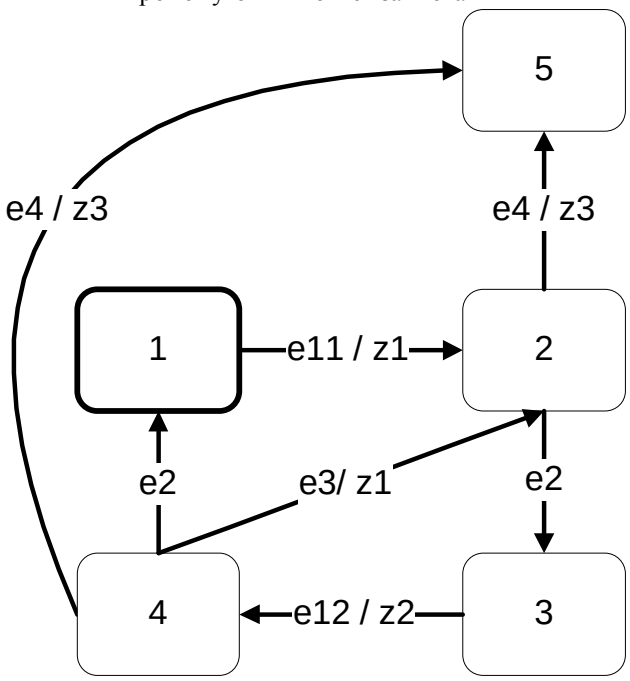


Рис. 73. Граф переходов автомата управления дверьми лифта, построенного на основе тестов и LTL-формул

Начальное состояние имеет номер «1» и выделено жирной рамкой.

1.5.5.2. Система тестовых примеров

В систему тестов для построения автомата управления дверьми лифта входят девять тестов. Они описывают различные варианты поведения дверей лифта и представлены в табл. 8.

Таблица 8. Тесты для автомата управления дверьми лифта

Тест	Комментарий
Input: *e11, e2, e12, e2* Answer: *z1, z2*	Описывает ситуацию открывания и закрывания дверей лифта.
Input: *e11, e2, e12, e2, e11, e2, e12, e2* Answer: *z1, z2, z1, z2*	Описывает процесс открывания и закрывания дверей лифта дважды.
Input: *e11, e2, e12, e3, e2, e12, e2* Answer: *z1, z2, z1, z2*	Описывает открывание дверей и появления препятствия при закрывании. Дверь закрывается со второго раза.
Input: *e11, e2, e12, e2, e11, e2, e12, e3, e2, e12, e2* Answer: *z1, z2, z1, z2, z1, z2*	Открывание и закрывание дверей лифта, закрывание, снова открывание, и при закрывании появление препятствия.
Input: *e11, e2, e12, e3, e2, e12, e3, e2, e12, e2* Answer: *z1, z2, z1, z2, z1, z2*	Процесс открывания дверей, при закрывании появление препятствия, вторая попытка закрывания и опять препятствие мешает закрыть дверь.
Input: *e11, e4* Answer: *z1, z3*	Описывает процесс возникновения поломки в процессе открывания дверей.
Input: *e11, e2, e12, e4* Answer: *z1, z2, z3*	Описывает процесс возникновения поломки в процессе закрывания дверей.
Input: *e11, e2, e12, e2, e11, e4* Answer: *z1, z2, z1, z3*	Описывает процесс возникновения поломки во время второго открывания дверей лифта.
Input: *e11, e2, e12, e3, e4* Answer: *z1, z2, z1, z3*	Описывает процесс возникновения поломки в момент открывания дверей из-за препятствия.

Совместно с тестами применялись 11 темпоральных свойств. Их описание приведено в табл. 9.

Таблица 9. Темпоральные свойства автомата управления дверьми лифта

Формула	Комментарий
G(wasEvent(e11) => wasAction(z1))	Если было событие e11, то было вызвано действие z1. В терминах модели утверждение можно записать как при обработке события «открыть двери» обязательно будет начато открывание дверей.
G(wasEvent(e12) <=> wasAction(z2))	Событие e12 обрабатывается тогда и только тогда, когда вызывается z2. В терминах модели: при нажатии кнопки закрывания дверей будет начато закрывание, и закрывание дверей может начаться только при нажатии кнопки закрыть двери.
G(wasEvent(e4) <=> wasAction(z3))	Событие e4 обрабатывается тогда и только тогда, когда вызывается z3. В терминах модели: звонок в аварийную службу будет произведен тогда и только тогда, когда лифт сломается.
G(wasEvent(e3) => wasAction(z1))	Если было событие e3, то было вызвано действие z1. Если препятствие мешает закрыть двери, то дверь начнет открываться.
G(wasEvent(e2) => X[wasEvent(e11) || wasEvent(e12)])	Если было событие e2, то следующим обработанным событием будет e11 или e12. В терминах модели: если дверь успешно открылась или закрылась, то следующее обработанное событие может быть только «открыть двери» или «закрыть двери».
G(wasEvent(e11) => X[wasEvent(e4) or wasEvent(e2)])	Если было событие e11, то следующим обработанным событием будет e4 или e2. В терминах модели: если была нажата кнопка «открыть двери», то следующее событие будет либо успешное открывание дверей, либо дверь сломается.
G(wasAction(z1) => X[wasEvent(e2) or wasEvent(e4)])	Если было вызвано действие z1, то следующим обработанным событием будет e2 или e4. В терминах модели: если дверь начала закрываться, то либо она успешно откроется, либо сломается.
G(wasEvent(e12) => X[wasEvent(e2) or wasEvent(e3) or wasEvent(e4)])	Если было событие e12, то следующим обработанным событием будет e2 или e3 или e4. В терминах модели: если нажали кнопку «закрыть двери», то либо двери закроются, либо препятствие помешает закрыть двери, либо лифт сломается.
G(wasAction(z1) => X[U(!wasAction(z1), wasAction(z2) or wasEvent(e4))])	Если было вызвано действие z1, то оно не будет больше вызвано, пока не будет вызвано z2 или событие z4.

G(wasAction(z2) => X[U(!wasAction(z2), wasAction(z1) or wasEvent(e4))])	Если было вызвано действие z2, то оно не будет больше вызвано, пока не будет вызвано z1 или событие z4. В терминах модели: если дверь начала закрываться, то она не будет снова закрываться до тех пор, пока она не будет открываться или не сломается.
!F(wasEvent(e4) and X(F(wasEvent(e11) || wasEvent(e12) || wasEvent(e2) || wasEvent(e3))))	Не верно, что в будущем будет после события e4 когда либо вызвано e11, e12, e2 или e3. В терминах модели: не верно, что после поломки лифта будут обработаны события «открыть двери», «закрыть двери», успешное открывание или закрывание дверей, препятствие мешает закрыть двери. Или, что то же самое, лифт не может быть починен.

Так же как и при построении автомата управления часами с будильником, при построении автомата, управляющего дверьми лифта, не получается использовать только темпоральные свойства. Но сразу заметим, что и применять только тестовые примеры не получается, так как по ним строится неправильный конечный автомат, о чем будет написано в следующем разделе.

1.5.5.3. Результаты эксперимента

Для сравнения двух способов построения конечных автоматов (на основе только тестов и на основе тестов и темпоральных свойств) было проведено два варианта экспериментов. Эксперимент каждого типа запускался по 1000 раз. Входные параметры каждого эксперимента были идентичными и эксперименты отличались только наличием или отсутствием *LTL*-формул.

Общие параметры экспериментов представлены в табл. 10.

Таблица 10. Параметры эксперимента построения автомата, управляющего дверьми лифта

Параметр эксперимента	Значение
Размер начальной популяции	2000
Число состояний у автоматов в начальном поколении	6
Ожидаемое число переходов	7
Доля особей, переходящих в следующее поколение. Остальные будут получены с помощью кроссовера	10%
Число поколений до «малой» мутации	70
Число поколений до «большой» мутации	100
Вероятность мутации особи	1%

В результате экспериментов, использующих только тестовые примеры в качестве входных данных, более чем в 99% случаях получался неправильный конечный автомат. Один из таких автоматов представлен на рис. 74, жирной рамкой выделено начальное состояние.

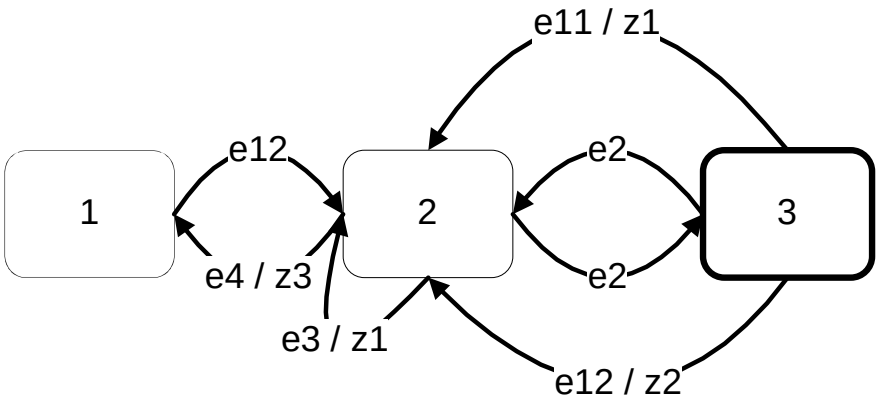


Рис. 74. Граф переходов автомата управления дверьми лифта, построенного только на основе тестов

Ошибка в представленном автомате заключается в том, что после поломки лифта дверь может снова начать закрываться, а затем лифт начнет функционировать как рабочий. Также данный автомат может повторно начать закрывать или открывать двери после закрытия или открытия соответственно.

Заметим, что формально мог построиться и автомат из одного состояния, все переходы которого являлись бы петлями. Такой автомат также проходил бы все тесты, но, естественно, был бы абсолютно неверным.

Указанных проблем можно избежать, применяя верификацию. При построении конечного автомата совместно на основе тестов и *LTL*-формул был построен правильный автомат, изоморфный автомату, изображенному на рис. 73.

Эксперименты производились без разбивки тестов и темпоральных свойств на группы. Функция приспособленности вычислялась по формуле $\mathrm{FF} = \mathrm{FF}_{test} + \mathrm{FF}_{test} \cdot \mathrm{FF}_{LTL} + \frac{1}{10 \cdot M} \cdot (M - \mathrm{cnt})$. В отличие от формулы из раздела 1.5.3.3, из вклада тестов ($\mathrm{FF}_{\mathrm{test}}$) удалена надбавка за прохождения всех тестов, так как автомат, проходящий все тесты, может не проходить все формулы. Но, в тоже время, прохождение тестов тоже важно, и автомат, не проходящий тесты, но удовлетворяющий всем формулам имеет нулевую функцию приспособленности. M было взято равным 100, и вклад числа переходов достигал максимума 0.093 на особях, проходящих все тесты и формулы.

Первый вид экспериментов заключался в построении автомата управления дверьми лифта только на основе тестов. Значение функции приспособленности построенного автомата было 1.093. В качестве численной оценки скорости работы алгоритма измерялось число вычислений функции приспособленности при нахождении автомата. Среднее значение вычислений функции приспособленности оказалось равным 7.479×10^4. Минимальное число вычислений – 2.184×10^4. Максимальное число – 2.999×10^5. Среднеквадратичное отклонение – 2.54×10^4. Плотность распределения вероятности числа вычислений функции приспособленности при построении автомата только на основе тестов представлена на рис. 75.

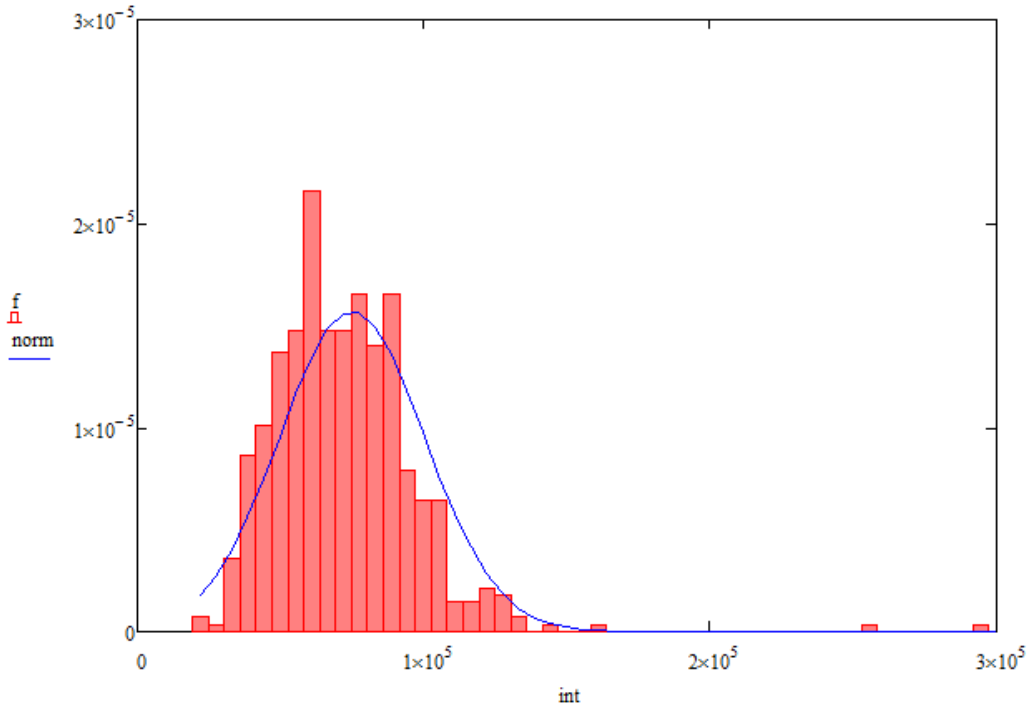


Рис. 75. Плотность распределения вероятности числа вычислений функции приспособленности при построении автомата управления дверьми лифта только на основе тестов

Второй эксперимент заключался в построении автомата, управляющего дверьми лифта на основе тестов и темпоральных свойств. Темпоральные свойства были представлены *LTL*-формулами из предыдущего раздела. Значение функции приспособленности построенного автомата – 2.093. Среднее значение вычислений функции приспособленности оказалось равным 7.246×10^5. Минимальное число вычислений – 7.054×10^4. Максимальное число – 5.492×10^6. Среднеквадратичное отклонение – 7.729×10^5. Плотность распределения вероятности числа вычислений функции приспособленности при построении автомата на основе тестов и *LTL*-формул представлена на рис. 76.

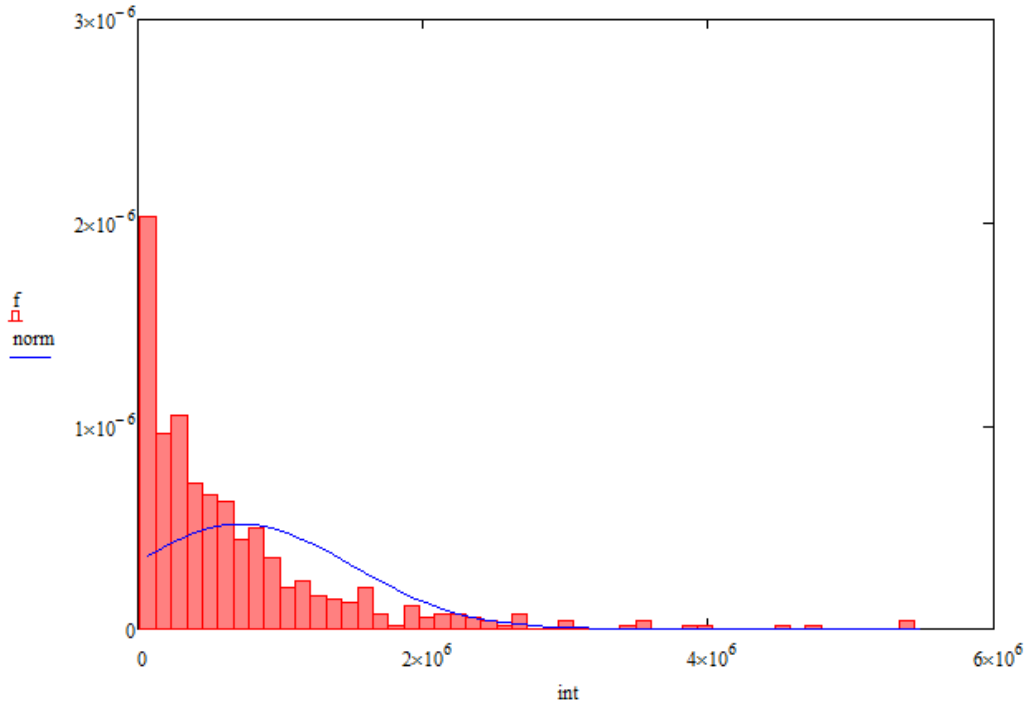


Рис. 76. Плотность распределения вероятности числа вычислений функции приспособленности при построении автомата управления дверьми лифта на основе тестов и LTL-формул

Число вычислений функции приспособленности для экспериментов с *LTL*-формулами оказалось больше практически в 10 раз. Однако, из 1000 построений конечного автомата только на основе тестов, только 9 не содержали ошибок.

2. РЕЗУЛЬТАТЫ ТЕОРЕТИЧЕСКИХ И ЭКСПЕРИМЕНТАЛЬНЫХ ИССЛЕДОВАНИЙ

Исследования, проводимые, например, в университетах Йорка, Брунеля (Великобритания), в исследовательском центре *British Telecom*, в университете Дрекселя (Филадельфия, США), в университете страны басков (Испания), показывают, что методы поиска в условиях ограничений могут успешно применяться для решения ряда задач, возникающих при разработке программных систем. В число таких задач входят: автоматическое построение тестов, разбиение программ на модули, прогнозирование времени отклика. Разработанный в рамках НИР метод применения верификации моделей программ при построении управляющих конечных автоматов с помощью генетического программирования позволяет применять методы поиска в условиях ограничений для построения корректных автоматных программ, удовлетворяющих формальным требованиям.

Одним из важнейших направлений искусственного интеллекта является создание роботов. Исследования по этому направлению ведутся в большом числе научных центров и университетов во всем мире. В большинстве работ методы искусственного интеллекта применяются либо при программной реализации управляющей системы, либо для решения задач обеспечения процесса разработки. В настоящей работе делается попытка распространить область применения методов искусственного интеллекта на задачи автоматизированного построения управляющих программных систем. При этом в качестве предметной области выбрано построение систем управления малоразмерными мобильными роботами. В этом направлении в рамках НИР получены новые результаты, не уступающие мировым аналогам, такие как синтез эффективных систем управления виртуальными роботами для упрощенной модели игры в футбол, а также автоматическое построение системы верхнего уровня, решающей некоторые часто встречающиеся в робототехнике задачи навигации.

Изложенное позволяет утверждать, что результаты выполнения второго этапа научно-исследовательской работы превышают мировой уровень разработок в рассматриваемой области.

Результаты НИР могут быть использованы в реальном секторе экономики двумя способами. Во-первых, возможно создание инструментального средства, реализующего разработанный метод применения верификации моделей программ при построении управляющих конечных автоматов с помощью генетического программирования. Такое инструментальное средство может быть использовано как для решения задач автоматизации построения автоматных программ с одновременной их формальной верификацией, так и для построения управляющих конечных автоматов для других целей. При таком варианте использования будет достигнуто сокращение затрат и влияния человеческого фактора на процесс разработки программ на основе автоматного подхода, а также повышения качества получаемых программ, так как они будут автоматически верифицированы.

Второе направление использования заключается в интеграции разработанного метода совместного применения конечных автоматов и нейронных сетей в системы управления, построенные на основе автоматного подхода. Данный метод может быть применен для адаптивного контроля процесса работы системы, что улучшит надежность управления системой и повысит ее отказоустойчивость.

Результаты НИР могут быть использованы при создании научно-образовательных курсов двумя способами. Во-первых, разработанные методы применения коэволюции для построения конечных автоматов с помощью генетических алгоритмов, а также метод совместного применения конечных автоматов и нейронных сетей могут быть использованы при разработке лабораторных работ по дисциплинам, связанным с автоматным программированием, искусственным интеллектом и машинным обучением.

Во-вторых, разработанный метод применения верификации моделей программ при построении управляющих конечных автоматов с помощью генетического программирования может быть

использован в учебном процессе при преподавании дисциплин, связанных с автоматным программированием, искусственным интеллектом и программной инженерии.

3. ПУБЛИКАЦИЯ РЕЗУЛЬТАТОВ НИР

По результатам НИР были опубликованы следующие статьи:

- К. В. Егоров, Ф. Н. Царев, А. А. Шалыто. Применение генетического программирования для построения автоматов управления системами со сложным поведением на основе обучающих примеров и спецификации // Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики, №5 (69), 2010, с. 81–86.
- С. И. Попов, Ю. И. Попов, А. А. Шалыто. Задача о муравьеде и муравьях // Информационные технологии, №8, 2010, с. 18–22.
- Д. И. Суясов. Выделение структурных признаков изображений символов на основе клеточных автоматов с метками // Информационно-управляющие системы, №4, 2010, с. 39–45.

Копии текстов указанных статей приведены в приложении 1.

Кроме этого, по результатам НИР сделаны доклады на следующих конференциях:

- 32 конференция молодых ученых и специалистов Института проблем передачи информации им. А. А. Харкевича РАН «Информационные технологии и системы» (ИТИС`09). М.: ИППИ, 15–18 декабря 2009. Доклад – Егоров К. В., Царев Ф. Н. Совместное применение генетического программирования и верификации моделей для построения автоматов управления системами со сложным поведением /Сборник трудов конференции «Информационные технологии и системы» (ИТИС`09). М.: Институт проблем передачи информации им. А. А. Харкевича РАН. 2009, с. 77–82;
- Егоров К. В., Царев Ф. Н. Применение генетического программирования для построения автоматов управления системами со сложным поведением на основе обучающих примеров и спецификации /VII Всероссийская межвузовская конференция молодых ученых. 20–23 апреля 2010 года, СПбГУ ИТМО.
- Буздалов М. В. Генерация конечных автоматов с помощью генетических алгоритмов для решения задач навигации /VII Всероссийская межвузовская конференция молодых ученых. 20–23 апреля 2010 года, СПбГУ ИТМО.
- Законов А.Ю. Применение генетических алгоритмов к генерации тестов для автоматных программ /VII Всероссийская межвузовская конференция молодых ученых. 20–23 апреля 2010 года, СПбГУ ИТМО.
- Попов С. И., Попов Ю. И., Шалыто А. А. Задача о муравьеде и муравьях / Сборник статей третьей Всероссийской научной конференции «Нечеткие системы и мягкие вычисления». Том II. Волгоград: 2009 г., с. 57–63.
- Буздалов М.В., Парфенов В.Г. Генерация конечных автоматов с помощью генетических алгоритмов для решения задач навигации /Труды XVII Всероссийской научно-методической конференции «Телематика`2010». Т. 2. СПбГУ ИТМО. 2010, с. 343–344.
- Егоров К.В., Парфенов В.Г., Царев Ф.Н. Совместное применение генетического программирования и верификации моделей для построения автоматов управления системами со сложным поведением /Труды XVII Всероссийской научно-методической конференции «Телематика`2010». Т. 2. СПбГУ ИТМО. 2010, с. 344–345.
- Zakonov A., Stepanov O., Shalyto A.A. A GA-Based Approach for Test Generation for Automata-Based Programs /Proceeding of the 4 Spring/Summer Young Researcher`s Colloquium on Software Engineering» (SYRCoSE 2010). Nizhny Novgorod. 2010, pp.37–42.
- Zakonov A., Stepanov O., Shalyto A. GA-based and Design by Contract Approach to Test Generation for EFSMs /Proceedings of IEEE East-West Design & Test Symposium (EWDTS'10). 2010, pp.152–155.

В процессе выполнения НИР также получены свидетельства о регистрации программы для ЭВМ №№ 2010614264, 2010615014 (копии свидетельств приведены в приложении 2). Копия заключения экспертной комиссии о возможности опубликования настоящего отчета приведена в приложении 3.

ЗАКЛЮЧЕНИЕ

В результате исследований, выполненных на втором этапе работ по контракту, были проведены следующие работы:

- разработка, программная реализация и экспериментальное исследование метода применения генетического программирования для построения систем управления виртуальными роботами для упрощенной модели игры в футбол;
- разработка, программная реализация и экспериментальное исследование метода применения коэволюции для построения взаимодействующих управляющих конечных автоматов;
- разработка, программная реализация и экспериментальное исследование метода применения коэволюции для построения управляющих конечных автоматов, решающих задачи навигации;
- разработка, программная реализация и экспериментальное исследование метода совместного применения конечных автоматов и нейронных сетей;
- разработка метода применения верификации моделей программ при построении управляющих конечных автоматов с помощью генетического программирования.

Кроме этого, по результатам НИР были опубликованы три статьи в журналах «Информационные технологии», «Информационно-управляющие системы» и «Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики» (входят в перечень ВАК) и сделано четыре доклада на трех конференциях.

Результаты выполненных работ позволяют утверждать, что научно-технический уровень исследований превышает уровень исследований в рассматриваемой области, проводимых в лучших исследовательских центрах мира.

ИСТОЧНИКИ

1. *Бедный Ю. Д., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов в задаче «Умный муравей». СПбГУ ИТМО, 2007. http://is.ifmo.ru/works/ant
2. *Буч Г., Рамбо Д., Джекобсон А.* Язык UML. Руководство пользователя. — 2-е изд. — М., СПб.: ДМК Пресс, Питер, 2004.
3. *Горбатов В. А., Горбатов А. В., Горбатова М. В.* Теория автоматов. Астрель, 2008.
4. *Гладков Л. А., Курейчик В. В., Курейчик В. М.* Генетические алгоритмы. М.: Физматлит. 2006.
5. *Гуров В. С., Мазин М. А., Нарвский А. С., Шалыто А. А.* UML. SWITCH-технология. Eclipse // Информационно-управляющие системы. 2004. № 6. с. 12–17.
6. *Давыдов А. А., Соколов Д. О., Царев Ф. Н., Шалыто А. А.* Виртуальная лаборатория обучения генетическому программированию для генерации управляющих конечных автоматов / Сборник докладов III Международной научно-практической конференции «Современные информационные технологии и ИТ-образование». ВМК МГУ. М.: МАКС Пресс. 2008, с. 179-183. http://is.ifmo.ru/works/_2_93_davidov_sokolov.pdf
7. *Данилов В. Р.* Технология генетического программирования для генерации автоматов управления со сложным поведением. СПбГУ ИТМО, 2007. Бакалаврская работа. http://is.ifmo.ru/papers/danilov_bachelor
8. *Джонс М.* Программирование искусственного интеллекта в приложениях. М.: ДМК Пресс, 2006.
9. *Егоров К. В., Шалыто А. А.* Методика верификации автоматных программ // Информационно-управляющие системы. СПб: Политехника, 2008, № 5, с. 15–21.
10. *Елкин Д. И., Скорынин П. А., Шалыто А. А.* Применение алгоритма имитации отжига для построения автомата управления виртуальным роботом-футболистом. СПбГУ ИТМО, 2009.
11. *Емельянов В. В., Курейчик В. В., Курейчик В. М.* Теория и практика эволюционного моделирования. М.: ФИЗМАТЛИТ, 2003.
12. Интернет-ресурс http://www.robocup.org/ – сайт, посвященный международной исследовательской и образовательной инициативе RoboCup.
13. История развития компьютерных шахмат. http://www.gambiter.ru/index.php?option=com_content&task=view&id=103&Itemid=9
14. *Кларк Э., Грамберг О., Пелед Д.* Верификация моделей программ: Model Checking. М.: МЦНМО, 2002. 416 с.
15. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: построение и анализ. М.: МЦНМО, 2000.
16. *Кретинин А. В., Солдатов Д. В., Шалыто А. А., Шостак А. В.* Ракеты. Автоматы. Нейронные сети. //Нейрокомьютеры: разработка и применение, №5, 2005г. с.50-59.
17. *Левенштейн В. И.* Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академии Наук СССР 163.4, с. 845–848.
18. *Лобанов П. Г., Шалыто А. А.* Использование генетических алгоритмов для автоматического построения конечных автоматов в задаче о флибах //Известия РАН. Теория и системы управления. 2007, №5, с. 127–136.
19. Научно-технический отчет о выполнении Государственного контракта по теме "Разработка методов машинного обучения на основе генетических алгоритмов для построения управляющих конечных автоматов" (этап 1). СПбГУ ИТМО. 2009.
20. *Петросян Л. А. и др.* Теория игр: Учебное пособие для университетов. – М: Высшая школа, Книжный дом «Университет», 1998.

21. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб.: Питер, 2009. http://is.ifmo.ru/books/_book.pdf
22. *Поликарпова Н. И., Точилин В. Н., Шалыто А. А.* Метод сокращенных таблиц для генерации автоматов с большим числом входных переменных на основе генетического программирования //Известия РАН. Теория и системы управления. 2010. № 2, с.100 – 117.
23. *Попов С. И., Попов Ю. И., Шалыто А. А.* Задача о муравьеде и муравьях / Сборник статей третьей Всероссийской научной конференции «Нечеткие системы и мягкие вычисления». Том II. Волгоград: 2009, с. 57 – 63.
24. Разработка технологии верификации управляющих программ со сложным поведением, построенных на основе автоматного подхода. Второй этап. СПбГУ ИТМО, 2007. 105 с. http://is.ifmo.ru/verification/_2007_02_report-verification.pdf
25. *Рутковская Д., Пилиньский М., Рутковский Л.* Нейронные сети, генетические алгоритмы и нечеткие системы. М.: Горячая линия-Телеком. 2008.
26. *Сивухин Д. В.* Общий курс физики. — Издание 4-е. — М.: Физматлит, 2002.
27. *Хопкрофт Дж., Мотвани Р., Ульман Дж.* Введение в теорию автоматов, языков и вычислений. М.: Вильямс, 2002.
28. *Шалыто А. А.* Switch-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
29. *Angeline P. J., Pollack J.* Evolutionary Module Acquisition // Proceedings of the Second Annual Conference on Evolutionary Programming. 1993. http://www.demo.cs.brandeis.edu/papers/ep93.pdf
30. *Barricelli N. A.* Esempi numerici di processi di evoluzione / Methodos. 1954, p. 45–68.
31. *Chambers L.* Practical Handbook of Genetic Algorithms. Complex Coding Systems. Volume III. CRC Press, 1999.
32. *Courcoubetis C., Vardi M., Wolper P., Yannakakis M.* Memory-Efficient Algorithms for the Verification of Temporal Properties / Formal Methods in System Design. 1992, pp. 275–288.
33. Deutsche Bank. Алгоритмические системы и финансовые рынки. http://www.gambiter.ru/index.php?option=com_content&task=view&id=103&Itemid=9
34. *Fraser A. S.* Simulation of Genetic Systems by Automatic Digital Computers. I. Introduction / Australian Journal of Biological Sciences 10. 1957, p. 484 – 491.
35. *Fogel L. J.* Biotechnology: Concepts and Applications. Prentice Hall. 1963.
36. *Gastin P., Oddoux D.* Fast LTL to Büchi Automata Translation / 13th Conference on Computer Aided Verification (CAV'01). 2001, pp. 53–65.
37. *Gerth R., Peled D., Vardi M. Y.,Wolper P.* Simple On-the-fly Automatic Verification of Linear Temporal Logic / Proc. of the 15th Workshop on Protocol Specification, Testing, and Verification. Warsaw. 1995, pp. 3–18.
38. *Hillis W. D.* Co-evolving parasites improve simulated evolution as an optimization procedure. Phys. D Vol. 42, Issue 1-3 (Jun. 1990), pp. 228–234.
39. *Hoffman L.* Talking Model-Checking Technology // Communications of the ACM. 2008. V. 51. № 7, pp. 110–112.
40. *Holland John H.* Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor. 1975.
41. *Jefferson D., Collins R., Cooper C., Dyer M., Flowers M., Korf R., Taylor C., Wang A.* The Genesys System. 1992. http://www.cs.ucla.edu/~dyer/Papers/AlifeTracker/Alife91Jefferson.html
42. *Kamon I, Rimon E, Rivlin E.* A New Range-Sensor Based Globally Convergent Navigation Algorithm for Mobile Robots. CIS – Center of Intelligent Systems 9517, Computer Science Dept., Technion, Israel, 1995.

43. *Koza J.* Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press. 1998.
44. *Liu Y. H., Arimoto S.* Path planning using a tangent graph for mobile robots among polygonal and curved obstacles. International Journal of Robotic Research, 11(4):376–382, 1992.
45. LTL2BA project. http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/
46. *Lumelsky V. J., Stepanov A. A.* Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. Algorithmica, 2: 403–430, 1987.
47. *Lumelsky V. J., Skewis T.* Incorporating range sensing in the robot navigation function. IEEE Transactions on Systems, Man, and Cybernetics, 20(5):1058–1068, 1990.
48. *Mitchell M.* An Introduction to Genetic Algorithms. MA: The MIT Press, 1999.
49. *Rechenberg I.* Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution (PhD thesis). 1971.
50. *Schwefel H.-P.* Cybernetic Evolution as Strategy for Experimental Research in Fluid Mechanics (in German). Diploma Thesis. Hermann Föttinger-Institute for Fluid Mechanics, Technical University of Berlin, March 1965.

ПРИЛОЖЕНИЕ 1. КОПИИ ТЕКСТОВ ОПУБЛИКОВАННЫХ СТАТЕЙ

В данном приложении приведены копии следующих статей, опубликованных в рамках выполненных на втором этапе работ по контракту:

- К. В. Егоров, Ф. Н. Царев, А. А. Шалыто. Применение генетического программирования для построения автоматов управления системами со сложным поведением на основе обучающих примеров и спецификации // Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики, №5 (69), 2010, с. 81–86.
- С. И. Попов, Ю. И. Попов, А. А. Шалыто. Задача о муравьеде и муравьях // Информационные технологии, №8, 2010, с. 18–22.
- Д. И. Суясов. Выделение структурных признаков изображений символов на основе клеточных автоматов с метками // Информационно-управляющие системы, №4, 2010, с. 39–45.

К.В. Егоров, Ф.Н. Царев, А.А. Шалыто

5 КОМПЬЮТЕРНЫЕ СИСТЕМЫ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

УДК 004.4'242

ПРИМЕНЕНИЕ ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ ДЛЯ ПОСТРОЕНИЯ АВТОМАТОВ УПРАВЛЕНИЯ СИСТЕМАМИ СО СЛОЖНЫМ ПОВЕДЕНИЕМ НА ОСНОВЕ ОБУЧАЮЩИХ ПРИМЕРОВ И СПЕЦИФИКАЦИИ

К.В. Егоров, Ф.Н. Царев, А.А. Шалыто

Предлагается метод машинного обучения, основанный на совместном применении генетического программирования и верификации моделей для построения автоматов управления системами со сложным поведением на основе обучающих примеров. Приводится описание структуры хромосом, генетического алгоритма, операций мутации и скрещивания. Изложены результаты экспериментального исследования на задаче построения конечного автомата управления дверьми лифта.

Ключевые слова: генетическое программирование, машинное обучение, верификация моделей, автоматное программирование.

Введение

Автоматное программирование – это парадигма программирования, в рамках которой программы предлагается проектировать в виде совокупности взаимодействующих автоматизированных объектов управления [1]. В автоматных программах выделяют три типа объектов: поставщики событий, система управления и объекты управления. Система управления представляет собой конечный автомат или систему взаимодействующих конечных автоматов. Поставщики событий генерируют события, а система управления по каждому событию может совершить переход из рассматриваемого состояния, считывая значения входных переменных у объектов управления для проверки условия перехода, в другое состояние.

Для многих задач автоматы удается строить эвристически, однако существуют задачи, для которых такое построение затруднительно [2–4]. Одним из авторов настоящей работы был предложен метод построения автоматов с помощью генетического программирования на основе тестов (обучающих примеров) [5]. Однако, как известно, тесты не могут полностью описывать поведение программы, а их выполнимость не является критерием ее корректности.

Цель настоящей работы – расширение возможностей указанного метода построения автоматных программ за счет использования обучающих примеров и верификации в процессе работы алгоритма генетического программирования.

Верификация автоматных программ

Для описания спецификации управляющего конечного автомата будем применять язык логики линейного времени LTL (Linear Temporal Logic). Алгоритм верификации основан на проверке пустоты языка, задаваемого пересечением рассматриваемого конечного автомата и автомата Бюхи, соответствующего отрицанию LTL-формулы, представляющей требование к автомату [6, 7]. Эта проверка осуществляется с помощью алгоритма двойного обхода в глубину.

Верификатор получает на вход модель автоматной программы и LTL-формулу [8]. После проверки модели верификатор либо сообщает, что формула выполняется, либо приводит контрпример – путь в модели, опровергающий утверждение [10].

Построение управляющих конечных автоматов на основе обучающих примеров с помощью генетического программирования

При использовании метода построения управляющих конечных автоматов на основе обучающих примеров, каждый из них содержит последовательность входных событий (входная последовательность, соответствующая i-ому тесту, будет обозначаться как Input[i]) и соответствующую ей последовательность выходных воздействий (в дальнейшем будет обозначаться, как Answer[i]). Отметим, что в настоящей работе в обучающих примерах не используются входные переменные.

Функция приспособленности основана на редакционном расстоянии [9]. Для этой функции выполняются следующие действия: на вход автомату подается каждая из последовательностей Input[i].

Обозначим последовательность выходных воздействий, которую сгенерировал автомат на входе Input[i] как Output[i]. После этого вычисляется функция FF_1 вида:

$$FF_1 = \frac{\sum_{i=1}^{n}(1 - \frac{ED(Output[i], Answer[i])}{\max(|Output[i]|, |Answer[i]|)})}{n}.$$

Здесь ED(A, B) – редакционное расстояние между строками A и B, а Answer[i] – эталонная выходная последовательность, которую должен генерировать автомат на входе Input[i]. Отметим, что значения этой функции лежат в пределах от 0 до 1. При этом чем «лучше» автомат соответствует тестам, тем больше значение функции приспособленности.

Функция приспособленности должна зависеть не только от того, насколько «хорошо» автомат работает на тестах, но и числа переходов, которые он содержит. Предлагается ее вычислять по формуле

$$FF_2 = FF_1 + \frac{1}{M} \cdot (M - cnt),$$

где cnt – число переходов в рассматриваемом конечном автомате, а M – некоторое число, большее максимально возможного числа переходов в автомате с заданным числом состояний. При этом отметим, что все рассматриваемые в процессе работы алгоритма генетического программирования автоматы имеют одинаковое наперед заданное число состояний.

Функция приспособленности построена так, что при одинаковом значении функции FF_1, учитывающей «прохождение» тестов автоматом, преимущество имеет автомат, содержащий меньшее число переходов. Учет числа переходов в функции приспособленности необходим, так как минимизация их числа приводит к тому, что в результирующем автомате отсутствуют переходы, неиспользуемые в тестах.

Совместное применение генетического программирования и верификации

Предлагается при вычислении функции приспособленности учитывать как поведение автомата при обработке тестов, так и число выполняемых (верных) для автомата LTL-формул, составляющих спецификацию. При этом, чем больше число выполняемых формул и успешно пройденных тестов, тем больше значение функции приспособленности.

Для вычисления функции приспособленности конечный автомат, задаваемый рассматриваемой особью, запускается на всех тестах и проверяется на соответствие всем темпоральным формулам, составляющим спецификацию. Для учета указанных особенностей необходимо изменить введенную выше функцию приспособленности:

$$FF = FF_1 \cdot (1 + \frac{n_1}{n_2}) + \frac{1}{M} \cdot (M - cnt).$$

Здесь n_2 – общее число темпоральных формул в спецификации, а n_1 – число формул, которые выполняются для рассматриваемого конечного автомата.

Структура хромосомы в алгоритме генетического программирования

Конечный автомат в алгоритме генетического программирования представляется в виде объекта, который содержит описания переходов для каждого состояния и номер начального состояния. Для каждого состояния хранится список переходов. В свою очередь, каждый переход описывается событием, при поступлении которого этот переход выполняется, и числом выходных воздействий, которые должны быть сгенерированы при выборе этого перехода. Таким образом, в особи кодируется только «скелет» управляющего конечного автомата, а конкретные выходные воздействия, вырабатываемые на переходах, определяются с помощью алгоритма расстановки пометок, который аналогичен предложенному в работе [11]. Идея алгоритма расстановки пометок состоит в том, что пометки на переходах (вырабатываемые на них выходные воздействия) расставляются на основе тестов. При этом расстановка пометок происходит таким образом, чтобы получившийся в результате автомат достаточно хорошо соответствовал тестам.

Опишем более формально алгоритм расстановки пометок на переходах, применяемый в настоящей работе. Подадим на вход конечному автомату последовательность событий, соответствующую одному из тестов, и будем наблюдать за тем, какие переходы выполняет автомат. Зная эти переходы и информацию о том, сколько выходных воздействий должно быть сгенерировано на каждом переходе, можно определить, какие выходные воздействия должны вырабатываться на переходах, использовавшихся при обработке входной последовательности (рис. 1).

На рис. 1 буквами A, H, M и T обозначены события, поступающие на вход конечного автомата, а как z3, z4 и z5 обозначены выходные воздействия.

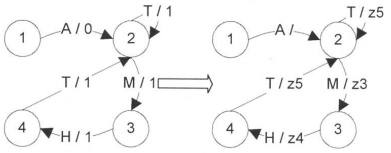


Рис. 1. Применение алгоритма расстановки пометок

Для случая нескольких тестов этот принцип можно обобщить следующим образом. Для каждого перехода Tr и каждой последовательности выходных воздействий zs вычисляется величина C[Tr][zs] – число раз, когда при обработке входной последовательности, соответствующей одному из тестов, на переходе Tr должны быть выработаны выходные воздействия, образующие последовательность zs. Далее, каждый переход помечается той последовательностью zs_0, для которой величина C[Tr][zs_0] максимальна.

Операции мутации и скрещивания

Операция мутации может выполняться двумя способами – традиционным и учитывающим результат верификации. Традиционный способ используется в методе построения управляющих автоматов на основе обучающих примеров [5].

Операция скрещивания может быть осуществлена тремя способами – традиционным, с учетом тестов и с учетом результата верификации. Первые два способа также описаны в работе [5].

Опишем методы мутации и скрещивания, учитывающие верификацию. Как отмечалось выше, алгоритм верификации основан на двойном обходе в глубину автомата, задаваемого особью, и автомата Бюхи, построенного по отрицанию LTL-формулы. При использовании такого алгоритма, та часть модели, которая была помещена в процессе первого обхода в глубину, удовлетворяет LTL-формуле и может быть использована в процессе скрещивания точно так же, как в методе скрещивания с учетом тестов (помеченные переходы копируются в новые особи напрямую). Иными словами, подграф переходов, которые обошел верификатор в процессе верификации, может перейти без изменений в новую особь. В то же время, должна быть обеспечена возможность не только сохранять часть модели, на которой выполняется темпоральное свойство, но и удалять те переходы, которые входят в контрпример, возвращаемый верификатором. Такой контрпример представляет собой путь в модели. Поэтому при мутации можно либо удалить переход из этого пути, либо изменить его конечное состояние, число генерируемых выходных воздействий или событие, инициирующее переход.

Экспериментальное исследование

Экспериментальное исследование предлагаемого метода машинного обучения проводилось на задаче построения автомата управления дверьми лифта. Эта система содержит пять входных событий (`e11` – нажата кнопка «Открыть двери»; `e12` – нажата кнопка «Закрыть двери»; `e2` – открытие или закрытие дверей успешно завершено; `e3` – препятствие мешает закрыть дверь; `e4` – дверь сломалась) и три выходные воздействия (`z1` – начать открытие дверей; `z2` – начать закрытие дверей; `z3` – позвонить в аварийную службу).

При построении управляющего автомата использовались девять тестов (табл. 1).

Входная последовательность	Выходная последовательность
e11, e2, e12, e2	z1, z2
e11, e2, e12, e2, e11, e2, e12, e2	z1, z2, z1, z2
e11, e2, e12, e3, e2, e12, e2	z1, z2, z1, z2
e11, e2, e12, e2, e11, e2, e12, e3, e2, e12, e2	z1, z2, z1, z2, z1, z2
e11, e2, e12, e3, e2, e12, e3, e2, e12, e2	z1, z2, z1, z2, z1, z2
e11, e4	z1, z3
e11, e2, e12, e4	z1, z2, z3
e11, e2, e12, e2, e11, e4	z1, z2, z1, z3
e11, e2, e12, e3, e4	z1, z2, z1, z3

Таблица 1. Тесты для системы управления дверьми лифта

Спецификация управляющего автомата содержит 11 темпоральных свойств (табл. 2).

Формула	Комментарий
`G( wasEvent(ep.e11) => wasAction(co.z1) )`	Если было событие e11, то было вызвано действие z1
`G( wasEvent(ep.e12) <=> wasAction(co.z2) )`	Событие e12 обрабатывается тогда и только тогда, когда вызывается z2
`G( wasEvent(ep.e4) <=> wasAction(co.z3))`	Событие e4 обрабатывается тогда и только тогда, когда вызывается z3
`G( wasEvent(ep.e3) => wasAction(co.z1) )`	Если было событие e3, то было вызвано действие z1
`G( wasEvent(ep.e2) => X(wasEvent(ep.e11) or wasEvent(ep.e12)) )`	Если было событие e2, то следующим обработанным событием будет e11 или e12
`G( wasEvent(ep.e11) => X(wasEvent(ep.e4) or wasEvent(ep.e2)) )`	Если было событие e11, то следующим обработанным событием будет e4 или e2
`G( wasAction(co.z1) => X(wasEvent(ep.e2) or wasEvent(ep.e4)) )`	Если было вызвано действие z1, то следующим обработанным событием будет e2 или e4
`G( wasEvent(ep.e12) => X(wasEvent(ep.e2) or wasEvent(ep.e3) or wasEvent(ep.e4)) )`	Если было событие e12, то следующим обработанным событием будет e2, или e3, или e4
`G( wasAction(co.z1) => X( U(!wasAction(co.z1), wasAction(co.z2) or wasEvent(ep.e4)) ) )`	Если было вызвано действие z1, то оно не будет больше вызвано, пока не будет вызвано z2 или обработано событие e4
`G( wasAction(co.z2) => X( U(!wasAction(co.z2), wasAction(co.z1) or wasEvent(ep.e4)) ) )`	Если было вызвано действие z2, то оно не будет больше вызвано, пока не будет вызвано z1 или обработано событие e4
`!F(wasEvent(ep.e4) and X(F(wasEvent(ep.e11) or wasEvent(ep.e12) or wasEvent(ep.e2) || wasEvent(ep.e3))))`	Не верно, что в будущем будет после события e4 когда либо будут обработаны e11, e12, e2 или e3 (лифт не может самостоятельно починиться)

Таблица 2. Темпоральные свойства, составляющие спецификацию системы управления дверьми лифта

Целью экспериментального исследования было сравнение метода построения управляющих конечных автоматов на основе тестов с предлагаемым в настоящей работе методом, использующим верификацию моделей на различных стадиях работы алгоритма генетического программирования (вычисление функции приспособленности, скрещивание и мутация). Для оценки трудоемкости сравниваемых методов было проведено по 1000 экспериментов, и для каждого эксперимента записывалось число вычислений функции приспособленности. Эксперименты показали, что при построении автоматов только на основе тестов, очень редко (всего в девяти случаях из 1000) результатом являлся автомат, который полностью удовлетворяет спецификации. Пример автомата, построенного только на основе тестов, приведен на рис. 2.

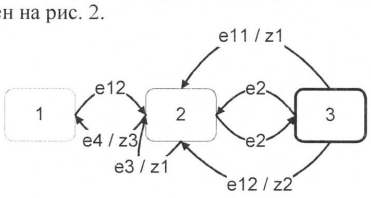


Рис. 2. Автомат управления дверьми лифта, построенный только на основе обучающих примеров

Это автомат обладает тем недостатком, что может отдать команду на закрытие дверей после того, как они сломаются или же начать открывать (закрывать) двери, когда они уже открыты (закрыты). Отметим, что некоторые из построенных в этом эксперименте автоматов обладали и другими недостатками. При использовании предлагаемого метода (с применением верификации моделей) построение автомата (рис. 3) занимало больше времени, но построенный автомат удовлетворял всем требованиям спецификации.

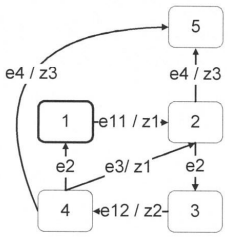


Рис. 3. Автомат управления дверьми лифта, построенный с использованием верификации

При построении конечного автомата управления дверьми лифта только на основе тестов, среднее значение вычислений функции приспособленности оказалось равным $7{,}479 \cdot 10^4$ (минимальное число вычислений – $2{,}184 \cdot 10^4$, максимальное – $2{,}999 \cdot 10^5$, среднеквадратичное отклонение – $2{,}54 \cdot 10^4$).

При использовании верификации моделей совместно с тестами, среднее значение числа вычислений функции приспособленности оказалось равным $7.246 \cdot 10^5$ (минимальное число вычислений – $7{,}054 \cdot 10^4$, максимальное – $5{,}492 \cdot 10^6$, среднеквадратичное отклонение – $7{,}729 \cdot 10^5$).

Таким образом, использование верификации хоть и замедляет процесс построения управляющего конечного автомата примерно в десять раз, но если принять во внимание то, что при построении только на основе тестов процент правильно построенных автоматов меньше 1%, то применение предлагаемого в настоящей работе метода оправдывает себя.

Заключение

Предложен метод машинного обучения для построения управляющих конечных автоматов на основе обучающих примеров. Предложенный метод основан на совместном применении генетического программирования и верификации моделей программ. Применение верификации в процессе работы алгоритма генетического программирования позволяет говорить об автоматизированном построении автоматов с гарантированным поведением.

Исследование проводится в рамках Федеральной целевой программы «Научные и научно-педагогические кадры инновационной России на 2009–2013 годы», а также финансируется по гранту РФФИ № 10-01-00654а.

Литература

1. Поликарпова Н.И., Шалыто А.А. Автоматное программирование. – СПб: Питер, 2009.
2. Angeline P.J., Pollack J. Evolutionary Module Acquisition // Proceedings of the Second Annual Conference on Evolutionary Programming. 1993. [Электронный ресурс]. – Режим доступа: http://www.demo.cs.brandeis.edu/papers/ep93.pdf, свободный. Яз. англ. (дата обращения 17.06.2010).
3. Jefferson D., Collins R., Cooper C., Dyer M., Flowers M., Korf R., Taylor C., Wang A. The Genesys System. 1992. [Электронный ресурс]. – Режим доступа: http://www.cs.ucla.edu/~dyer/Papers/AlifeTracker/Alife91Jefferson.html, свободный. Яз. англ. (дата обращения 17.06.2010).
4. Chambers L. Practical Handbook of Genetic Algorithms. Complex Coding Systems // CRC Press, 1999. – V. III. – P. 659.
5. Царев Ф.Н. Метод построения автоматов управления системами со сложным поведением на основе тестов с помощью генетического программирования // Материалы международной научной конференции «Компьютерные науки и информационные технологии». – Саратов: СГУ, 2009. – С. 216–220.
6. Clarke E., Peled D., Vardi M.Y., Wolper P. Simple On-the-fly Automatic Verification of Linear Temporal Logic // Proc. of the 15th Workshop on Protocol Specification, Testing, and Verification, Warsaw. – 1995. – P. 3–18.
7. Кларк Э., Грамберг О., Пелед Д. Верификация моделей программ: Model Checking. – М.: МЦНМО, 2002.
8. Разработка технологии верификации управляющих программ со сложным поведением, построенных на основе автоматного подхода. Второй этап. СПбГУ ИТМО, 2007 [Электронный ресурс]. – Режим доступа: http://is.ifmo.ru/verification/_2007_02_report-verification.pdf, свободный. Яз. рус. (дата обращения 17.06.2010).
9. Левенштейн В.И. Двоичные коды с исправлением выпадений, вставок и замещений символов // Доклады Академии наук СССР. – 1963. – № 4. – С. 845–848.

10. Егоров К.В., Шалыто А.А. Методика верификации автоматных программ // Информационно-управляющие системы. – 2008. – № 5. – С. 15–21.
11. Lucas S., Reynolds T. Learning Finite State Transducers: Evolution versus Heuristic State Merging // IEEE Transactions on Evolutionary Computation. – 2007, June. – V. 11. – Is. 3. – P. 308–325.

Егоров Кирилл Викторович – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, магистр прикладной математики и информатики, kegorof@gmail.com

Царев Федор Николаевич – Санкт-Петербургский государственный университет информационных технологий, механики и оптики, аспирант, fedor.tsarev@gmail.com

Шалыто Анатолий Абрамович Санкт-Петербургский государственный университет информационных технологий, механики и оптики, доктор технических наук, профессор, зав. кафедрой, shalyto@mail.ifmo.ru

УДК 004.021

АЛГОРИТМ ВОССТАНОВЛЕНИЯ ТРЕХМЕРНОЙ МОДЕЛИ ЛИЦА ПО ФОТОГРАФИИ

А.В. Шлянников

Предложен метод построения трехмерной модели лица на основе одной входной фотографии. Алгоритм основывается на выделении контрольных точек и характерных особенностей на представленном изображении и переносе их на модель. Для получения характерных особенностей фотографии применяется специальное wavelet-преобразование, выделяющие наиболее информативные признаки. Полученная модель может быть использована в дальнейшем в задачах визуализации и распознавания образов.
Ключевые слова: распознавание лица, 3D-моделирование, wavelet-обработка.

Введение

Задача построения трехмерных моделей лица в настоящее время широко востребована в задачах визуализации трехмерных объектов и в системах распознавания и контроля доступа, поэтому ей посвящено большое количество работ. Однако решенной данную проблему считать нельзя.

Известно несколько подходов для решения поставленной задачи. Наиболее точными являются методы, при которых модель лица строится при помощи специального оборудования. В работе [1] предложен алгоритм обработки информации, полученной с лазерного сканера, для построения моделей и идентификации лиц. Данный подход дает наибольшую эффективность в решении проблемы, однако сужает область применения за счет необходимости использования специального оборудования.

В данной работе применяется подход, основанный на обработке только одного монокулярного изображения, что уменьшает идентичность модели и реального лица на фотографии, но позволяет расширить область применения алгоритма.

В работе [2] предложен метод идентификации лиц по характерным признакам, присутствующим на лице. В текущей работе этот метод был использован для выделения характерных признаков лица на фотографии и построения модели по этим признакам.

Задача формально описывается следующим образом: дается некоторое изображение лица, и требуется построить приближенную пространственную фигуру, отражающую лицо на фотографии. На формат входного изображения накладываются следующие ограничения: фотография содержит только одно лицо и должна быть сделана в анфас. Результатом работы алгоритма является набор точек/полигонов в пространстве, представляющий собой модель лица на фотографии, что и является решением задачи.

Описание алгоритма

Алгоритм решения задачи строится по итеративному принципу. В качестве исходной итерации выбирается некоторая усредненная модель головы человека, и производится пошаговое ее улучшение. На каждом шаге процесса итеративной модификации выполняются следующие действия.

1. Генерируется изображение модели, которое затем используется для сравнения с входным изображением.
2. На входной фотографии и на сгенерированном изображении модели в автоматическом режиме выделяются контрольные точки, содержащие данные о точном положении лица на фотографии, его общей форме и о точных координатах конкретных черт лица на фотографии.
3. Используя координаты контрольных точек, производится аппроксимация конкретных регионов на обоих изображениях с помощью шаблонов, находящихся в специальном словаре.

и естественность синтезированной речи, а также делают их доступными для людей с ограниченными физическими возможностями, т. е. с дефектами слуха, зрения. Разрабатываемая бимодальная система синтеза речи может быть эффективно использована для образовательных задач, в развлекательных приложениях и системах виртуальной реальности, в информационно-справочных системах и автоматах самообслуживания, аудиовизуальных инсталляциях и на телевидении.

Данное исследование поддержано Грантом Президента РФ (проект МК-64898.2010.8); Министерством образования и науки РФ в рамках ФЦП "Научные и научно-педагогические кадры инновационной России" (госконтракты П2579 и П2360); фондами РФФИ и БРФФИ в рамках совместного проекта № 08-07-90002/№ Ф08Р-016а; КНВШ Администрации Санкт-Петербурга (проект 26-05/130); Министерством Образования, Молодежи и Спорта Чешской Республики (проект DIMAS-CZ № ME08106).

Список литературы

1. **Železný M., Krňoul Z., Císař P., Matoušek J.** Design, implementation and evaluation of the Czech realistic audio-visual speech synthesis // Signal Processing. 2006. Т. 86. № 12. С. 3657—3673.

2. **Lobanov B., Tsirulnik L.** Development of multi-voice and multi-language TTS synthesizer (languages: Belarussian, Polish, Russian) // Труды 11-й Международной конференции "Речь и Компьютер" SPECOM'2006, Санкт-Петербург, 2006. P. 274—283.

3. **Govokhina O., Bailly G., Breton G.** Learning optimal audiovisual phasing for a HMM-based control model for facial animation // Proc. of ISCA Speech Synthesis Workshop, Bonn, Germany, 2007, CD.

4. **Sekiyama K.** Differences in auditory-visual speech perception between Japanese and America: McGurk effect as a function of incompatibility // Journal of the Acoustical Society of Japan. 1994. Т. 15. С. 143—158.

5. **Karpov A., Tsirulnik L., Zelezny M., Krnoul Z., Ronzhin A., Lobanov B.** Study of Audio-Visual Asynchrony of Russian Speech for Improvement of Talking Head Naturalness // Труды 13-й Международной конференции SPECOM'2009, Санкт-Петербург, 2009. С. 130—135.

6. **Conrey B., Pisoni D.** Audiovisual asynchrony detection for speech and nonspeech signals // Audio-Visual Speech Processing AVSP'2003, St. Jorioz, France, 2003. С. 25—30.

7. **McGurk H., MacDonald J.** Hearing Lips and Seeing Voices // Nature. 1976. Т. 264, № 5588. С. 746—748.

УДК 004.8

С. И. Попов, студент,
Ю. И. Попов, студент,
А. А. Шалыто, д-р техн. наук, проф., зав. каф.,
Санкт-Петербургский государственный
университет информационных технологий,
механики и оптики,
e-mail: yurpopov@rambler.ru

Задача о муравьеде и муравьях

Рассматривается расширение "Задачи об умном муравье-3", названное авторами "Задача о муравьеде и муравьях". В задаче используется двумерный тор произвольного размера с расположенными на нем муравьедом, муравьями (пища для муравьеда) и яблоками (пища для муравьев). Муравьедом и муравьями управляют автоматы Мили.

Цель работы — построить с помощью генетического алгоритма автоматы, управляющие муравьедом и муравьями, которые позволят эффективно решить поставленную задачу. Для построения генетического алгоритма необходимо разработать способы представления автоматов, выбрать функции приспособленности, операторы скрещивания, мутации и отбора.

Авторами разработано инструментальное средство для решения "Задачи о муравьеде и муравьях" при различных значениях используемых параметров.

Ключевые слова: *задача об умном муравье, искусственный интеллект, коэволюция, генетический алгоритм, автомат Мили*

Известна классическая "Задача об умном муравье" [1, 2], в которой с помощью генетического алгоритма должен быть сгенерирован автомат, управляющий одним муравьем с простым полем обзора. Муравей за фиксированное число шагов съедает максимальное число яблок, расположенных определенным образом на двумерном торе. В работах [3, 4] рассмотрена более сложная "Задача об умном муравье-3", в которой муравей имеет большее поле обзора, а число яблок при их случайном расположении на торе не фиксировано.

Однако для практики представляют интерес задачи, в которых особи взаимодействуют. Такие задачи принадлежат классу "коэволюций" [2, 5, 6]. Однако среди задач этого класса неизвестны задачи, решением которых были бы автоматы, сгенерированные генетическими алгоритмами.

Для устранения этого пробела авторами была предложена "Задача о муравьеде и муравьях". Настоящая работа посвящена описанию и решению этой задачи.

1. Постановка задачи

Задано поле — двумерный тор произвольного размера с координатной сеткой, которая делит его поверхность на клетки. На поле расположены муравьед, муравьи (еда для муравьеда) и яблоки (еда для муравьев). Яблоки и муравьи располагаются случайным образом. На одной из клеток находится муравьед (рис. 1).

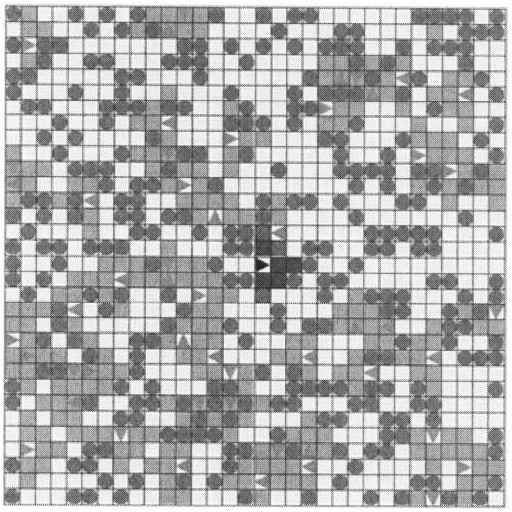


Рис. 1. Тор с муравьедом, муравьями и яблоками. Черный треугольник изображает муравьеда, черные клетки — это те клетки, которые он видит; серые треугольники — муравьи, серые клетки —та часть поверхности, которую видят муравьи; темно-серые кружочки обозначают места расположения яблок

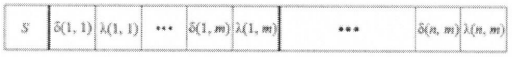


Рис. 2. Представление автомата Мили

Цель каждого муравья — за заданное число шагов съесть как можно больше яблок и "не попасться" муравьеду. Цель муравьеда — помешать им сделать это, съев за заданное число шагов максимальное число муравьев. Муравьи не должны "наступать" на других муравьев и муравьеда, так как они при этом погибают.

Будем предполагать, что муравьедом и муравьями управляют автоматы Мили [1].

Яблоки лежат на поле неподвижно, а муравьед и муравьи на каждом шаге могут совершить одно из трех действий (выходные воздействия соответствующих автоматов):

- повернуть налево;
- повернуть направо;
- пойти на одну клетку вперед, и, если там есть еда (муравей для муравьеда или яблоко для муравья), то съесть ее.

Муравьед и муравьи видят перед собой по восемь клеток: две слева, две справа, две спереди и по одной спереди по диагонали.

Задача, решаемая в настоящей работе: построить генетический алгоритм [2, 5—7], позволяющий сгенерировать автоматы Мили, управляющие муравьедом и муравьями, которые обеспечат достижение целей, указанных выше.

Для построения генетического алгоритма необходимо разработать способы представления автоматов, выбрать функции приспособленности, операторы скрещивания, мутации и отбора.

2. Представление автоматов

Для представления автомата Мили в работе используется массив байтов, в котором первый элемент содержит номер начального состояния автомата S. Затем следуют n секторов (n — число состояний автомата), каждый из которых содержит m пар чисел (m — число входных воздействий), каждая из которых состоит из следующего состояния δ и выходного воздействия λ (рис. 2).

В рассматриваемой задаче автоматы могут принимать большое число входных воздействий, так как каждая клетка поля может содержать муравьеда, муравья, яблоко или не содержать их.

Муравьед не различает яблоки и пустые клетки, и поэтому его автомат может принять $2^8 = 256$ входных воздействий (на клетке есть муравей либо его нет).

Муравей различает клетки с яблоками и пустые клетки. К тому же он может видеть (если они находятся в поле его зрения) других муравьев (для того, чтобы не наткнуться на них) и муравьеда (для того, чтобы вовремя убежать от него). Таким образом, автомат муравья может принять $3^8 + 8 \cdot 3^7 = 6561 + 8 \cdot 2187 = 24057$ входных воздействий.

3. Функции приспособленности

Функция приспособленности автомата, управляющего муравьем, равна сумме числа съеденных последним яблок и дроби, числителем которой является число этих яблок, а знаменателем — номер шага, на котором было съедено последнее яблоко. Функция приспособленности автомата, управляющего муравьедом, вычисляется аналогично, но при этом вместо яблок используются муравьи.

При решении задачи эти функции приспособленности вычисляют следующим образом: по очереди из текущего поколения выбирается один автомат, управляющий муравьедом, и для него случайным образом из текущего поколения автоматов, управляющих муравьями, выбирается заданное в эксперименте число различных автоматов k.

На поле размещают муравьеда и k муравьев, управляемых этими автоматами. При этом для яблок и муравьев генерируются случайные начальные координаты, а для каждого муравья также случайно указывается начальное направление движения. Все участвующие в эксперименте особи (муравьед и муравьи) совершают заданное число шагов.

Выбранные муравьед и муравьи имеют заданное число попыток для случайного размещения

на поле муравьев и яблок. При этом для каждой особи в отдельности значения функции приспособленности автомата на всех полях суммируются, а в конце эксперимента сумма делится на число совершенных попыток. Если число попыток у особей разное, то последние попытки не учитываются у тех особей, для которых было задано меньшее число попыток. В результате для каждой особи получается среднее (за несколько попыток) число съеденных ею единиц еды. Если несколько муравьев хотят пойти на одну и ту же клетку, то приоритет имеет тот муравей, чей автомат имеет наибольшее значение функции приспособленности (учитываются и предыдущие попытки). При этом автоматы, управляющие муравьями, которые не смогли сделать шаг, остаются в предыдущих состояниях.

После вычисления значений функций приспособленности всех автоматов муравьедов, если остаются автоматы муравьев, которые не участвовали в эксперименте, то для каждого из них выбирают автомат муравьеда и необходимое число автоматов муравьев и вычисляют значения их функций приспособленности. При этом функции приспособленности автоматов, вычисленные ранее, имеют после эксперимента предыдущие значения.

В заключение раздела отметим, что в ходе коэволюции в общем случае автоматы муравьев, даже при одинаковом числе состояний, отличаются друг от друга.

4. Оператор скрещивания

Авторами был выбран один и тот же оператор скрещивания для автоматов муравьедов и муравьев. Он обеспечивает компромисс между временем работы и скоростью улучшения автоматов. Оператор скрещивания сначала копирует автомат первого предка в одного потомка, а автомат второго предка — в другого потомка. Затем он случайно устанавливает потомкам различные начальные состояния от предков. После этого 30 раз для каждого из потомков выбирается фрагмент другого предка (чьей копией потомок до этого не был) длиной в 1 % от общей длины (напомним, что автоматы хранятся в виде байтовой строки), и этот фрагмент устанавливается на те же позиции потомка, на которых он был у предка.

5. Оператор мутации

В операторе мутации для муравьеда и муравья с заданной вероятностью изменяется начальное состояние автомата. Затем выполняется проход по всем состояниям (в каждом из них, в свою очередь, выполняется проход по всем входным воздействиям) и с той же вероятностью на случайные изменяются номера состояний для переходов и выходные воздействия.

6. Оператор отбора автоматов в следующее поколение

Каждое поколение в генетическом алгоритме делится на элиту (автоматы с наибольшими значениями функций приспособленности) и остальные автоматы. Сначала в следующее поколение переходят автоматы из элиты. Далее для отбора автоматов используется метод рулетки [2, 5, 7]. К автоматам, которые были отобраны методом рулетки, применяется оператор скрещивания, а к их потомкам — оператор мутации. После этого они добавляются в следующее поколение к элите.

7. Инструментальное средство

Для решения поставленной задачи разработано инструментальное средство, поддерживающее описанные особенности рассмотренного генетического алгоритма. Это средство позволяет проследить процесс "выращивания" автоматов, а также визуализировать поведение особей, управляемых этими автоматами, на случайном поле с муравьями и яблоками.

Инструментальное средство позволяет задать значения параметров эксперимента (рис. 3): ширины поля, высоты поля, заполненности поля яблоками, заполненности поля муравьями, на-

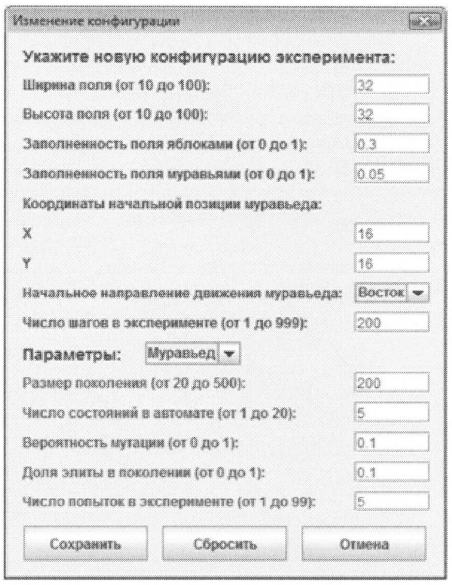


Рис. 3. Выбор параметров эксперимента и особей

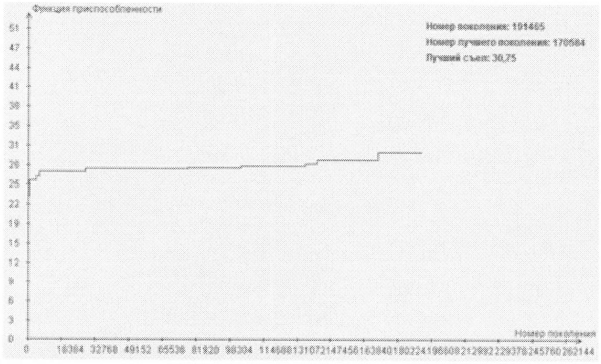


Рис. 4. График зависимости числа съеденных муравьев от номера поколения муравьеда

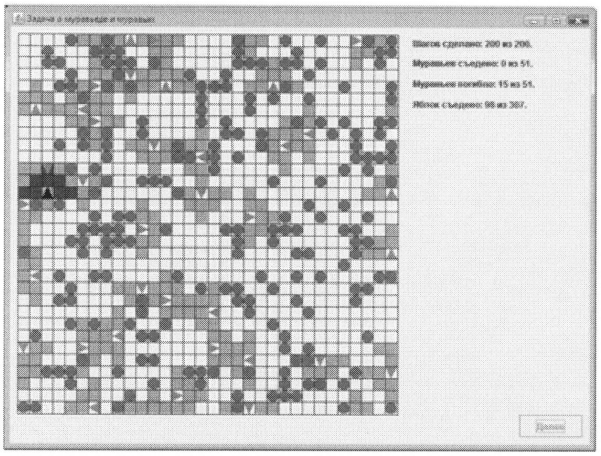


Рис. 5. Вид поля в начале генерации поколений после 200 шагов. Муравьед никого не съел, муравьи съели меньше половины яблок

чальных координат муравьеда, начального направления движения муравьеда, число шагов в эксперименте.

Для автоматов муравьеда и муравьев (все автоматы муравьев имеют одинаковые параметры) раздельно можно задать следующие параметры: размер поколения, число состояний в автомате, вероятность мутации, долю элиты в поколении, число попыток в эксперименте (число случайных полей, на которых особь пытается добиться своей цели и улучшить значение функции приспособленности своего автомата).

Инструментальное средство занимает около 200 Мбайт оперативной памяти. Для вычисления следующего поколения требуется примерно 2,5 с при использовании процессора *Intel Pentium* 4 с частотой 3,2 ГГц.

8. Результаты экспериментов

Для быстрого обучения автоматов, управляющих муравьедом, необходимо иметь большое число муравьев на поле. Аналогично для автоматов муравьев: если на поле больше яблок, то муравьи учатся есть быстрее. Авторы решили рассматриваемую "Задачу о муравьеде и муравьях" при следующих параметрах:

Параметр	Значение
Ширина поля	32
Высота поля	32
Заполненность поля яблоками	0,3
Заполненность поля муравьями	0,05
Координата X начальной позиции муравьеда	16
Координата Y начальной позиции муравьеда	16
Начальное направление движения муравьеда	Восток
Число шагов в эксперименте	200

Для автоматов муравьедов и муравьев были выбраны одинаковые значения параметров, указанные ниже:

Параметр	Значение
Размер поколения	200
Число состояний в автомате	5
Вероятность мутации	0,1
Доля элиты в поколении	0,1
Число попыток в эксперименте	5

Из графика на рис. 4 видно, что особь постепенно учится съедать все большее число муравьев (исходно на поле был размещен 51 муравей). При этом, если вначале муравьед передвигается случайным образом и чаще всего вращается на месте (рис. 5), то, как показали эксперименты, уже к 50-му поколению прослеживается его стремление догнать жертву, хотя иногда в нужный момент он сворачивает не туда.

Чем больше номер поколения, тем меньше подобных ошибок. Ближе к 2000-му поколению муравьед уже уверенно преследует муравьев, несмотря на то, что они пытаются от него убежать (муравьи отличают муравьеда от себе подобных и яблок и постепенно учатся к нему не приближаться) (рис. 6).

При дальнейшем увеличении числа поколений муравьед поедает больше муравьев. Так, в 170584-м поколении значение функции приспособленности автомата муравьеда равно 30,75 (см. рис. 4).

На втором графике (рис. 7) виден процесс роста числа съеденных яблок у муравьев от поколения к поколению (сравните рис. 5 и 6). На результат (значение функции приспособленности равно 15,87 в 127473-м поколении) повлияло то, что му-

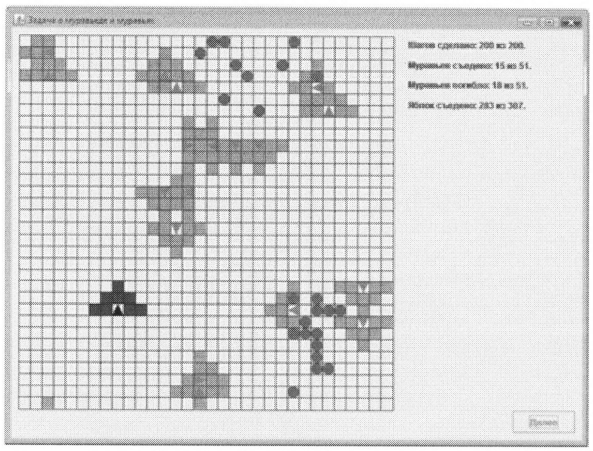


Рис. 6. Вид поля в процессе генерации поколений после 200 шагов. Муравьед съел 15 муравьев, муравьи съели большинство яблок

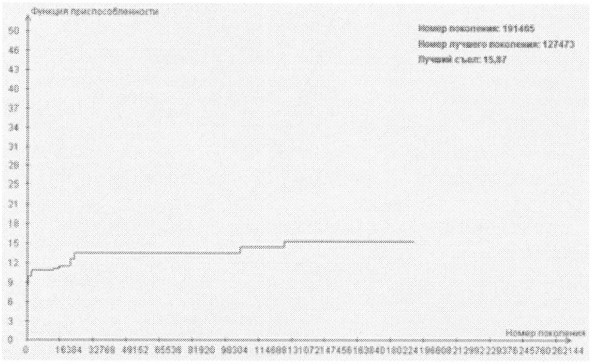


Рис. 7. График зависимости числа съеденных яблок от номера поколения муравьев

равьи фактически соревнуются между собой за яблоки, число которых на поле исходно равнялось 307.

Заключение

Авторы сформулировали новую коэволюционную задачу и для ее решения разработали инструментальное средство (все графики и рисунки к статье получены с помощью этого средства).

Результаты, приведенные выше, свидетельствуют о том, что генетические алгоритмы "справляются" с этой задачей не хуже, чем со значительно более простой "Задачей об умном муравье-3". При этом отметим, что в рассматриваемой задаче генерируется большее число поколений по сравнению с "Задачей об умном муравье-3" для того, чтобы муравьед съел более половины муравьев. Это связано с тем, что если в известной задаче расположение еды муравья фиксировано, то в рассматриваемой задаче еда от муравьеда может "убегать".

Список литературы

1. **Поликарпова Н. И., Шалыто А. А.** Автоматное программирование. СПб.: Питер, 2009. URL: http://is.ifmo.ru/books/ _book.pdf

2. **Koza J.** Genetic programming. On the Programming of Computers by Means of Natural Selection. MA: The MIT Press, 1998.

3. **Бедный Ю. Д., Шалыто А. А.** Применение генетических алгоритмов для построения автоматов в задаче "Умный муравей". СПбГУ: ИТМО, 2007. URL: http://is.ifino.ru/works/ant.

4. **Давыдов А. А., Соколов Д. О., Царев Ф. Н., Шалыто А. А.** Виртуальная лаборатория обучения генетическому программированию для генерации управляющих конечных автоматов // Сборник докладов III Международной научно-практической конференции "Современные информационные технологии и ИТ-образование". ВМК МГУ. М.: МАКС Пресс. 2008. С. 179—183. URL: http://is.ifmo.ru/works/_2_93_davidov_sokolov.pdf

5. **Mitchell M.** An Introduction to Genetic Algorithms. MA: The MIT Press, 1999.

6. **Джонс М.** Программирование искусственного интеллекта в приложениях. М.: ДМК Пресс, 2006.

7. **Гладков Л. А., Курейчик В. В., Курейчик В. М.** Генетические алгоритмы. М.: Физматлит, 2006.

УДК 681.5

ВЫДЕЛЕНИЕ СТРУКТУРНЫХ ПРИЗНАКОВ ИЗОБРАЖЕНИЙ СИМВОЛОВ НА ОСНОВЕ КЛЕТОЧНЫХ АВТОМАТОВ С МЕТКАМИ

***Д. И. Суясов*[1],**
аспирант
Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Рассматривается выделение структурных признаков изображений символов, что является частью процесса распознавания текста. Использование клеточных автоматов позволяет упростить вычисления, исключив операции с плавающей точкой, и обеспечивает возможность параллельной обработки точек изображения. Вводятся понятия клеточного автомата с метками и управления набором клеточных автоматов. Предлагаются алгоритмы по выделению признаков изображений символов на основе клеточных автоматов с метками и приводятся результаты экспериментов над ними.

Ключевые слова *— клеточный автомат, распознавание текста, структурные признаки.*

Введение

С середины XX в. ведутся работы по распознаванию текстов. Распознаванием текста называют перевод последовательности графических образов символов в последовательность символов, воспринимаемую ЭВМ в качестве текста. Решение этой задачи позволит упростить работу человека с системами документооборота, анализ данных, автоматическую обработку текстов и анкет при переводе в электронный вид печатных архивов и т. д. В настоящее время эта задача продолжает быть актуальной, так как погрешность существующих систем и алгоритмов при распознавании текстов хоть и достаточно низка (в некоторых системах достигает 0,2–0,5 %), но не настолько мала, чтобы ею можно было пренебречь.

В большинстве работ, связанных с областью систем распознавания, предлагается разделение процесса распознавания на составляющие. В них рассматриваются отдельные подпроцессы по фильтрации изображений [1], сегментированию [2, 3], преобразованию изображений [4, 5] и выделению из них признаков [6] с последующей классификацией.

Данная работа является частью исследований по определению перспективных подходов к построению систем распознавания текста, основанных на простых вычислениях и возможности параллельной обработки изображений символов текста. В исследовании, аналогично большинству работ, предлагается рассматривать системы распознавания текста как совокупность подпроцессов, но при этом решаются задачи по их упрощению и реализации на основе автоматного подхода [7, 8].

В настоящей работе рассматривается процесс выделения структурных признаков изображений символов, как один из наиболее важных этапов распознавания текста. Предлагается простой и эффективный подход к решению данной задачи с помощью клеточных автоматов.

Клеточные автоматы были предложены в работе фон Неймана [7] и получили развитие в 70–80-х гг. прошлого века. Они являются примером распределенных систем, основанных на простых правилах, которые позволяют реализовывать сложное поведение [8–10]. Сегодня существует большое число статей по клеточным автоматам, связанных с системами распознавания текста [2, 3, 11–13]. В основном в них делается упор на генетический подход к построению эффектив-

[1] Научный руководитель — доктор технических наук, профессор, заведующий кафедрой технологий программирования Санкт-Петербургского государственного университета информационных технологий, механики и оптики *А. А. Шалыто*.

ных клеточных автоматов. К сожалению, данный подход позволяет лишь приблизиться к оптимальному решению для выделенных наборов экспериментальных изображений с текстом. Универсального алгоритма при таком подходе не найдено.

В настоящей статье рассматривается клеточный автомат и вводится новый его тип — клеточный автомат с метками, а также прорабатывается вопрос построения набора клеточных автоматов. Затем определяются признаки, присущие символам текста, и излагаются принципы их выделения, описываются алгоритмы выделения указанных признаков с помощью клеточных автоматов, обсуждается вопрос производительности и эффективности работы этих алгоритмов.

Клеточный автомат

Клеточный автомат — это распределенная система, состоящая обычно из однородных простых элементов, локально связанных на периодической решетке (поле), глобальное состояние которых изменяется с течением времени на основе локальных правил переходов элементов поля [8].

Формально клеточный автомат определяется как набор {G, Z, N, f}, где:

G — метрика поля, на котором действует клеточный автомат;

Z — множество состояний каждой клетки;

N — окрестность клетки, которая влияет на состояние данной клетки;

f — правила клеточного автомата, $Z \times Z^{|N|} \to Z$.

Существует несколько классификаций клеточных автоматов: по способу применения локальных правил на элементы поля (синхронные и асинхронные), по возможности применять различные правила для разных элементов поля (однородные и неоднородные), а также по наличию вероятностных правил переходов (детерминированные и вероятностные) и по стационарности клеток поля (подвижные и неподвижные). В данной работе рассматриваются синхронные однородные детерминированные неподвижные клеточные автоматы.

Дальнейшая классификация клеточных автоматов состоит в следующем [8]. Для циклического клеточного автомата определяется несколько наборов правил переходов, которые поочередно применяются к полю. Другой тип клеточных автоматов — мобильные клеточные автоматы, в которых определены одна или несколько активных клеток, участвующие в локальных правилах переходов. Часто изменения в мобильном клеточном автомате происходят только с активными клетками поля.

В качестве развития идеи мобильного клеточного автомата введем другой тип — клеточный автомат с метками. В этом автомате состояние каждого элемента (клетки) поля определяется двумя составляющими. Первая из них представляет собой классическое состояние, введенное в рамках основного определения клеточного автомата. Вторая часть — это набор меток, которые по отдельности или в некоторой комбинации могут принадлежать клетке или не принадлежать ей. Метки в данном случае могут играть роль маркеров для клеток.

Формально клеточный автомат с метками можно определить как набор {G, M, Z, N, f}, где:

G — конечное дискретное метрическое множество, гарантирующее конечность расстояний между клетками;

M — конечное множество меток, определенное для каждой клетки;

Z — конечный набор состояний клеток;

N — конечное множество, определяющее окрестность клетки ($|N|$ — число соседних клеток, которые влияют на состояние данной клетки);

f — правила клеточного автомата, соответствующие математической функции переходов $Z \times C \times (Z \times C)^{|N|} \to Z \times C$, где $C \subset M$.

Отметим аналогию между клеточным автоматом с метками и понятием памяти в тьюринговых машинах [14]. Головка тьюринговой машины может хранить некоторое конечное число данных, которые используются в правилах автомата машины для изменения ленты или передвижения головки.

Клеточный автомат является примером системы, которая решает сложные задачи на основе простых локальных правил. К сожалению, не все задачи поддаются решению лишь одним клеточным автоматом. Для решения может потребоваться набор (последовательность) клеточных автоматов, каждый из которых будет решать специализированную подзадачу.

Набор клеточных автоматов — это совокупность клеточных автоматов, связанных единой метрикой поля и одинаковым набором состояний клеток, но имеющих разный набор правил переходов. В наборе должны быть введены дополнительные правила:

- правила очередности запуска клеточных автоматов (при этом каждый следующий клеточный автомат начинает работу на поле, которое было сформировано предыдущим клеточным автоматом);
- правила завершения работы каждого автомата, которые могут иметь вид: работа клеточного автомата может ограничиваться числом итераций, клеточный автомат может функционировать до выполнения определенного условия или

завершаться, когда применение правил автомата не изменяет состояние поля.

Набор клеточных автоматов может содержать не только сами клеточные автоматы, но и управляющие элементы — операторы переходов и циклов, условные операторы и т. д. Управление запуском клеточных автоматов и управляющих элементов может осуществляться на основе конечных автоматов [15].

Выделение признаков символов

Существует несколько подходов к процессу идентификации (распознаванию) символов. Можно работать с изображением символа как единым объектом или выделять некоторые базисные характеристики, которые легче поддаются исследованию. Структурный подход [16, 17] используется при выделении характеристик и признаков из изображений. В данной работе в рамках структурного подхода предлагается выделять структурные признаки изображений символов с помощью клеточных автоматов. При этом предлагается выделять следующие признаки: расположение концов отрезков, из которых состоит изображение символа; положение замкнутых контуров; места пересечений отрезков. В общем случае для распознавания образов или текстов с различных языков такого числа признаков может быть недостаточно [16], однако настоящая работа не ставит перед собой задачу показать все возможные признаки изображений, основной упор здесь делается на простоту их извлечения. Для задач распознавания русскоязычных символов данных признаков в большинстве случаев оказывается достаточно (существует зависимость от сложности стилей, применяемых к тексту).

Для того чтобы выделить подобные признаки, введем понятие распространения волны. *Волна* определяется как итерационный процесс распространения меток от одних клеток (точек) к соседним. Предлагается пускать волну по точкам изображения символа.

Волна состоит из *фронта* (точек, помеченных специальной меткой фронта волны) и *шлейфа* (точек, помеченных специальной меткой шлейфа волны, которая присваивается точке с меткой фронта волны на следующей итерации). Метка обработанных точек выставляется для тех точек изображения, в которых в предыдущие итерации выставлялись метки фронта и шлейфа волны. Метки фронта волны, шлейфа волны и обработанных точек не могут присутствовать на одной клетке одновременно.

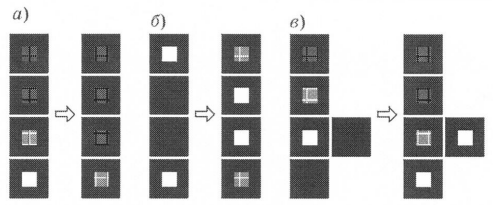
В процессе распространения волна, начинаясь с некоторой позиции, обходит все изображение символа и угасает на его концах или при встрече с другой волной. Реализация идеи распространения волны сводится к циклическому процессу вида, представленного на рис. 1:

- в начальный момент выделяется точка изображения символа и ей присваивается метка фронта волны (существует некоторая корреляция между расположением результирующих признаков символов и способом выделения начальной точки распространения волны, поэтому предлагается использовать единую стратегию для всех символов с целью уменьшить данный эффект, например выделять крайнюю верхнюю точку или верхнюю среднюю точку, если их несколько);
- первый шаг цикла включает замену всех меток шлейфа волны для точек изображения на метки обработанных точек;
- все точки с метками фронта волны изменяют метку на метку шлейфа волны;
- всем точкам изображения символа без меток, соседствующим (по фон Нейману [8]) с любыми точками с метками, присваиваются метки фронта волны;
- цикл повторяется.

Для выделения структурных признаков символов фиксируются моменты встречи волн, угасания волн на концах изображения символа и моменты их разделения на части:

- момент угасания волны определяется, когда у группы смежных точек с метками шлейфа волны отсутствует по соседству точка с меткой фронта волны (рис. 2, *а*);
- момент встречи двух волн определяется, когда точки с метками шлейфа волны разделены точками с метками фронта волны (рис. 2, *б*);

■ ***Рис. 1.*** *Этапы итерационного процесса распространения волны по точкам изображения*



■ ***Рис. 2.*** *Процесс распространения волн в ситуациях: а — угасания волны на конце символа; б — встречи двух волн; в — разделения волны на части*

• момент разделения волны на части определяется, когда точки с метками фронта волны разделены точками с метками шлейфа волны (рис. 2, *в*).

На основе информации о расположении точек, в которых волна угасает, разделяется на части или встречается с другими волнами, можно судить о расположении структурных признаков символов.

Алгоритмы выделения признаков символов

Для реализации процесса выделения признаков символов предлагается использовать набор клеточных автоматов с метками. При этом в качестве базовой используется идея распространения волны. Выделенные в ходе распространения волны позиции (позиции угасания волны, встречи волн и разделение волны) интерпретируются алгоритмом как признаки изображения символа:

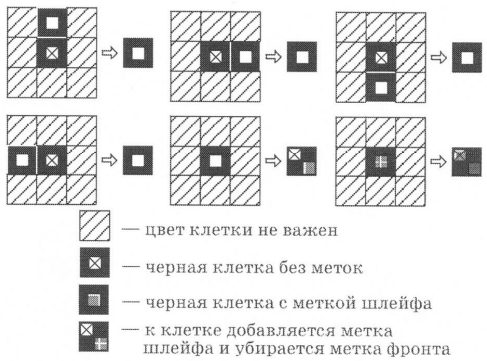
• конец символа определяется местом угасания волны;

• пересечение отрезков символа — это место раздвоения волны на части;

• петли в символе определяются позициями встречи волн.

Таким образом, алгоритм по выделению признаков символов будет следующим:

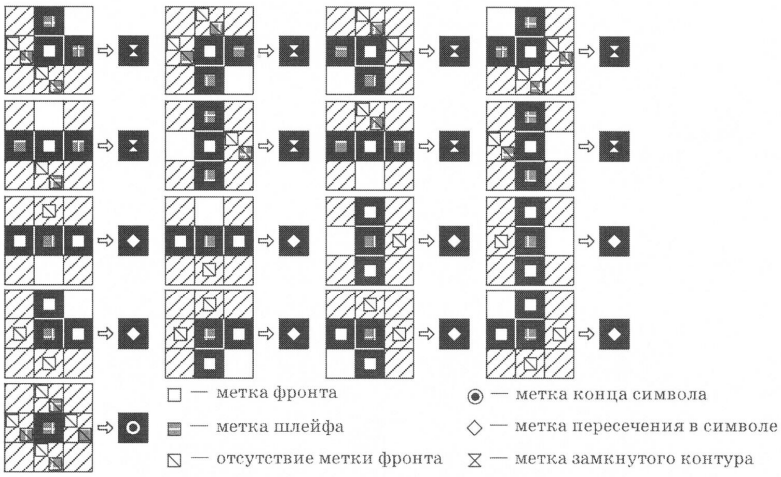
<Пометить точку изображения символа меткой фронта волны>
<Итерироваться до момента полного угасания волны>
Начало итерации
  <Заменить метку шлейфа волны на метку обработанных точек>
  <Заменить метку фронта волны на метку шлейфа волны>
  <Выставить для необработанных точек, граничащих с помеченными точками, метку фронта волны>
  <Проверить наличие точек с меткой шлейфа волны, не граничащих с точками фронта волны, в случае нахождения выделить конец символа>
  <Проверить наличие точек шлейфа волны, разбитых точками фронта волны, в случае нахождения выделить место встречи волн>
  <Проверить наличие точек фронта волны, разбитых точками шлейфа волны, в случае нахождения выделить место разделения волны на части>
Конец итерации



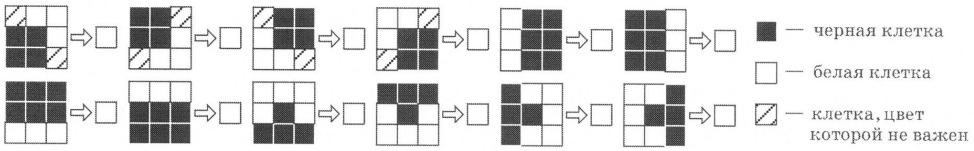
■ *Рис. 3. Правила клеточного автомата с метками, осуществляющего распространение волны по точкам изображения символа*

Правила клеточного автомата с метками, осуществляющего распространение волны по точкам изображения символа, показаны на рис. 3.

Правила клеточного автомата с метками, выделяющего места угасания, встречи и разделения волн, показаны на рис. 4.



■ *Рис. 4. Правила клеточного автомата с метками, выделяющего места угасания, встречи и разделения волн*



■ *Рис. 5. Правила циклического клеточного автомата по утончению символов*

Отметим одну особенность распространения волн: две волны, распространяющиеся из одной точки дискретной области, могут прийти в другую точку изображения лишь одновременно. Доказательством этого может служить тот факт, что область, которую огибают волны в процессе распространения, имеет четное число соседних (в смысле Мура [8]) клеток. Поэтому волны, проходя по разным путям и имеющие одну скорость распространения, могут пройти только одинаковое расстояние до точки. Таким образом, можно считать корректными правила клеточного автомата, определяющие встречу волн.

Основной недостаток предложенного алгоритма — это возможность появления лишних подельных признаков для изображения символов. Данное событие может случиться из-за особенностей начертания символов, что обеспечит, например, выделение несуществующих концов символа.

Это можно решить следующим образом. Перед началом распространения волны необходимо утончить изображение, уменьшить толщину отрезков, из которых состоит символ, до толщины в одну точку (клетку). Данную задачу можно выполнить с помощью алгоритма, предложенного Чжаном (T. Y. Zhang) и Суэнем (C. Y. Suen) [5], или используя алгоритм постепенного утончения отрезков символов на основе циклического клеточного автомата [18]. Правила клеточного автомата для данной операции представлены на рис. 5.

Использование алгоритма выделения признаков символов после предварительного утончения изображения позволяет получить более точные позиции признаков символа на изображении. Утонченные линии символа получены удалением лишних точек со всех сторон. Полученное скелетное представление является набором центральных линий исходного изображения символа, а следовательно, выделенные признаки позиционируются более точно.

Структурная схема связи клеточных автоматов, реализующих работу алгоритма по выделению признаков символов, показана на рис. 6.

Исследование характеристик представленных алгоритмов проводилось на нескольких наборах изображений русских символов как печатных, так и рукописных. В целом, алгоритмы показывают хороший результат и позволяют выделять характерные признаки изображений символов точно. При этом возможны некоторые несоответствия в выделенных признаках для различных шрифтов, например, между шрифтами с засечками и без них. Более точные результаты показывает алгоритм, включающий предварительное утончение изображения символа.

Пример выделенных признаков представлен на рис. 7.

Основные недостатки алгоритмов, как собственно всех структурных методов, — необходимость четкого определения границ изображения символа, и как следствие — подверженность влиянию шумов.

Достоинством структурных методов являются устойчивость к сдвигу, масштабированию, повороту символа на небольшой угол и к возможным дисторсиям и различным стилевым вариациям шрифтов.

Кроме того, реализация на основе клеточных автоматов позволяет распараллелить механизм

■ *Рис. 6. Структурная схема связи клеточных автоматов, реализующих алгоритм выделения структурных признаков символов*

■ *Рис. 7. Выделение признаков с помощью клеточных автоматов с метками на примере буквы «А» в различных стилях без предварительного утончения изображения (а) и с утончением изображения (б)*

выделения признаков, руководствуясь простыми правилами без вычислений с плавающей точкой. Производительность системы во многом зависит от размеров изображения, которое поступает на вход набора клеточных автоматов, и от размеров и толщины линий изображенных символов. Введение алгоритмов объединения точек изображения и кэширования может помочь уменьшить влияние размеров на производительность. В таблице представлены показатели качества и производительности алгоритма для разных входных изображений символов.

Результат выделения признаков может быть использован в системе распознавания текста для дальнейшей классификации изображений символов. В процессе исследования качества и достаточности выделения структурных признаков символов из изображений на основе клеточных автоматов была создана тестовая среда, которая классифицирует выделяемые признаки с помощью шаблонного метода.

Шаблоны признаков были выделены из изображений алфавита русского языка за исключением букв «е», «й» и «ы», так как они состоят из нескольких отдельно записанных элементов, что должно анализироваться другой системой. Данные шаблоны сравнивались с признаками тестовых изображений текстов. Сравнение распознаваемых изображений символов с шаблонными производилось на основе анализа смещений выделенных признаков между этими изображениями. Пример этапов распознавания показан на рис. 8.

Анализ полученных данных позволил сделать вывод, что выделенные признаки в целом дают возможность корректно классифицировать изображения символов. Основные ошибки распознавания связаны с наличием минимальных отличий между схожими символами. Выделенные признаки данных символов показаны на рис. 9.

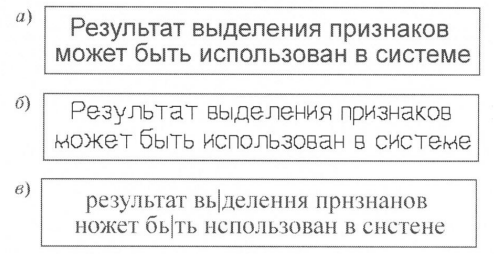
Решение задачи по различению изображений схожих символов решается с помощью учета взаимного расположения выделенных признаков. Для символов, представленных на рис. 9, данное условие позволит решить проблему корректной идентификации. Дополнительным способом правильной идентификации символов может служить морфологический разбор распознанных текстов.

В целом анализ выделенных признаков для разных групп символов и разных шрифтов показал:

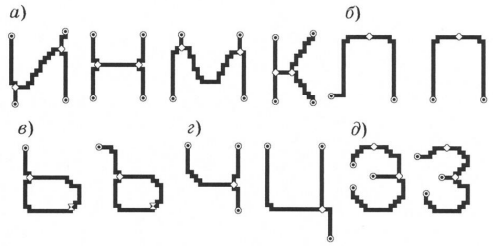
• шрифты с засечками и без засечек формируют разные наборы признаков;

■ *Показатели качества и производительности алгоритма выделения признаков*

Описание изображения	Время работы алгоритма, с	Ошибки при выделении признаков
Рукописный символ «а», 50 × 50	0,1	Без ошибок
Русский алфавит, 450 × 50, шрифт Arial	1,6	Две ошибки, ошибочно выделены признаки для букв «б» и «ж»
Русский алфавит, 600 × 60, шрифт Arial	2,8	Одна ошибка, ошибочно выделен признак для буквы «ж»
Русский алфавит, 600 × 60, шрифт Times New Roman	3,4	Две ошибки, ошибочно выделены признаки для букв «х» и «ъ»
Текст, 600 × 70, шрифт Times New Roman, 57 символов	2,5	Некорректно выделены признаки в букве «м»
Текст, 900 × 40, шрифт Arial полужирный, 57 символов	2,5	Две ошибки, ошибочно выделены признаки для букв «е» и «к»
Текст, 900 × 40, шрифт Arial курсив, 57 символов	2,4	Две ошибки, ошибочно выделены признаки для букв «е» и «к»



■ ***Рис. 8.*** *Начальный (а), промежуточный (б) и конечный (в) вид данных процесса распознавания текста, построенного на предварительном выделении структурных признаков изображений символов*



■ ***Рис. 9.*** *Выделенные признаки схожих символов: «и», «н», «м» и «к» (а); «л» и «п» (б); «ь» и «ъ» (в); «ч» и «ц» (г) и «э» и «з» (д)*

• использование полужирных шрифтов и шрифтов с наклонами в меньшей степени влияет на результат выделения признаков;

• выделенные признаки не могут позволить однозначно идентифицировать символы, поэтому в системах распознавания текста требуются различные инструменты по анализу изображений;

• алгоритм выделения признаков эффективнее работает на утонченных изображениях символов.

Заключение

В данной работе предложен метод выделения признаков символов, основанный на идее распространения волны и реализованный на основе теории клеточных автоматов. Представленные алгоритмы показывают хорошие результаты как по характеристикам качества, так и производительности. Результаты работы могут быть использованы в системах распознавания текста.

Литература

1. **Ramos V., Muge F.** On Image Filtering, Noise and Morphological Size Intensity Diagrams // 11th Portuguese Conf. on Pattern Recognition. Portugal, 2000. P. 483–491.
2. **Rastegar R., Rahmati M., Meybodi M. R.** A Clustering Algorithm using Cellular Learning Automata based Evolutionary Algorithm // Adaptive and Natural Computing Algorithms: Proc. of the Intern. Conf. in Coimbra. Springer Vienna, 2005. P. 144–150.
3. **Ramos V., Muge F.** Image Colour Segmentation by Genetic Algorithms // 11th Portuguese Conf. on Pattern Recognition. Portugal, 2000. P. 125–129.
4. **Фролов А. Б., Четрафилов И. Д.** О некоторых подходах к распознаванию оптических образов текстов // Интеллектуальные системы. 1997. Т. 2. Вып. 1–4. С. 189–200.
5. **Zhang T. Y., Suen C. Y.** A Fast Parallel Algorithm for Thinning Digital Patterns. Image Processing and Computer Vision // Communications of the ACM. 1984. Vol. 27. N. 3. P. 236–239.
6. **Вершок Д. А.** Алгоритмические средства обработки и анализа изображений на основе преобразования Хафа: Автореф. дис. ... канд. техн. наук / БГУ информатики и радиоэлектроники. — Минск, 2002. — 22 с.
7. **Нейман Дж. фон.** Теория самовоспроизводящихся автоматов. — М.: Мир, 1971. — 382 с.
8. **Wolfram S.** A New Kind of Science. — Wolfram Media, 2002. — 1192 p.
9. **Наумов Л.** Решение задач с помощью клеточных автоматов посредством программного обеспечения CAME&L. Ч. I // Информационно-управляющие системы. 2005. № 5. С. 22–30.
10. **Наумов Л.** Решение задач с помощью клеточных автоматов посредством программного обеспечения CAME&L. Ч. II // Информационно-управляющие системы. 2005. № 6. С. 30–38.
11. **Ganguly N.** et al. Evolving Cellular Automata Based Associative Memory for Pattern Recognition // Lecture Notes In Computer Science. London: Springer-Verlag, 2000. Vol. 2228. P. 115–124.
12. **Oliveira C. C., Oliveira P. P.** An Approach to Searching for Two-Dimensional Cellular Automata for Recognition of Handwritten Digits // Lecture Notes In Artificial Intelligence. Berlin; Heidelberg: Springer-Verlag, 2008. Vol. 5317. P. 462–471.
13. **Pradipta M., Sikdar B. K., Chaudhuri P. P.** Cellular Automata Evolution for Pattern Classification // Lecture Notes in Computer Science. Berlin; Heidelberg: Springer-Verlag, 2004. Vol. 3305. P. 660–669.
14. **Хопкрофт Д., Мотвани Р., Ульман Д.** Введение в теорию автоматов, языков и вычислений. — М.: Вильямс, 2002. — 528 с.
15. **Шалыто А. А.** Switch-технология. Алгоритмизация и программирование задач логического управления. — СПб.: Наука, 1998. — 628 с.
16. **Фу К.** Структурные методы в распознавании образов. — М.: Мир, 1977. — 320 с.
17. **Абламейко С. В., Берегов Б. С., Бокуть Л. В.** Исследование структурного строения изображений на основе принципа симметрии // Цифровая обработка изображений: Сб. науч. тр. / Ин-т техн. кибернетики НАН Беларуси. Минск, 1997. Вып. 1. С. 5–14.
18. **Суясов Д. И.** Утончение изображений символов на основе клеточных автоматов // VI Всерос. межвуз. конф. молодых ученых: Сб. тр. / СПбГУ ИТМО, 2009. http://fppo.ifmo.ru/kmu/kmu6/oleg-25.html (дата обращения: 15.12.2009).

ПРИЛОЖЕНИЕ 2. КОПИИ СВИДЕТЕЛЬСТВ О РЕГИСТРАЦИИ ПРОГРАММЫ ДЛЯ ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2010615014

Программное средство для генерации автоматов представленных линейными бинарными графами на основе генетического программирования

Правообладатель(ли): ***Государственное образовательное учреждение высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» (RU)***

Автор(ы): ***Данилов Владимир Рюрикович,***
Шалыто Анатолий Абрамович (RU)

Заявка № **2010613402**
Дата поступления **15 июня 2010 г.**
Зарегистрировано в Реестре программ для ЭВМ
3 августа 2010 г.

Руководитель Федеральной службы по интеллектуальной собственности, патентам и товарным знакам

Б.П. Симонов

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2010614264

Программный комплекс для исследования автоматного управления роботами

Правообладатель(ли): ***Государственное образовательное учреждение высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» (RU)***

Автор(ы): ***Алексеев Сергей Андреевич, Калиниченко Александр Игоревич, Клебан Виталий Олегович, Шалыто Анатолий Абрамович (RU)***

Заявка № **2010612605**
Дата поступления **12 мая 2010 г.**
Зарегистрировано в Реестре программ для ЭВМ
30 июня 2010 г.

Руководитель Федеральной службы по интеллектуальной собственности, патентам и товарным знакам

Б.П. Симонов

ПРИЛОЖЕНИЕ 3. КОПИЯ ЭКСПЕРТНОГО ЗАКЛЮЧЕНИЯ О ВОЗМОЖНОСТИ ОПУБЛИКОВАНИЯ

УТВЕРЖДАЮ
Начальник НИЧ
_____ Л. М. Студеникин
«13» 09 2010 г.

ЭКСПЕРТНОЕ ЗАКЛЮЧЕНИЕ О ВОЗМОЖНОСТИ ОПУБЛИКОВАНИЯ

Экспертная комиссия Санкт-Петербургского государственного университета информационных технологий, механики и оптики Министерства образования и науки РФ рассмотрела на заседании (протокол № 1 от «13» 09 2010 г.)

научно-технический отчет о выполнении 2 этапа по государственному контракту №П2236 от 11 ноября 2009 года по проекту «Применение методов искусственного интеллекта в разработке управляющих программных систем»
(вид, название материала, Ф.И.О. автора (ов))

подтверждает, что в материале не содержатся сведения, предусмотренные разделом 3 Положения 88.
(содержатся ли сведения, предусмотренные разделом 3 Положения 88)

На публикацию материала не следует получить разрешение Министерства образования и науки РФ.

Заключение: материал разрешен к публикации в открытой печати РФ

Председатель комиссии _____ / Зубов Д.А. /

Представитель ОИСиНТИ _____ / Горкина Н.М. /

Секретарь комиссии _____ / Ловцов В.В. /