Министерство образования и науки Российской Федерации

УДК 004.4’242
ГРНТИ 20.01.01, 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 № П1188 от 27 августа 2009 г. и Дополнению от 22 октября 2009 г. № 1/П1188, Дополнению от 02 апреля 2010 г. № 2/П1188

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 № 1.3.1 Проведение научных исследований молодыми учеными - кандидатами наук.
Проект: Разработка методов совместного применения генети-ческого и автоматного программирования для по-строения систем управления беспилотными летатель-ными аппаратами
Руководитель проекта: **_____/Гуров Вадим Сергеевич** (подпись)

Санкт-Петербург
2010 г.

СПИСОК ОСНОВНЫХ ИСПОЛНИТЕЛЕЙ

по Государственному контракту П1188 от 27 августа 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:

кандидат технических наук, без ученого звания	_____ подпись, дата	Гуров В. С.

Исполнители темы:

без ученой степени, без ученого звания	_____ подпись, дата	Царев Ф. Н.
без ученой степени, без ученого звания	_____ подпись, дата	Данилов В. Р.
без ученой степени, без ученого звания	_____ подпись, дата	Поликарпова Н. И.
без ученой степени, без ученого звания	_____ подпись, дата	Буздалов М. В.
без ученой степени, без ученого звания	_____ подпись, дата	Царев М. Н.
без ученой степени, без ученого звания	_____ подпись, дата	Александров А. В.

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

без ученой степени, без ученого звания _____ Казаков С. В.
подпись, дата

РЕФЕРАТ

Отчет 144 с., 3 гл., 4 прил., 65 рис., 0 табл., 30 источников.

Ключевые слова: генетическое программирование; автоматное программирование; машинное обучение; обучающие примеры; программные симуляторы летательных аппаратов.

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

Целями настоящего этапа являются:

1. Разработка и программная реализация метода построения управляющих конечных автоматов на основе обучающих примеров с помощью генетического программирования.
2. Разработка и программная реализация метода двухэтапного генетического программирования.
3. Разработка и программная реализация метода применения верификации моделей при вычислении функции приспособленности.
4. Разработка программной модели беспилотного летательного аппарата и экспериментальное исследование разработанных методов на задаче построения системы управления моделью беспилотного летательного аппарата.

При выполнении второго этапа работ использовался следующий инструментарий:

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

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

Описывается метод построения управляющих конечных автоматов на основе обучающих примеров с помощью генетического программирования. Приводится его программная реализация.

Описывается метод двухэтапного генетического программирования. Приводится его программная реализация.

Описывается метод применения верификации моделей при вычислении функции приспособленности. Приводится его программная реализация.

Приводятся результаты экспериментального исследования метод построения управляющих конечных автоматов на основе обучающих примеров с помощью генетического программирования на задаче построения автомата управления программной моделью беспилотного самолета.

В заключении дается общая оценка результатов работ по второму этапу НИР.

В приложениях приведены протоколы экспериментов, копии опубликованных в рамках НИР статей и копия экспертного заключения о возможности опубликования.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

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

Целями настоящего этапа являются:

1. Разработка и программная реализация метода построения управляющих конечных автоматов на основе обучающих примеров с помощью генетического программирования.
2. Разработка и программная реализация метода двухэтапного генетического программирования.
3. Разработка и программная реализация метода применения верификации моделей при вычислении функции приспособленности.
4. Разработка программной модели беспилотного летательного аппарата и экспериментальное исследование разработанных методов на задаче построения системы управления моделью беспилотного летательного аппарата.

Отчет имеет следующую структуру:

- первая глава содержит аналитический отчет о теоретических и экспериментальных исследованиях, проведенных на настоящем этапе;
- вторая глава содержит детальное описание результатов теоретических и экспериментальных исследований;
- третья глава содержит информацию о публикациях по результатам НИР;
- в заключении дается общая оценка работ по этапу.

В приложениях к отчету приводятся протоколы экспериментов, копии статей, экспертного заключения о возможности опубликования.

Исследования, проводимые, например, в университете Стэнфорда, показывают, что для управления беспилотным летательным аппаратом целесообразно применять модели, основанные на состояниях. Автомат управления беспилотным летательным аппаратом на верхнем уровне построить сравнительно просто – он будет содержать состояния «Посадка», «Взлет», «Набор высоты», состояния, соответствующие различным режимам полета и т.д. В то же время построение автоматов, соответствующих различным режимам полета, вручную является достаточно трудоемкой и сложной задачей.

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

В настоящее время работы по построению автоматов на основе генетического программирования ведутся в ряде зарубежных университетов, в том числе в Массачусетском технологическом институте и Университете Южной Калифорнии. Знакомство с результатами этих работ показало, что в них строятся более простые автоматы, чем те, которые требуются в системах управления беспилотными летательными аппаратами.

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

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

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

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

Указанная проблема может быть решена, если учесть специфику автоматов, применяемых в системах управления, состоящую в том, что состояния декомпозируют входные воздействия на группы, в каждую из которых обычно входит небольшое число переменных. Это позволяет строить хромосомы только для подмножества всех автоматов, что существенно сокращает пространство возможных решений, и как следствие, время поиска. Отметим, что при этом могут рассматриваться хромосомы не только для автомата в целом, но и для отдельных его состояний.

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

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

1. АНАЛИТИЧЕСКИЙ ОТЧЕТ О ПРОВЕДЕНИИ ТЕОРЕТИЧЕСКИХ И ЭКСПЕРИМЕНТАЛЬНЫХ ИССЛЕДОВАНИЙ

В настоящей главе приводится аналитический отчет о проведении теоретических и экспериментальных исследований, которые проводились по следующим направлениям:

- разработка и программная реализация метода построения управляющих конечных автоматов на основе обучающих примеров с помощью генетического программирования:
  - разработка метода представления управляющих конечных автоматов в виде хромосом алгоритма генетического программирования;
  - разработка алгоритмов реализации операций мутации и скрещивания;
  - разработка алгоритма расстановки пометок на переходах автомата на основе обучающих примеров;
- разработка и программная реализация метода двухэтапного генетического программирования:
  - разработка методов генетического программирования, осуществляющих построение репозитория состояний;
  - разработка методов генетического программирования, осуществляющих построение автоматов на основе репозитория состояний.
- разработка и программная реализация метода применения верификации моделей при вычислении функции приспособленности.

1.1. Разработка и программная реализация метода двухэтапного генетического программирования

В настоящем разделе описывается метод двухэтапного генетического программирования. Основная идея этого метода состоит в разделении процесса построения управляющего конечного автомата на две этапа: построение репозитория состояний с использованием нескольких функций приспособленности (соответствующих различным режимам работы системы) и построение автомата на основе этого репозитория состояний.

Для использования предлагаемого метода необходимо задать характеристики объекта управления и системы управления в виде двух множества: множества входных переменных X и множества выходных воздействий Z. Далее будем обозначать число входных переменных как n, а число выходных воздействий – как m. Будем также считать, что каждое входное и выходное воздействие задается вещественным числом (примерами входных переменных такого рода являются скорость, координаты, ускорение и т.д., а примерами выходных воздействий являются воздействия типа «повернуть на некоторый угол», «увеличить/уменьшить подачу топлива на некоторую величину» и т.д.).

1.1.1. Разработка методов генетического программирования, осуществляющих построение репозитория состояний

На первом этапе работы метода двухэтапного генетического программирования осуществляется построение репозитория состояний. Для этого заранее вручную программистом определяются режимы работы системы. Далее, для каждого режима определяется своя функция приспособленности, оценивающая работу систему в этом режиме. После этого для каждого режима с помощью генетического программирования строится набор управляющих функций из R^n в R^m, реализующих управление в этом режиме. Каждая из этих функций представляется в виде набора

деревье разбора, соответствующих функциям, генерирующим величину кажждого из выходных воздействий на основе входных переменных.

1.1.1.1. Метод представления функции вещественных переменных в виде деревьев разбора

Предлагаемый метод является развитием методов, описанных в работах [13] и [15]. Ограничим класс искомых отображений из R^n в R функциями, которые являются композицией некоторых базовых функций нескольких аргументов (от одного до n). Выбор базовых функций, как правило, определяется конкретной задачей. Требования, предъявляемые к набору базовых функций, таковы – набор должен быть достаточно полным, для того чтобы выразить необходимые зависимости, и не слишком избыточным, для того чтобы работа алгоритма поиска функции управления была эффективной. Можно заранее взять достаточно широкий (для задач управления, встречающихся на практике) класс функций: арифметические, показательные, логарифмические, тригонометрические, условные, вероятностные и т.д. Далее в качестве базисных функций будем использовать следующие:

- $if\ (x < y) return \sim w \sim else \sim return \sim v$;
- $\min(x, y)$;
- xy;
- $-x$;
- $\frac{1}{1+\exp(-x)}$;
- $x + y + w$;
- xyw;
- константы: 0.1, 0.5, 1, 2, 5, 10.

Дерево разбора представляет собой дерево, в котором во внутренних вершинах находятся функции. У каждой внутренней вершины ровно столько потомков, какова арность функции в данной вершине. В листья подаются входные переменные или заранее зафиксированные константы (терминалы).

Пример дерева разбора изображен на рисунке рис. 1.

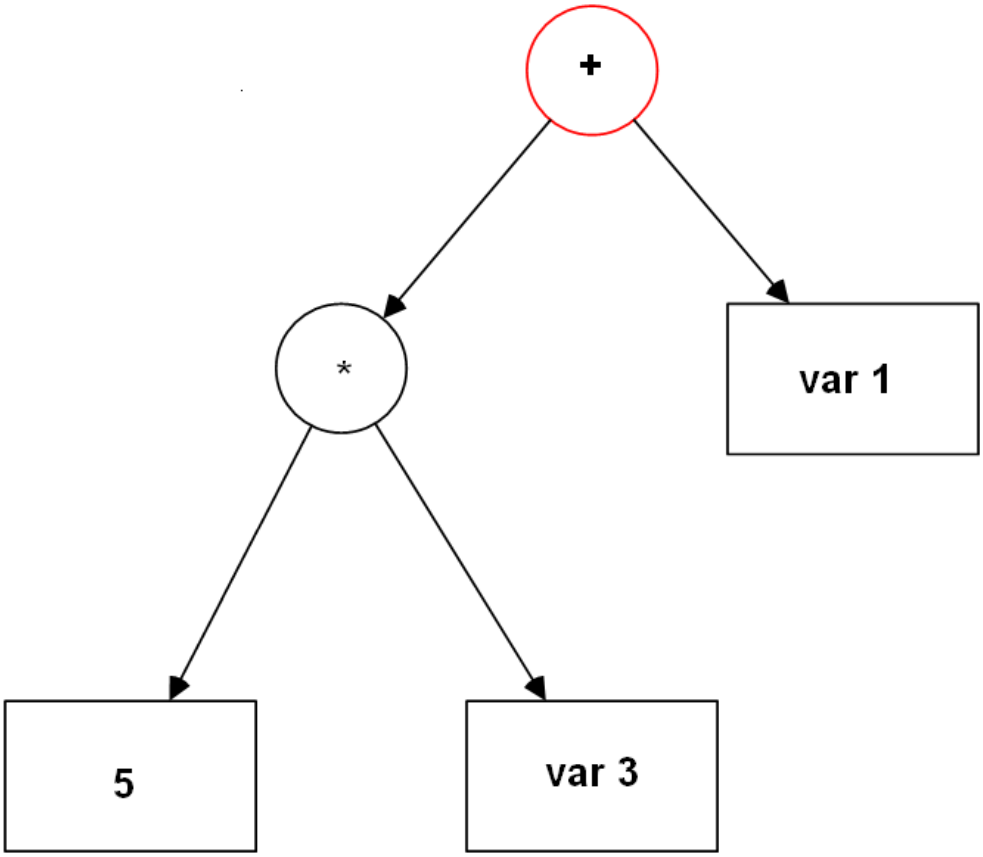


Рис. 1. Пример дерева разбора

Условное обозначение *var i* означает, что в данных лист подается значение *i*-ой переменной. Приведем исходный код класса на языке *Java*, реализующий дерево разбора:

1.1.1.2. Генерация случайного дерева

Генерация случайного дерева выполняется рекурсивным образом. Алгоритм состоит из четырех шагов:

1. С вероятностью p генерируется внутренняя вершина, иначе переходим в шагу 4.
2. Равновероятно выбираем функцию из числа базисных. Помечаем рассматриваемую вершину выбранной функцией.
3. Рекурсивно вызываем процедуру генерации нужного числа детей (это числа зависит от того, какая функция была выбрана на шаге 2). Генерация поддеревьев-детей осуществляется также с помощью описываемого алгоритма.
4. Равновероятно из множества переменных и констант выбираем одну и помещаем ее в рассматриваемую вершину.

Чтобы деревья не вырождались и не слишком разрастались, в настоящей работе вероятность p вычисляется по формуле $p = c_1 c_2^h$, где $c_1, c_2 \le 1$ – константы, а h – текущая высота генерируемой вершины.

1.1.1.3. Оператор мутации

Оператор мутации, также как и операция генерации случайного дерева, представляет собой рекурсивную процедуру.

Эта процедура является модификацией метода обхода дерева в глубину. При обработке каждой вершины дерева происходит следующее: с вероятностью *p* запускаем процедуру мутации для случайного поддерева этой вершины, иначе все поддерево, корнем которого является текущая вершина, заменяется на случайно сгенерированное.

Работа данной процедуры проиллюстрирована на рис. 2.

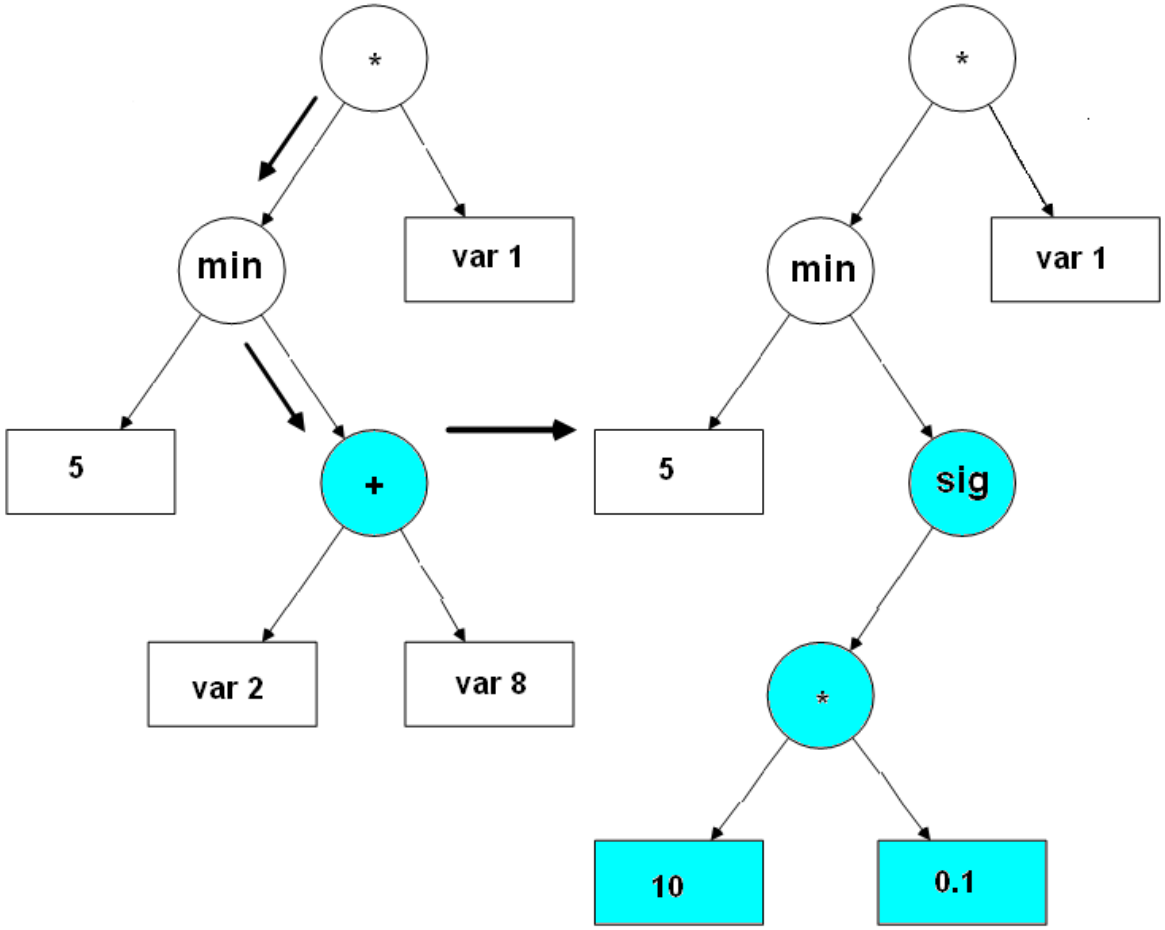


Рис. 2. Пример мутации дерева разбора

1.1.1.4. Оператор скрещивания

Данный оператор также представляет собой рекурсивную процедуру, получает на вход две особи P_1, P_2, и выдает две особи S_1, S_2.

У особей P_1, P_2 выбирается по случайному поддереву, при помощи алгоритма аналогичному, описанному в разделе 1.1.1.3. Данный алгоритм проиллюстрирован на рис. 3.

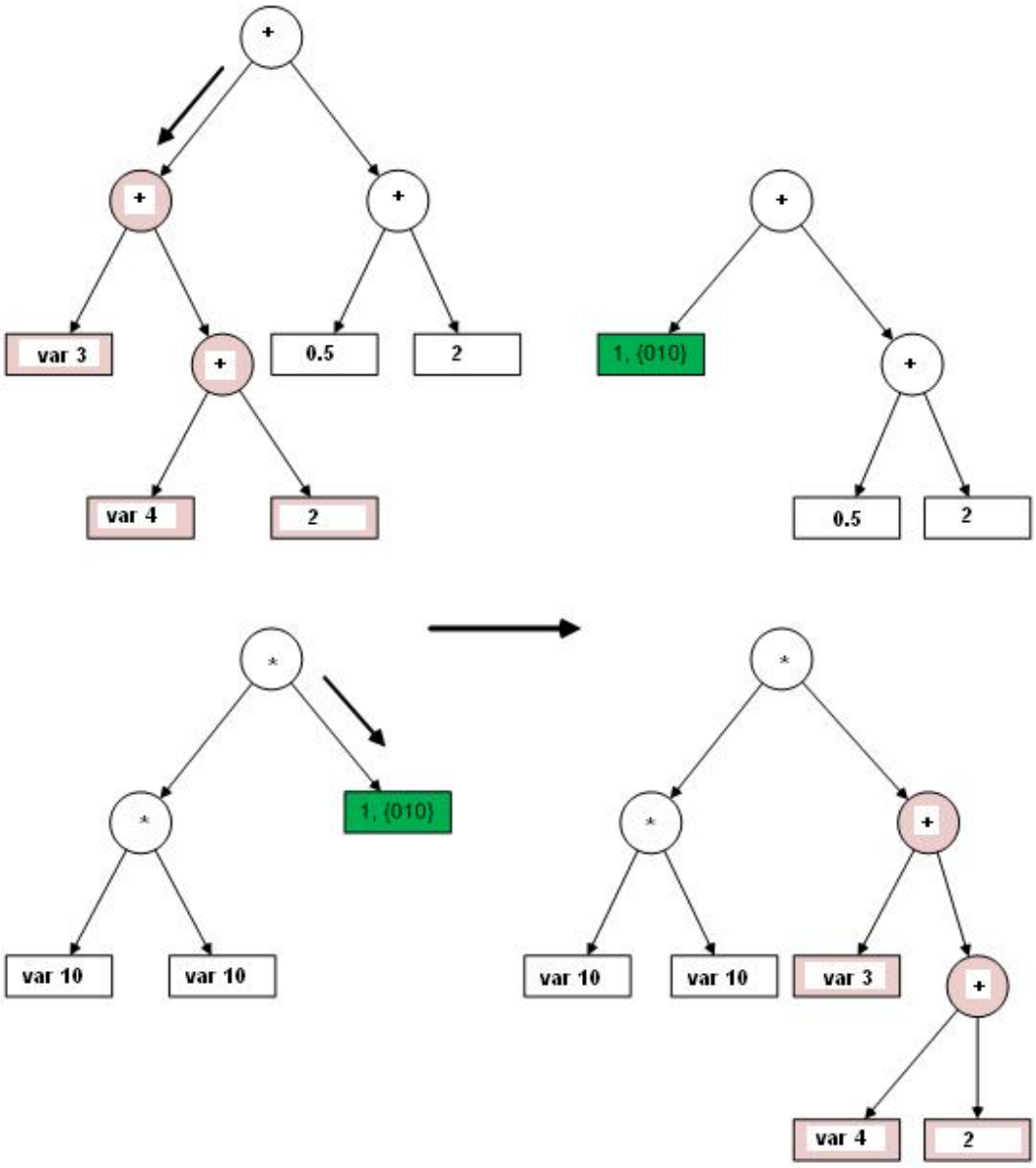


Рис. 3. Пример скрещивания деревьев разбора

1.1.1.5. Генетический алгоритм, используемый на первом этапе построения автоматов

Для построения наборов деревьев решений на первом этапе используется островной генетический алгоритм, который является расширением классического [30]. Также для избежания одной из проблем генетических алгоритмов – попадания в локальные максимумы, в предлагаемый алгоритм был добавлены еще один генетический оператор – оператор большой мутации. Общая схема данного алгоритма выглядит следующим образом (рис. 4).

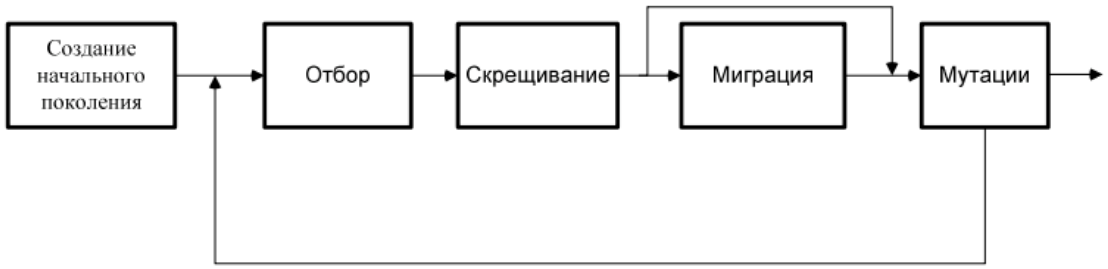


Рис. 4. Схема островного генетического алгоритма

Основные отличия островного генетического алгоритма от классического состоят в следующем:

- разделение популяции на несколько, развивающихся независимо – разделение на *острова*;
- добавление оператора миграции ([30]).

На первом этапе построения конечного автомата генетический алгоритм запускается несколько раз для каждого из режимов работы системы. Для каждого режима работы строится несколько наборов из m деревьев разбора, отвечающих за генерацию выходных воздействий. Эти наборы помещаются в так называемый репозиторий состояний, который подается на вход генетическому алгоритму, работающему на втором этапе (описан далее).

В качестве основной стратегии формирования следующего поколения используется *элитизм*. При рассмотрении текущего поколения отбрасываются все особи, кроме некоторой доли наиболее приспособленных – элиты. Эти особи переходят в следующее поколение. После этого оно дополняется в определенной пропорции случайными особями, особями из текущего поколения, которые мутировали, и результатами скрещивания особей из текущего поколения (отдельно отметим, что скрещиваться могут не только элитные особи, а все). Особи, «имеющие право» давать потомство, определяются «в поединке»: выбираются две случайные пары особей, и более приспособленная особь в каждой из них становится одним из родителей.

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

Через заранее заданное число поколений происходит *большая мутация* – фиксированная доля островов заменяется островами со случайными особями. Проведение мутации в момент, когда функция приспособленности (имеются в виду только элитные особи) изменяется незначительно, невозможно, так как за счет миграции особей между островами среднее значение функции приспособленности постоянно изменяет свое значение.

1.1.1.6. Особенности первого этапа двухэтапного алгоритма

На первом этапе генетического алгоритма выращиваются четверки деревьев разбора, или, что то же самое, конечные автоматы. Начиная с заранее заданного поколения, лучшая особь добавляется в репозиторий. При генерации стартового поколения, все острова заполняются случайно сгенерированными особями. Для каждого режима работы системы используется отдельная функция приспособленности.

1.1.2. Разработка метода генетического программирования, осуществляющего построение автоматов на основе репозитория состояний

В настоящем разделе описан метод генетического программирования, осуществляющий построение автоматов на основе репозитория состояний. Этот метод работает на втором этапе двухэтапного метода построения управляющих конечных автоматов.

1.1.2.1. Конечные автоматы, обрабатывающие вещественные переменные

Пусть необходимо построить конечный автомат из k состояний для управления системой, имеющий n входных переменных и m выходных воздействий. Этот автомат в алгоритме генетического программирования будет представляться следующим образом (рис. 5):

- для каждого из k состояний задается набор из m деревьев разбора, отвечающих за генерацию выходных воздействий;
- задачется один упорядоченный набор из $\log_2 k$ деревьев, отвечающих за вычисление функции переходов. В этих деревьев в качестве пометок вершин используются как входные переменные, так и номер состояния. Результат вычисления функций, описываемых деревьями разбора из этого набора, преобразуется в номер очередного состояния следующим образом: если результат вычисления функции неотрицателен, то ему сопоставляется единица, а иначе – ноль. После этого полученная последовательность нулей и единиц рассматривается как двоичная записб номера нового состояния.

Отметим, что при использовании предлагаемого метода число состояний k необходимо выбрать так, чтобы оно было равно точной степени двойки.

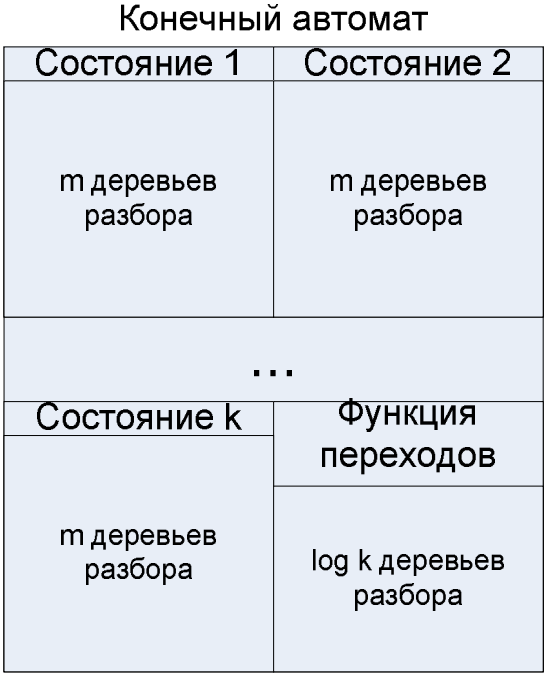


Рис. 5. Структура конечного автомата, построенного на основе деревьев разбора

Для описанного представления конечных автоматов генетические операторы были определены следующим образом.

Оператор мутации выполняет следующие действия:

- с вероятностью 0.5 изменяется стартовое состояние;
- равновероятно выбирается одно из состояний и мутирует;
- также мутируют деревья отвечающие за логические переменные.

Оператор скрещивания получает на вход две особи P_1, P_2, и выдает две особи S_1, S_2. За $P_i.n_j$ обозначим j-ое состояние автомата P_i.

Тогда оператор скрещивания выполняет следующие действия:

- стартовое состояние S_i приравнивается стартовому состоянию P_i;
- для каждого $i \in N$ проводим скрещивание для $P_1.n_j$ и $P_2.n_j$, результат обозначим за V_1, V_2. Возможны два случая:
  - $S_1.n_i = V_1$;
    $S_2.n_i = V_2$;
  - $S_1.n_i = V_2$;
    $S_2.n_i = V_1$;
- попарно скрещиваются деревья отвечающие за логические переменные, а переходы скрещиваются аналогично работе [28].

Оценим размер пространства решений в данном случае.

Напомним, что вероятность генерации внутренней вершины на высоте h выражается формулой $p = c_1 c_2^h$. Математическое ожидание числа вершин в дереве разбора E – это константа, зависящая от выбора параметров c_1, c_2.

Таким образом, число видов деревьев можно представить в виде Z^E, где Z – число различных функций, для автомата в целом получим соотношение: $|\Omega_{tree}| = (Z^E)^{V_c} (Z^E)^{4N} N^{2^{V_c}}$, где V_c – число логических переменных.

Данное пространство решений имеет меньший размер, чем у других известных методов, таким образом, при прочих равных условиях предлагаемый метод должен гораздо быстрее находить решение задачи, что полностью подтверждается практикой.

1.1.2.2. Особенности второго этапа генерации особей

Состояния конечных автоматов генерируется следующим образом: из репозитория равновероятно выбирается по одной особи, отвечающей за каждое действие. Конечные состояния переходов генерируются случайным образом. Деревья разбора, отвечающие за логические переменные в конечном автомате, представляются аналогично деревьям разбора, используемым на первом этапе. При генерации стартового поколения, все острова заполняются случайно сгенерированными особями.

Также отметим, что операторы скрещивания и мутации к деревьям разбора, находящимся *внутри* состояний не применяются, то есть построенные на первом этапе работы алгоритма деревья разбора .

Оценим размер пространства решений в случае применения двухэтапного генетического алгоритма.

Для деревьев, отвечающих за логические переменные соотношение не изменилось: $(Z^E)^{V_c}$. Также не изменилось соотношение переходов: $N^{2^{V_c}}$. Однако для генерации состояния теперь

достаточно лишь выбрать m деревьев из репозитория, в таком случае очевидно соотношение: S^m, где S – размер репозитория. Таким образом: $|\Omega_{tree(new)}| = (Z^E)^{V_c} S^m N^{2^{V_c}}$.

Очевидно что, $|\Omega_{tree(new)}| << |\Omega_{tree}|$.

1.1.3. Программная реализация метода двухэтапного генетического программирования

В настоящем разделе описывается программная реализация метода двухэтапного генетического программирования.

1.1.3.1. Принципиальная схема программной реализации

Программная реализация метода двухэтапного генетического программирования выполнялась на языке программирования *Java*. При этом в реализации содержатся классы для представления вершины дерева разбора, дерева разбора, управляющего конечного автомата, вычислителя функции приспособленности и островного генетического алгоритма (рис. 6).

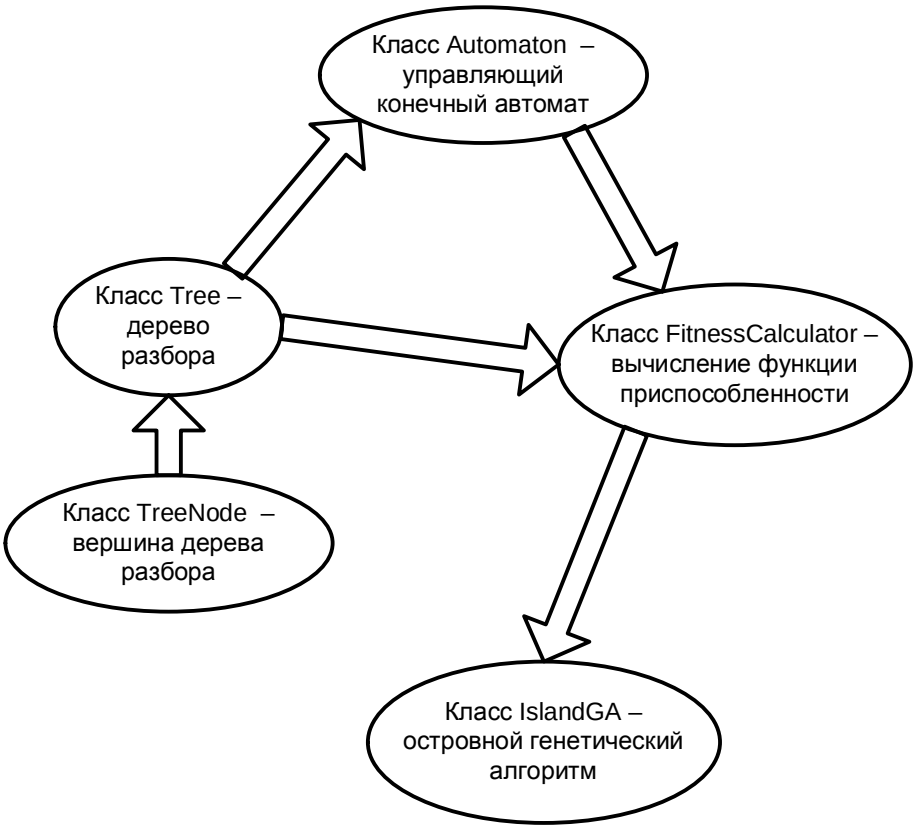


Рис. 6. Принципиальная схема программной реализации

1.1.3.2. Программная реализация дерева разбора

Дерево разбора представляется в программе в виде объекта класса `Tree`, который поддерживает операции взятия корневого узла, установки значений входных переменных, а также вычисления значения функции, задаваемой соответствующим деревом. Ниже приведен текст программной реализации этого класса.

```
public class Tree implements Operable {

    private final TreeNode start;
    private double[] values;

    public Tree(TreeNode start){
        this.start = start;
    }

    public TreeNode getStart(){
        return start;
    }

    public double getValue(double[] inValue){
        if((values == null) || (inValue != values)){
            setValues(start, inValue);
            values = inValue;
        }
        return start.getValue();
    }

    private void setValues(TreeNode v, double[] inValue){
        if(v.getType() == NodeGenerator.LEAF_TYPE){
            Leaf f = (Leaf)v;
            f.setValues(inValue);
            return;
        }
        for(int i = 0;i < v.getChildCount();i ++){
            setValues(v.getChild(i), inValue);
        }
    }

    public Tree clone(){
        return new Tree(start);
    }

    public String toString(){
        return start.toString();
    }
}

public interface TreeNode{

    public TreeNode getChild(int i);
    public int getChildCount();
    public double getValue();

    public int getType();
```

```
    }
```

1.1.3.3. Программная реализация алгоритма генерации случайного дерева

Приведем реализацию алгоритма генерации случайного дерева (разд. 1.1.1.2) на языке *Java*:

```
    public TreeNode randomTree(int curHeight, Random r){
            if(r.nextDouble() < subTreeProbability *
Math.pow(hprobabulity, curHeight)){
                int type = r.nextInt(COUNT_IN_VERTEX);
                if(type == NodeGenerator.IF_TYPE){
                    return new If(new TreeNode[]{randomTree(curHeight +
1, r), randomTree(curHeight + 1, r),
                                 randomTree(curHeight + 1, r),
randomTree(curHeight + 1, r)});
                }
                if(type == NodeGenerator.MIN_TYPE){
                    return new Min(new TreeNode[]{randomTree(curHeight +
1, r), randomTree(curHeight + 1, r)});
                }
                if(type == NodeGenerator.MULT_TYPE){
                    return new Mult(new TreeNode[]{randomTree(curHeight
+ 1, r), randomTree(curHeight + 1, r)});
                }
                if(type == NodeGenerator.NEG_TYPE){
                    return new Neg(new TreeNode[]{randomTree(curHeight +
1, r)});
                }
                if(type == NodeGenerator.SIG_TYPE){
                    return new Sig(new TreeNode[]{randomTree(curHeight +
1, r)});
                }
                if(type == NodeGenerator.SUM_TYPE){
                    return new Sum(new TreeNode[]{randomTree(curHeight +
1, r), randomTree(curHeight + 1, r)});
                }
                if(type == NodeGenerator.THREEMULT_TYPE){
                    return new ThreeMult(new
TreeNode[]{randomTree(curHeight + 1, r), randomTree(curHeight + 1, r),
randomTree(curHeight + 1, r)});
                }
                if(type == NodeGenerator.THREESUM_TYPE){
                    return new ThreeSum(new
TreeNode[]{randomTree(curHeight + 1, r), randomTree(curHeight + 1, r),
randomTree(curHeight + 1, r)});
                }
            }
            int temp = r.nextInt(countInVariables + term.length);
            if(temp < countInVariables){
```

```
                return new Leaf(temp);
            }
            return new Const(term[temp - countInVariables]);
        }
```

1.1.3.4. Программная реализация операции мутации для дерева разбора

Реализация данного алгоритма на языке *Java*:

```
    public Tree mutate(Tree ind, Random r){
            return new Tree(dfs(ind.getStart(), r, 0));
        }

        private TreeNode dfs(TreeNode node, Random r, int h){
            if((node.getChildCount() != 0) && (r.nextDouble() <
subTreeProbability)){
                int temp = r.nextInt(node.getChildCount());
                return NodeGenerator.setChild(node, temp,
dfs(node.getChild(temp), r, h + 1));
            }
            return fact.randomTree(h, r);
        }
```

Приведем реализацию описанного алгоритма на языке *Java*:

```
    public List<Tree> crossover(Tree p, Tree p1, Random r){
            Tree first = new Tree(findFirst(p.getStart(), r,
p1.getStart()));
            Tree second = new Tree(newSecondStart);
            List<Tree> res = new ArrayList<Tree>();
            res.add(first);
            res.add(second);
            return res;
        }
```

1.1.3.5. Программная реализация операции скрещивания для деревьев разбора

```
        private TreeNode findFirst(TreeNode node, Random r, TreeNode
secondStart){
            if((node.getChildCount() != 0) && (r.nextDouble() <
subTreeProbability)){
                int temp = r.nextInt(node.getChildCount());
                TreeNode newNode = findFirst(node.getChild(temp), r,
secondStart);
                return NodeGenerator.setChild(node, temp, newNode);
            }
            newSecondStart = findSecond(secondStart, r, node);
            return secondResult;
        }

        private TreeNode findSecond(TreeNode node, Random r, TreeNode
first){
```

```
            if((node.getChildCount() != 0) && (r.nextDouble() <
subTreeProbability)){
                int temp = r.nextInt(node.getChildCount());
                return NodeGenerator.setChild(node, temp,
findSecond(node.getChild(temp), r, first));
            }
            secondResult = node;
            return first;
        }
```

1.1.3.6. Программная реализация операции мутации и скрещивания для конечных автоматов

Приведем реализацию алгоритма мутации конечных автоматов на языке *Java*:

```
    public Automaton<I> mutate(Automaton<I> ind, Random r){
            Automaton<I> res = ind;
            if(r.nextBoolean()){
                res =
res.setInitialState(r.nextInt(ind.getNumberStates()));
            }
            int temp = r.nextInt(ind.getNumberStates());
            res = res.setState(temp, mut.mutate(res.getState(temp), r));
            return res;
        }
```

Приведем реализацию алгоритма скрещивания на языке *Java*:

```
    public List<Automaton<I>> crossover(Automaton<I> p, Automaton<I> p1,
Random r){
            Automaton<I> son = p;
            Automaton<I> son1 = p1;
            if(r.nextBoolean()){
                son = son.setInitialState(p1.getInitialState());
                son1 = son1.setInitialState(p.getInitialState());
            }
            for(int i = 0;i < p.getNumberStates();i ++){
                if(r.nextBoolean()){
                    List<I> st = cross.crossover(p.getState(i),
p1.getState(i), r);
                    son = son.setState(i, st.get(0));
                    son1 = son1.setState(i, st.get(1));
                }else{
                    I temp = son.getState(i);
                    son = son.setState(i, son1.getState(i));
                    son1 = son1.setState(i, temp);
                }
            }
            List<Automaton<I>> res = new ArrayList<Automaton<I>>();
            res.add(son);
            res.add(son1);
            return res;
```

```
}
```

1.2. Разработка и программная реализация метода построения управляющих конечных автоматов на основе обучающих примеров с помощью генетического программирования

В настоящем разделе описывается метод построения управляющих конечных автоматов на основе обучающих примеров с помощью генетического программирования и его программная реализация. На рис. 7 представлена общая схема построения управляющего автомата с использованием описываемого метода.

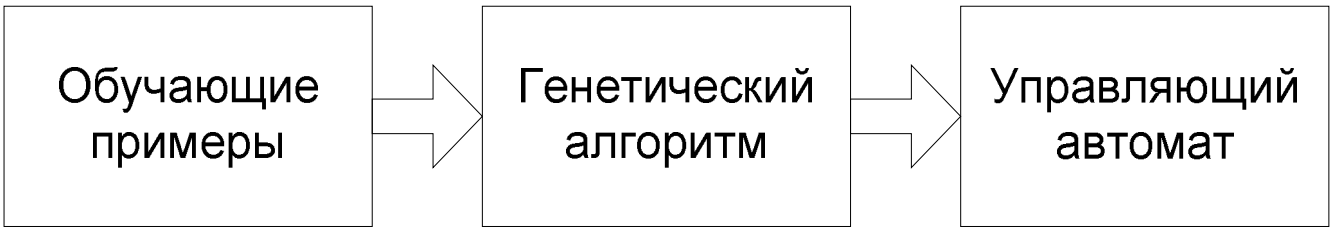


Рис. 7. Общая схема построения управляющего автомата

Исходными данными для построения управляющего конечного автомата является набор обучающих примеров (тестов), структура которых подробно описана ниже. Эти обучающие примеры создаются человеком.

Задача алгоритма генетического программирования состоит в построении конечного автомата, который задает поведение самолета, достаточно хорошо соответствующее на обучающих примерах эталонному. Если использовать достаточно большое число обучающих примеров, то появляется возможность избавиться от мелких неточностей, которые допускает при управлении человек. В настоящей работе число тестов достигало 40 при некоторых пробных запусках генетического алгоритма и порядка 10 при выращивании автоматов для решения поставленных задач.

1.2.1. Структура обучающего примера

При моделировании полета беспилотного летательного аппарата сохраняются значения параметров полета (скорость, направление полета и т.д.) и параметров состояния самолета (положение руля, элеронов, состояние стартера и т.п.). Параметры полета будем далее называть входными (для системы управления) параметрами, а параметры состояния – управляющими (за счет их изменения осуществляется управление самолетом).

При этом непрерывные действия изменяют соответственные параметры на некоторую вещественную величину, а дискретные – устанавливают соответствующий параметр в конкретное значение. Заметим, что последовательное выполнение действий с одним параметром эквивалентно сумме действий в случае непрерывного параметра и последнему действию – в случае дискретного.

Например, одним из непрерывных управляющих параметров является угол поворота руля. Непрерывным действием над ним является изменение угла поворота руля на некоторое значение. Тогда последовательность поворотов руля на x и y градусов эквивалентна повороту руля на $(x+y)$ градусов.

В свою очередь, хорошим примером дискретного управляющего параметра является состояние стартера. Дискретным действием над ним является его включение или выключение. Тогда последовательность включений и выключений стартера эквивалентна последнему совершенному над ним действию.

Схема обучающего примера приведена на рис. 8.

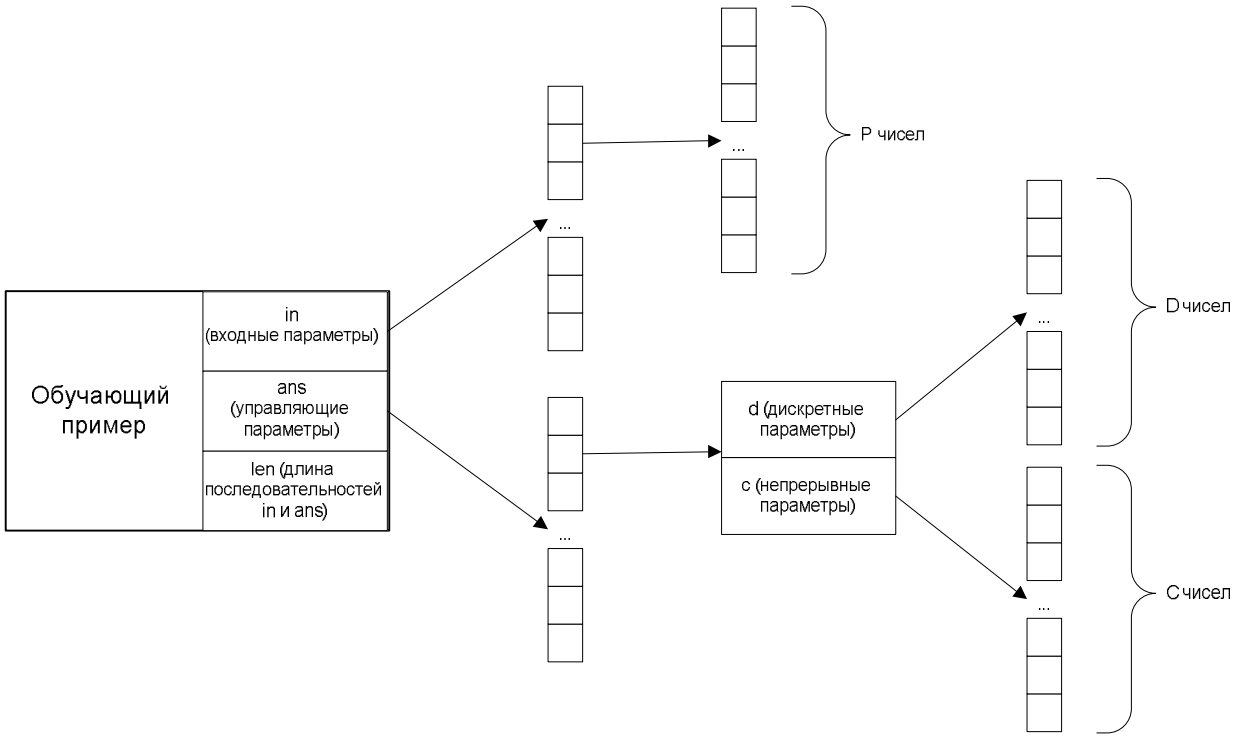


Рис. 8. Схема обучающего примера

Обучающий пример T состоит из двух частей: $T.in$ и $T.ans$. Каждая из них является последовательностью длиной $T.len$ – первая из них состоит из значений входных параметров, а вторая – из соответствующих эталонных значений управляющих параметров, которые записываются в ходе экспериментов, проводимых человеком.

Каждый элемент $T.in[t]$ последовательности входных параметров состоит из P чисел – значений этих параметров в момент времени t.

Элемент $T.ans[t]$ включает в себя два набора: $T.ans[t].d$ и $T.ans[t].c$. Последовательность $T.ans[t].d$ состоит из D дискретных параметров, а $T.ans[t].c$ – из C непрерывных. Таким образом, $T.ans[t]$ состоит из $(D + C)$ чисел.

Обозначим через $T.out$ набор управляющих параметров, выданных автоматом на обучающем примере T. Структура $T.out$ совпадает со структурой $T.ans$.

1.2.2. Разработка метода представления управляющих конечных автоматов в виде хромосом алгоритма генетического программирования

Конечный автомат в генетическом алгоритме представляется в виде объекта, который содержит описания переходов для каждого из состояний и номер начального состояния, причем число состояний автомата фиксировано и является параметром генетического алгоритма. Для каждого состояния хранится список переходов. Для каждого перехода задается условие, при котором он выполняется. Это условие имеет вид «X» или «$\neg X$», где X – некоторое утверждение о состоянии

самолета (например, «двигатель включен»). При этом, для переходов в особи не задаются выходные воздействия.

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

Проверяемые утверждения о состоянии самолета составляются вручную до запуска генетического алгоритма и не изменяются в течение его работы.

Работа автомата осуществляется следующим образом. На каждом такте система управления получает значения всех входных параметров. После этого в определенном порядке вычисляются логические значения всех утверждений о состоянии самолета. После вычисления каждого из значений, оно подается на вход конечному автомату. При этом в начале каждого такта автомату обязательно подается на вход утверждение $x1$, означающее начало такта. Автомат в течение одного такта может совершить несколько переходов, и результирующее действие автомата в каждый момент времени t может быть составлено из нескольких последовательно выполненных действий на отдельных переходах. Напомним, что каждый параметр результирующего действия – сумма действий в случае непрерывного параметра, и последнее действие – в случае дискретного.

На рис. 9 изображен возможный «скелет» автомата.

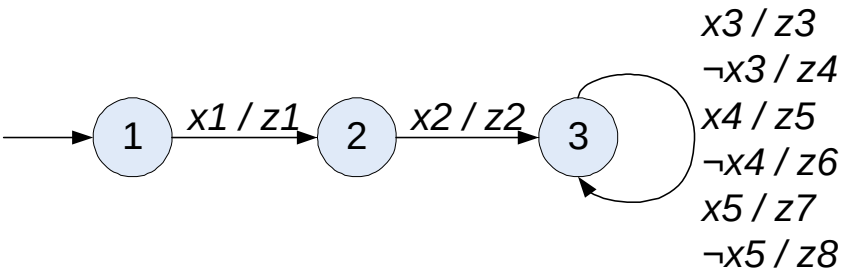


Рис. 9. «Скелет» автомата, поддерживающего постоянное направление движения

На этом рисунке утверждения о состоянии самолета имеют следующий смысл:

$x1$ – начало такта;

$x2$ – двигатель включен;

$x3$ – отклонение направления движения от курса меньше нуля;

$x4$ – скорость изменения направления меньше нуля;

$x5$ – ускорение изменения направления меньше нуля.

Выходные воздействия $z1$, …, $z8$, указанные на рис. 3, в «скелете» автомата не заданы – конкретные выходные воздействия определяются перед вычислением функции приспособленности с помощью алгоритма расстановки пометок.

Если для данного «скелета» выходные воздействия задать следующим образом:

$z1$ – включить стартер;

$z2$ – выключить стартер, поставить обороты двигателя на максимум;

$z3$/$z4$ – повернуть руль вправо/влево на $0,01$;

$z5$/$z6$ – повернуть руль вправо/влево на $0,005$;

$z7$/$z8$ – повернуть руль вправо/влево на $0,0025$,

то получится автомат, поддерживающий постоянное направление движения. Такой автомат был построен вручную.

1.2.3. Отбор

Операция отбора необходима для выделения из текущего поколения наиболее подходящих для решения задачи особей и помещения их в промежуточное поколение. Для сравнения особей вводится функция приспособленности, сопоставляющая каждой особи число, характеризующее насколько хорошо автомат, соответствующий особи, подходит для решения задачи. Чем больше это число, тем лучшей считается особь.

В настоящей работе используются *отбор методом рулетки* и *турнирным методом* [12].

В первом методе сначала для каждой особи рассчитывается вероятность ее выбора по формуле:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j},$$

где f_i – значение функции приспособленности i-ой особи, N – число особей в поколении. Затем в соответствии с полученным распределением алгоритм случайным образом выбирает особь из текущего поколения и добавляет ее в промежуточное поколение. Так делается столько раз, сколько особей требуется выбрать.

В турнирном методе выбор новой особи выполняется следующим образом. Случайным образом выбирается k особей из текущего поколения. Между ними проводится турнир, и определяется лучшая особь, которая и добавляется в промежуточное поколение. В данной реализации $k = 2$.

1.2.4. Вычисление функции приспособленности

Для вычисления функции приспособленности на вход автомату подается каждая из последовательностей `T[i].in` – входной набор `i`-го обучающего примера. Затем записывается последовательность выходных воздействий `T[i].out`, которую сгенерировал автомат. После этого значение функции приспособленности вычисляется по формуле

$$Fitness = 1 - \sqrt{\frac{1}{N}\sum_{i=1}^{N}\frac{Dist(T[i].out, T[i].ans)^2}{Dist(T[i].ans, 0)^2}} =$$

$$= 1 - \sqrt{\frac{1}{N}\left(\frac{Dist(T[1].out, T[1].ans)^2}{Dist(T[1].ans, 0)^2} + \frac{Dist(T[2].out, T[2].ans)^2}{Dist(T[2].ans, 0)^2} + \ldots + \frac{Dist(T[N].out, T[N].ans)^2}{Dist(T[N].ans, 0)^2}\right)}$$

Здесь под `Dist(out, ans)` понимается «расстояние» между выходной и эталонной последовательностями управляющих параметров, которое вычисляется по формуле

$$Dist(out, ans) = \sqrt{\sum_{t=1}^{T.len}\left(\sum_{k=1}^{D}\left[out[t].d[k] \neq ans[t].d[k]\right] + \sum_{k=1}^{C}\left(out[t].c[k] - ans[t].c[k]\right)^2\right)}$$

Перед вычислением функции приспособленности к «скелету» автомата применяется алгоритм расстановки действий. Этот алгоритм расставляет действия на переходах так, чтобы максимизировать значение функции приспособленности при заданном «скелете» автомата.

1.2.5. Разработка алгоритма расстановки действий на переходах автомата на основе обучающих примеров

Для расстановки действий на переходах скелету автомата на вход подается входной набор параметров из каждого обучающего примера. При этом отмечаются переходы, совершаемые автоматом.

1.2.5.1. Случай одного теста

Рассмотрим случай одного теста `T` (`N = 1`). Тогда функция приспособленности имеет вид:

$$Fitness = 1 - \sqrt{\frac{Dist(T.out, T.ans)^2}{Dist(T.ans, 0)^2}}$$

Она достигает максимума, когда расстояние $Dist(T.out, T.ans)^2$ минимально, следовательно, необходимо минимизировать сумму:

$$\sum_{t=1}^{T.len}\left(\sum_{k=1}^{D}\left[T.out[t].d[k] \neq T.ans[t].d[k]\right] + \sum_{k=1}^{C}\left(T.out[t].c[k] - T.ans[t].c[k]\right)^2\right) =$$

$$= \sum_{k=1}^{D}\sum_{t=1}^{T.len}\left[T.out[t].d[k] \neq T.ans[t].d[k]\right] + \sum_{k=1}^{C}\sum_{t=1}^{T.len}\left(T.out[t].c[k] - T.ans[t].c[k]\right)^2$$

Заметим, что сумму для каждого параметра можно минимизировать независимо. Значит, достаточно минимизировать суммы вида

$$\sum_{t=1}^{T.len}\left[T.out[t].d[k] \neq T.ans[t].d[k]\right] \quad (1)$$

для каждого `k` от `1` до `D` и суммы вида

$$\sum_{t=1}^{T.len}\left(T.out[t].c[m] - T.ans[t].c[m]\right)^2 \quad (2)$$

для каждого `m` от `1` до `C`. Далее будем считать индексы `k` и `m` зафиксированными.

1.2.5.2. Расстановка дискретных действий для случая одного теста

Напомним, что значение `T.out[t].d[k]` – это значение на последнем выполненном переходе в момент времени `t`. Поэтому сумму (1) можно переписать так:

$$\sum_{i=1}^{n}\sum_{t\in Time_i}\left[T.out[t].d[k]\neq T.ans[t].d[k]\right]=\sum_{t\in Time_1}\left[T.out[t].d[k]\neq T.ans[t].d[k]\right]+$$

$$+\sum_{t\in Time_2}\left[T.out[t].d[k]\neq T.ans[t].d[k]\right]+\ldots+\sum_{t\in Time_n}\left[T.out[t].d[k]\neq T.ans[t].d[k]\right] \quad (1.1)$$

Здесь за $Time_i$ обозначено множество таких значений времён t, при которых номер последнего выполненного перехода был равен i, за n – число переходов в автомате.

Для минимизации суммы (1.1) достаточно минимизировать суммы вида

$$\sum_{t\in Time_i}\left[T.out[t].d[k]\neq T.ans[t].d[k]\right] \quad (1.2)$$

для каждого i от 1 до n. Зафиксируем i. $T.out[t].d[k]$ при $t\in Time_i$ равно значению k-ого дискретного параметра на переходе i, которое не зависит от t. Обозначим его через z. Получим:

$$\sum_{t\in Time_i}\left[z\neq T.ans[t].d[k]\right] \quad (1.3)$$

Сгруппируем слагаемые с одинаковым значением $T.ans[t].d[k]$. Тогда имеем:

$$\sum_{t\in TimeV_1\subset Time_i}\left[z\neq v_1\right]+\sum_{t\in TimeV_2\subset Time_i}\left[z\neq v_2\right]+\ldots+\sum_{t\in TimeV_G\subset Time_i}\left[z\neq v_G\right]=$$

$$=\left[z\neq v_1\right]\cdot\sum_{t\in TimeV_1\subset Time_i}1+\left[z\neq v_2\right]\cdot\sum_{t\in TimeV_2\subset Time_i}1+\ldots+\left[z\neq v_G\right]\cdot\sum_{t\in TimeV_G\subset Time_i}1=$$

$$=\left[z\neq v_1\right]\cdot\left|TimeV_1\right|+\left[z\neq v_2\right]\cdot\left|TimeV_2\right|+\ldots+\left[z\neq v_G\right]\cdot\left|TimeV_G\right| \quad (1.4)$$

Здесь v_j – j-ое значение k-ого дискретного параметра, G – число возможных значений этого параметра, $TimeV_j$ есть множество тех моментов времени t из $Time_i$, при которых $T.ans[t].d[k]$ равно v_j.

Выбрав в качестве значения дискретного параметра на i-ом переходе v_j, мы получим значение суммы (1.4) равное $\left|Time_i\right|-\left|TimeV_j\right|$, так как все коэффициенты $\left[z\neq v_s\right]$ равны единице тогда и только тогда, когда $s\neq j$, а $\sum_{j=1}^{G}\left|TimeV_j\right|=\left|Time_i\right|$. Поэтому для минимизации суммы (1.4) необходимо выбрать то значение v_j, при котором $\left|TimeV_j\right|$ максимально, – наиболее часто встречаемое значение среди $T.ans[t].d[k]$ при $t\in Time_i$.

1.2.5.3. Расстановка непрерывных действий для случая одного теста

Для минимизации сумм вида (2) напомним, что величина изменения m-ого непрерывного параметра равна сумме изменений этого параметра на всех выполненных переходах. При этом на каждом из них рассматриваемый параметр меняется на определенную (постоянную для этого перехода), но неизвестную величину. Обозначим величину изменения параметра на i-ом переходе за

z_i. Тогда итоговое значение параметра равно сумме его начального значения (равного нулю) и его изменений на каждом выполненном переходе:

$$T.out[t].c[k] = \sum_{i=1}^{n} \alpha_i[t] z_i$$

Здесь за $\alpha_i[t]$ обозначено число раз, которое был выполнен *i*-ый переход к моменту времени *t*.

Таким образом, сумму (2) можно переписать в виде:

$$S = \sum_{t=1}^{T.len} \left(\sum_{i=1}^{n} \alpha_i[t] z_i - T.ans[t].c[k] \right)^2 \quad (2.1)$$

При этом ее производная по z_j имеет вид:

$$S'_{z_j} = \sum_{t=1}^{T.len} 2\alpha_j[t] \left(\sum_{i=1}^{n} \alpha_i[t] z_i - T.ans[t].c[k] \right).$$

Для того чтобы найти точку, в которой сумма *S* принимает минимальное значение, необходимо найти точку, в которой производные *S* по z_j для всех *j* были равны нулю. Таким образом, получаем систему линейных уравнений вида:

$$\begin{cases} \sum_{i=1}^{n} \left(\sum_{t=1}^{T.len} \alpha_1[t]\alpha_i[t] \right) z_i = \sum_{t=1}^{T.len} T.ans[t].c[k]\alpha_1[t] \\ \sum_{i=1}^{n} \left(\sum_{t=1}^{T.len} \alpha_2[t]\alpha_i[t] \right) z_i = \sum_{t=1}^{T.len} T.ans[t].c[k]\alpha_2[t] \\ \dots \\ \sum_{i=1}^{n} \left(\sum_{t=1}^{T.len} \alpha_n[t]\alpha_i[t] \right) z_i = \sum_{t=1}^{T.len} T.ans[t].c[k]\alpha_n[t] \end{cases}$$

В данной работе эти *C* систем (для *k* от *1* до *C*) решаются методом Гаусса, полученные z_j – искомые величины изменения параметра *k* на переходе *j*.

1.2.5.4. Случай нескольких тестов

Обобщим метод на случай нескольких тестов.

Напомним, что формула вычисления функции приспособленности имеет следующий вид:

$$Fitness = 1 - \sqrt{\frac{1}{N}\left(\frac{Dist(T[1].out, T[1].ans)^2}{Dist(T[1].ans, 0)^2} + \frac{Dist(T[2].out, T[2].ans)^2}{Dist(T[2].ans, 0)^2} + \dots + \frac{Dist(T[N].out, T[N].ans)^2}{Dist(T[N].ans, 0)^2} \right)}$$

Она достигает максимума, когда сумма

$$\frac{Dist(T[1].out, T[1].ans)^2}{Dist(T[1].ans, 0)^2} + \frac{Dist(T[2].out, T[2].ans)^2}{Dist(T[2].ans, 0)^2} + \dots + \frac{Dist(T[N].out, T[N].ans)^2}{Dist(T[N].ans, 0)^2}$$

минимальна.

Замечая, как и ранее, что суммы по каждому параметру можно минимизировать независимо друг от друга, получаем, что необходимо минимизировать суммы вида

$$\frac{\sum_{t=1}^{T[1].len}\left[T[1].out[t].d[k] \neq T[1].ans[t].d[k]\right]}{Dist(T[1].ans,0)^2} + \frac{\sum_{t=1}^{T[2].len}\left[T[2].out[t].d[k] \neq T[2].ans[t].d[k]\right]}{Dist(T[2].ans,0)^2} + \\ + \ldots + \frac{\sum_{t=1}^{T[N].len}\left[T[N].out[t].d[k] \neq T[N].ans[t].d[k]\right]}{Dist(T[N].ans,0)^2} \quad (3)$$

для каждого k от *1* до D и суммы вида

$$\frac{\sum_{t=1}^{T[1].len}\left(T[1].out[t].c[m] - T[1].ans[t].c[m]\right)^2}{Dist(T[1].ans,0)^2} + \frac{\sum_{t=1}^{T[2].len}\left(T[2].out[t].c[m] - T[2].ans[t].c[m]\right)^2}{Dist(T[2].ans,0)^2} + \\ + \ldots + \frac{\sum_{t=1}^{T[N].len}\left(T[N].out[t].c[m] - T[N].ans[t].c[m]\right)^2}{Dist(T[N].ans,0)^2} \quad (4)$$

для m от *1* до C. Далее считаем индексы k и m зафиксированными.

1.2.5.5. Расстановка дискретных действий для случая нескольких тестов

Переписав (3), имеем:

$$\sum_{i=1}^{n}\sum_{t\in Time_{i,1}}\frac{\left[T[1].out[t].d[k] \neq T[1].ans[t].d[k]\right]}{Dist(T[1].ans,0)^2} + \sum_{i=1}^{n}\sum_{t\in Time_{i,2}}\frac{\left[T[2].out[t].d[k] \neq T[2].ans[t].d[k]\right]}{Dist(T[2].ans,0)^2} + \\ + \ldots + \sum_{i=1}^{n}\sum_{t\in Time_{i,N}}\frac{\left[T[N].out[t].d[k] \neq T[N].ans[t].d[k]\right]}{Dist(T[N].ans,0)^2} = \\ = \sum_{i=1}^{n}\sum_{j=1}^{N}\sum_{t\in Time_{i,j}}\frac{\left[T[j].out[t].d[k] \neq T[j].ans[t].d[k]\right]}{Dist(T[j].ans,0)^2} \quad (3.1)$$

Здесь за $Time_{i,j}$ обозначено множество таких моментов времени t, при которых номер последнего выполненного перехода автомата, запущенного на тесте j, был равен i, за n – число переходов в автомате.

Таким образом, для каждого i от *1* до n необходимо минимизировать

$$\sum_{j=1}^{N}\sum_{t\in Time_{i,j}}\frac{\left[T[j].out[t].d[k] \neq T[j].ans[t].d[k]\right]}{Dist(T[j].ans,0)^2} \quad (3.2)$$

Зафиксируем *i*. Как и в случае одного теста, *T[j].out[t].d[k]* при $t \in Time_{i,j}$ равно значению *k*-ого дискретного параметра на переходе *i*, обозначим его через *z*.

Сгруппируем слагаемые с одинаковым значением *T[j].ans[t].d[k]*, получим:

$$\sum_{j=1}^{N}\sum_{t\in TimeV_{i,j,1}}\frac{[z\neq v_1]}{Dist(T[j].ans,0)^2}+\sum_{j=1}^{N}\sum_{t\in TimeV_{i,j,2}}\frac{[z\neq v_2]}{Dist(T[j].ans,0)^2}+\ldots+$$

$$+\sum_{j=1}^{N}\sum_{t\in TimeV_{i,j,G}}\frac{[z\neq v_G]}{Dist(T[j].ans,0)^2}=\sum_{j=1}^{N}\frac{[z\neq v_1]}{Dist(T[j].ans,0)^2}\sum_{t\in TimeV_{i,j,1}}1+$$

$$+\sum_{j=1}^{N}\frac{[z\neq v_2]}{Dist(T[j].ans,0)^2}\sum_{t\in TimeV_{i,j,2}}1+\ldots+\sum_{j=1}^{N}\frac{[z\neq v_G]}{Dist(T[j].ans,0)^2}\sum_{t\in TimeV_{i,j,G}}1=$$

$$=\sum_{j=1}^{N}\frac{[z\neq v_1]}{Dist(T[j].ans,0)^2}\left|TimeV_{i,j,1}\right|+\sum_{j=1}^{N}\frac{[z\neq v_2]}{Dist(T[j].ans,0)^2}\left|TimeV_{i,j,2}\right|+\ldots+$$

$$+\sum_{j=1}^{N}\frac{[z\neq v_G]}{Dist(T[j].ans,0)^2}\left|TimeV_{i,j,G}\right|=$$

$$=\sum_{j=1}^{N}\frac{1}{Dist(T[j].ans,0)^2}\left([z\neq v_1]\cdot\left|TimeV_{i,j,1}\right|+[z\neq v_2]\cdot\left|TimeV_{i,j,2}\right|+\ldots+[z\neq v_G]\cdot\left|TimeV_{i,j,G}\right|\right) \quad (3.3)$$

Здесь v_j – *j*-ое значение *k*-ого дискретного параметра, *G* – количество значений этого параметра, $TimeV_{i,j,h}$ есть множество тех моментов времени *t* из $Time_{i,j}$, при которых *T.ans[t].d[k]* равно v_h.

Выбрав в качестве значения дискретного параметра на *i*-том переходе v_k, мы получим следующее значение суммы:

$$\sum_{j=1}^{N}\frac{1}{Dist(T[j].ans,0)^2}\left(\left|Time_{i,j}\right|-\left|TimeV_{i,j,k}\right|\right)=$$

$$\sum_{j=1}^{N}\frac{1}{Dist(T[j].ans,0)^2}\left|Time_{i,j}\right|-\sum_{j=1}^{N}\frac{1}{Dist(T[j].ans,0)^2}\left|TimeV_{i,j,k}\right| \quad (3.4)$$

Для минимизации суммы (3.4) необходимо выбрать то значение v_k, при котором $\sum_{j=1}^{N}\frac{1}{Dist(T[j].ans,0)^2}\left|TimeV_{i,j,k}\right|$ максимально.

1.2.5.6. Расстановка непрерывных действий для случая нескольких тестов

Вспомнив, что величина изменения *k*-ого непрерывного параметра в момент времени *t* (*T[i].out[t].c[k]*) равна сумме изменений этого параметра на всех выполненных переходах к этому моменту, из формулы (4) получаем:

$$S = \sum_{i=1}^{N} \frac{1}{Dist(T[i].ans,0)^2} \sum_{t=1}^{T[i].len} \left(\sum_{j=1}^{n} \alpha_{i,j}[t] z_j - T[i].ans[t].c[k] \right)^2 \quad (4.1)$$

Здесь за $\alpha_{i,j}[t]$ обозначено число раз, которое был выполнен *j*-ый переход к моменту времени *t* автомата, запущенного на тесте *i*.

Производная *S* по z_r имеет вид:

$$S'_{z_r} = \sum_{i=1}^{N} \frac{1}{Dist(T[i].ans,0)^2} \sum_{t=1}^{T[i].len} 2\alpha_{i,r}[t] \left(\sum_{j=1}^{n} \alpha_{i,j}[t] z_j - T[i].ans[t].c[k] \right)$$

Приравняв все S'_{z_r} к нулю, получим систему *n* уравнений:

$$\begin{cases} \sum_{j=1}^{n} \left(\sum_{i=1}^{N} \frac{1}{Dist(T[i].ans,0)^2} \sum_{t=1}^{T[i].len} \alpha_{i,1}[t]\alpha_{i,j}[t] \right) z_j = \sum_{i=1}^{N} \frac{1}{Dist(T[i].ans,0)^2} \sum_{t=1}^{T[i].len} T[i].ans[t].c[k]\alpha_{i,1}[t] \\ \sum_{j=1}^{n} \left(\sum_{i=1}^{N} \frac{1}{Dist(T[i].ans,0)^2} \sum_{t=1}^{T[i].len} \alpha_{i,2}[t]\alpha_{i,j}[t] \right) z_j = \sum_{i=1}^{N} \frac{1}{Dist(T[i].ans,0)^2} \sum_{t=1}^{T[i].len} T[i].ans[t].c[k]\alpha_{i,2}[t] \\ \dots \\ \sum_{j=1}^{n} \left(\sum_{i=1}^{N} \frac{1}{Dist(T[i].ans,0)^2} \sum_{t=1}^{T[i].len} \alpha_{i,n}[t]\alpha_{i,j}[t] \right) z_j = \sum_{i=1}^{N} \frac{1}{Dist(T[i].ans,0)^2} \sum_{t=1}^{T[i].len} T[i].ans[t].c[k]\alpha_{i,n}[t] \end{cases}$$

Этот набор систем решается так же, как и в случае одного теста. Полученные z_j – искомые величины изменений рассматриваемого непрерывного *k*-го параметра на переходе *j*.

1.2.6. Разработка алгоритмов реализации операций мутации и скрещивания

В настоящем разделе описаны алгоритмы скрещивания и мутации для описанной выше структуры особей.

1.2.6.1. Операция мутации

При выполнении операции мутации с заданной вероятностью (по умолчанию, она равна 0,5) выполняется каждое из действий:

- изменение начального состояния;
- добавление, удаление или изменение (изменение состояния, в которое осуществляется переход по условию) конкретного (случайно выбранного) перехода.

1.2.6.2. Операция скрещивания

В скрещивании принимают участие две особи. При этом результатом операции скрещивания также являются две особи. Обозначим «родительские» особи как *P1* и *P2*, а «дочерние» – *C1* и *C2*. Тогда для стартовых состояний *C1.is* и *C2.is* справедливо одно из следующих утверждений:

- *C1.is = P1.is* и *C2.is = P2.is*;
- *C1.is = P2.is* и *C2.is = P1.is*.

Далее для каждой ячейки таблицы переходов *t[i][j]* с равной вероятностью выбирается один из следующих вариантов:

- *C1.t[i][j] = P1.t[i][j]* и *C2.t[i][j] = P2.t[i][j];*
- *C1.t[i][j] = P2.t[i][j]* и *C2.t[i][j] = P1.t[i][j].*

На рис. 10 представлена схема распределения переходов между дочерними особями.

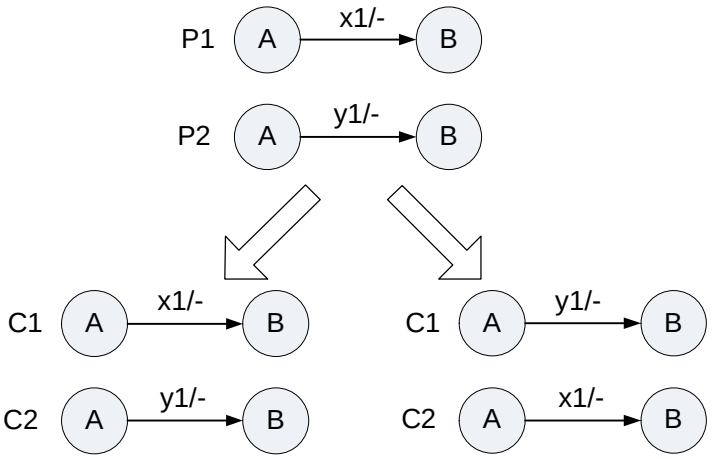


Рис. 10. Схема распределения переходов между дочерними особями

1.2.7. Программная реализация метода построения управляющих конечных автоматов на основе обучающих примеров с помощью генетического программирования

Программная реализация разработанного метода построения управляющих автоматов выполнялась на языке программирования *Java*. Основные классы и связи между ними представлены на рис. 11.

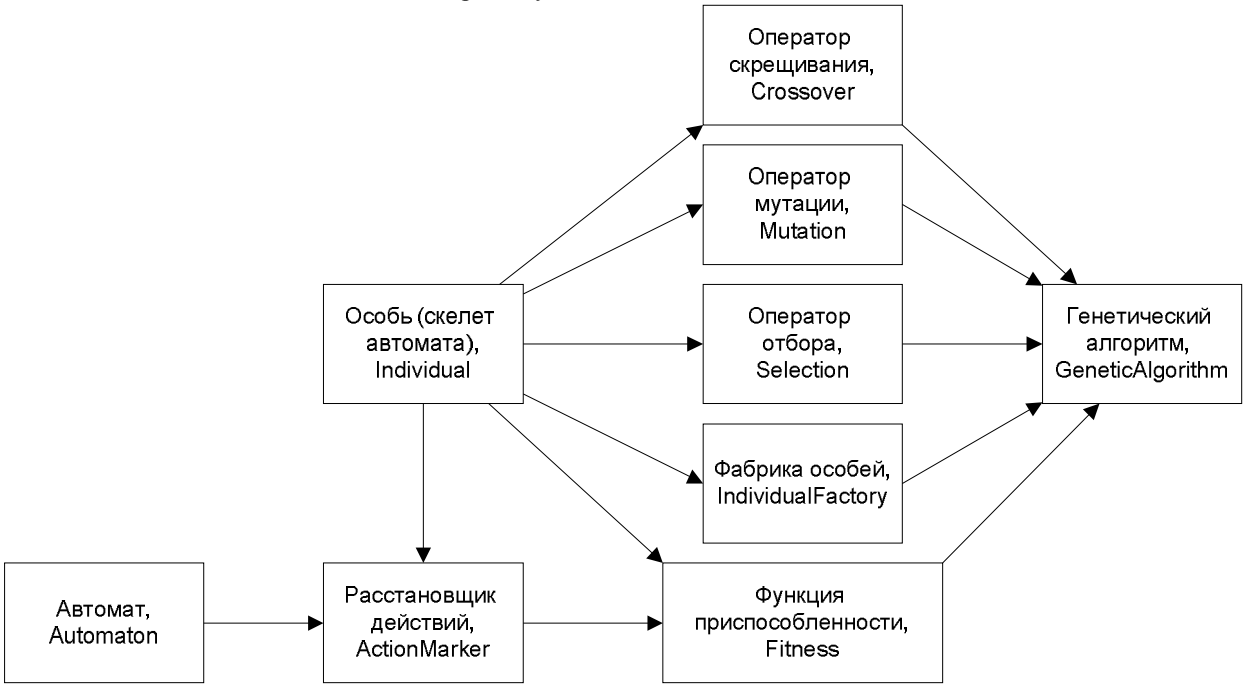


Рис. 11. Основные классы и связи между ними

Класс `GeneticAlgorithm` реализует генетический алгоритм, описанный в разделе 2. Создание начального поколения (шаг 1) происходит в конструкторе:

```
public GeneticAlgorithm (Random r, int generationSize,
        int eliteSize, IndividualFactory factory,
        Fitness foo, Crossover c, Mutation m,
        Selection s, double mutationProbability,
   Properties[][] props, Controls[][] ctrls);
```

отбор, скрещивание и мутация (шаги 2–4) совершаются в методе

```
public void nextGeneration().
```

Управление самолетом с помощью автомата происходит по схеме, представленной на рис. 12.

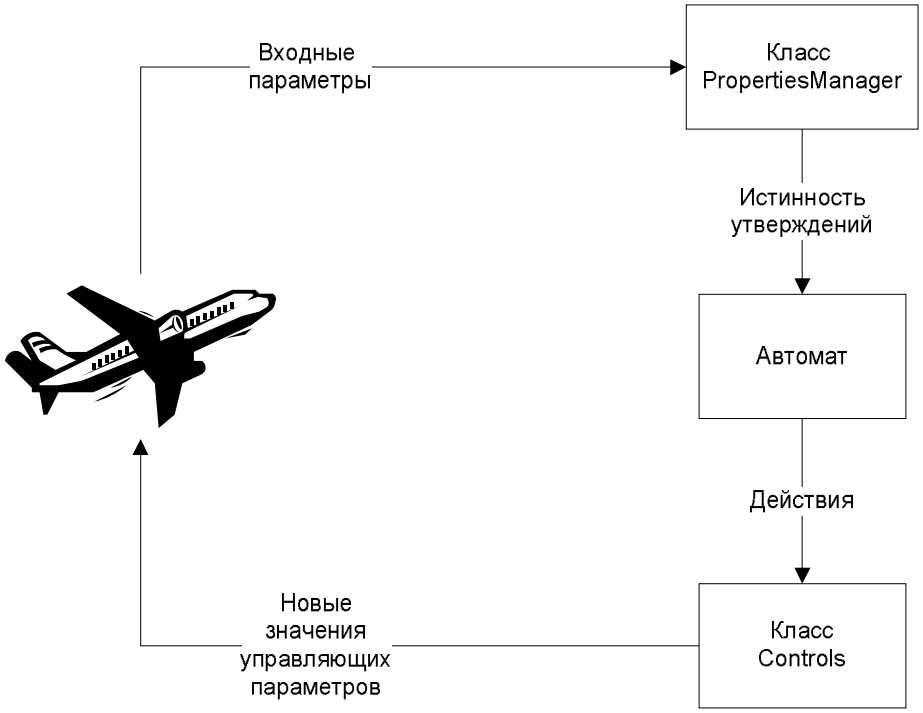


Рис. 12. Управление моделью беспилотного самолета с помощью автомата

1.2.7.1. Реализация оператора отбора

Интерфейс `Selection` определяет методы, которые должен содержать класс, реализующий тот или иной способ отбора:

```
public interface Selection {
     public FitIndividual[] apply(FitIndividual[] population,
          int selectionSize);
}
```

Этот интерфейс следующим образом реализуют классы `TournamentSelection` и `RouletteSelection`:

```
public class TournamentSelection implements Selection {
    private final Random r;

    public TournamentSelection(Random r) {
        this.r = r;
    }

    @Override
    public FitIndividual[] apply(FitIndividual[] population, int
selectionSize) {
        final int n = population.length;
        final int t = Math.max(2, (int) Math.round(n / (double)
selectionSize));
```

```
            FitIndividual[] res = new FitIndividual[selectionSize];
            for (int i = 0; i < selectionSize; i++) {
                FitIndividual w = null;
                for (int j = 0; j < t; j++) {
                    FitIndividual nw = population[r.nextInt(n)];
                    if ((w == null) || (nw.fitness[0] > w.fitness[0])) {
                        w = nw;
                    }
                }
                res[i] = w;
            }

            return res;
        }
    }

    public class RouletteSelection implements Selection {
        private final Random r;

        public RouletteSelection(Random r) {
            this.r = r;
        }

        @Override
        public FitIndividual[] apply(FitIndividual[] population, int
selectionSize) {
            final int n = population.length;
            final double[] weight = new double[n];
            weight[0] = population[0].fitness[0];
            for (int i = 1; i < n; i++) {
                weight[i] = weight[i - 1] + population[i].fitness[0];
            }

            FitIndividual[] res = new FitIndividual[selectionSize];
            for (int i = 0; i < selectionSize; i++) {
                double p = weight[n - 1] * r.nextDouble();
                int j = Arrays.binarySearch(weight, p);
                if (j >= 0) {
                    j++;
                } else {
                    j = - j - 1;
                }
                res[i] = population[j];
            }

            return res;
        }
```

```
    }
```

1.2.7.2. Реализация оператора скрещивания

Интерфейс `Crossover` определяет методы, которые должен содержать класс, реализующий тот или иной способ скрещивания.

```
    public interface Crossover {
              public List<Individual> apply(Individual p1, Individual
p2);
    }
```

Этот интерфейс следующим образом реализует класс `UniformCrossover`:

```
    public class UniformCrossover implements Crossover {
        private final static double DEFAULT_EXCANGE_PROBABILITY = 0.5;

        private Random r;
        private double exchangeProbability;

        public UniformCrossover(Random r) {
            this(r, DEFAULT_EXCANGE_PROBABILITY);
        }

        public UniformCrossover(Random r, double exchangeProbability) {
            this.r = r;
            this.exchangeProbability = exchangeProbability;
        }

        public List<Individual> apply(Individual p1, Individual p2) {
            if ((p1.rowsNumber != p2.rowsNumber) || (p1.columnsNumber !=
p2.columnsNumber))
                throw new IllegalArgumentException();
            Individual c1 = new Individual(p1);
            Individual c2 = new Individual(p2);
            for (int i = 0; i < c1.rowsNumber; ++i) {
                for (int j = 0; j < c1.columnsNumber; ++j) {
                    if (r.nextDouble() < exchangeProbability) {
                        Transition t1 = c1.transitions[i][j];
                        Transition t2 = c2.transitions[i][j];

                        c1.transitions[i][j] = t2;
                        c2.transitions[i][j] = t1;
                    }
                }
            }
            List<Individual> l = new ArrayList<Individual>();
            l.add(c1);
            l.add(c2);
```

```
            return l;
        }
    }
```

1.2.7.3. Реализация оператора мутации

Интерфейс `Mutation` определяет методы, которые должен содержать класс, реализующий тот или иной способ мутации.

```
    public interface Mutation {
        public Individual apply(Individual ind);
    }
```

Примером реализации этого интерфейса является класс `SimpleMutation`:

```
    public class SimpleMutation implements Mutation {
        private static final double INITIAL_STATE_CHANGE_PROBABILITY =
0.1;
        private static final double DOUBLE_FUZZINESS = 0.01;
        private static final double ANNUL_PROBABILITY = 0.5;
        private static final int INT_FUZZINESS = 1;

        Random r;

        public SimpleMutation(Random r) {
            this.r = r;
        }

        @Override
        public Individual apply(Individual ind) {
            Individual res = new Individual(ind);

            int i = r.nextInt(ind.rowsNumber);
            int j = r.nextInt(ind.columnsNumber);

            Action newAction = null;
            if (ind.withActions) {
                if (ind.transitions[i][j] == null) {
                    newAction = new Action(r);
                } else {
                    newAction = new Action();
                    for (int k = 0; k < Controls.INT_CONTROLS_NUMBER;
++k) {
                        newAction.setIntAction(k,
ind.transitions[i][j].action.getIntAction(k) +
                                                r.nextInt(INT_FUZZINESS*2
+ 1) - INT_FUZZINESS);
                    }
```

```
                    for (int k = 0; k < Controls.DOUBLE_CONTROLS_NUMBER;
++k) {
                        newAction.setDoubleAction(k,
ind.transitions[i][j].action.getDoubleAction(k) +

r.nextGaussian()*DOUBLE_FUZZINESS);
                    }
                }
            }
            Transition t = new Transition(r.nextInt(ind.rowsNumber),
newAction);
            if (r.nextDouble() < ANNUL_PROBABILITY)
                t = null;
            res.transitions[i][j] = t;


            if (r.nextDouble() < INITIAL_STATE_CHANGE_PROBABILITY) {
                res.initialState = r.nextInt(res.rowsNumber);
            }

            return res;
        }
    }
```

1.2.7.4. Реализация метода расстановки действий

```
    public class ActionMarker {
        public static void mark(Markable m, Properties[][] props,
Controls[][] controls) {
            Event[][] events = new Event[props.length][];
            EventGenerator eg = new EventGenerator();
            for (int i = 0; i < props.length; ++i)
            {
                events[i] = new Event[props[i].length];
                PropertiesManager pm = new PropertiesManager();
                for (int j = 0; j < props[i].length; ++j)
                {
                    pm.setCurrentProperties(props[i][j]);
                    events[i][j] = eg.generate(pm);
                }
            }
            mark(m, events, controls);
        }
        public static void mark(Markable m, Event[][] events,
Controls[][] controls) {
            Map<Integer, Integer> id2var = new HashMap<Integer,
Integer>();
            List<Integer> var2id = new ArrayList<Integer>();
            int varsCount = 0;
```

```
            for (int i = 0; i < events.length; ++i)
            {
                m.init();
                for (int j = 0; j < events[i].length; ++j)
                {
                    ArrayList<Integer> ids = m.process(events[i][j]);
                    for (int id: ids)
                    {
                        if (!id2var.containsKey(id)) {
                            id2var.put(id, varsCount++);
                            var2id.add(id);
                        }
                    }
                }
            }

            if (varsCount == 0)
                return;

            double[][] a = new double[varsCount][varsCount];
            double[][] b = new
double[varsCount][Controls.DOUBLE_CONTROLS_NUMBER];

            int[][] mostFreq = new
int[Controls.INT_CONTROLS_NUMBER][varsCount];

            Map<Integer,Double>[][] stat = new
Map[Controls.INT_CONTROLS_NUMBER][varsCount];
            for (int i = 0; i < Controls.INT_CONTROLS_NUMBER; ++i) {
                for (int j = 0; j < varsCount; ++j)
                {
                    stat[i][j] = new HashMap<Integer, Double>();
                }
            }

            for (int i = 0; i < events.length; ++i)
            {
                double testNorm2 = 0;
                for (int j = 0; j < controls[i].length; ++j)
                    testNorm2 += Math.pow(controls[i][j].norm(), 2);

                m.init();
                int[] actionsSum = new int[varsCount];
                int lastAction = -1;
                for (int j = 0; j < events[i].length; ++j) {
                    ArrayList<Integer> ids = m.process(events[i][j]);
                    for (int id: ids) {
                        int var = id2var.get(id);
```

```
                        actionsSum[var]++;
                        lastAction = var;
                    }
                    if (lastAction == -1)
                        continue;
                    for (int c = 0; c < Controls.INT_CONTROLS_NUMBER;
++c) {
                        int var = lastAction;
                        int curCtrl = controls[i][j].getIntControl(c);
                        Double prev = stat[c][var].get(curCtrl);
                        if (prev == null)
                            prev = 0.;
                        Double updated = prev+1/testNorm2;
                        stat[c][var].put(curCtrl, updated);
                        if (stat[c][var].size() == 1) {
                            mostFreq[c][var] = curCtrl;
                        } else {
                            int prevFreq = mostFreq[c][var];
                            if (updated > stat[c][var].get(prevFreq)) {
                                mostFreq[c][var] = curCtrl;
                            }
                        }
                    }
                    for (int v1 = 0; v1 < varsCount; ++v1) {
                        for (int v2 = 0; v2 < varsCount; ++v2) {
                            a[v1][v2] +=
(double)actionsSum[v1]*actionsSum[v2]/testNorm2;
                        }
                    }
                    for (int c = 0; c < Controls.DOUBLE_CONTROLS_NUMBER;
++c) {
                        for (int v1 = 0; v1 < varsCount; ++v1) {
                            b[v1][c] +=
actionsSum[v1]*controls[i][j].getDoubleControl(c)/testNorm2;
                        }
                    }
                }
            }

            Action[] actions = new Action[varsCount];
            for (int i = 0; i < varsCount; ++i)
                actions[i] = new Action();

            double[][] res = Util.solve(a, b);
            for (int c = 0; c < Controls.DOUBLE_CONTROLS_NUMBER; ++c) {
                for (int i = 0; i < varsCount; ++i) {
                    actions[i].setDoubleAction(c, res[i][c]);
                }
```

```
            }

            for (int c = 0; c < Controls.INT_CONTROLS_NUMBER; ++c) {
                for (int i = 0; i < varsCount; ++i) {
                    actions[i].setIntAction(c, mostFreq[c][i]);
                }
            }

            for (int i = 0; i < varsCount; ++i) {
                m.setAction(var2id.get(i), actions[i]);
            }
        }
    }
```

1.3. Разработка и программная реализация метода применения верификации моделей при вычислении функции приспособленности

Исходными данными для метода применения верификации при вычислении функции приспособленности:

- 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, описываемый в этом разделе, основан на методе построения конечных автоматов на основе обучающих примеров из работы [21], однако предлагаемый подход использует результат верификации в генетическом алгоритме на стадии вычисления функции приспособленности. 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, строя управляющий конечный автомат таким образом (на основе тестов и верификации), мы можем заранее описать ее требуемое поведение.

1.3.1. 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 [21].

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» управляющего конечного автомата, а конкретные выходные воздействия, вырабатываемые на переходах, определяются с помощью алгоритма расстановки пометок.

В настоящей работе генетический алгоритм остался таким же, как и в работе [21], 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.

1.3.2. 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]*».

1.3.3. 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 [21], 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) [20]. 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 темоpальных 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 [6]. Значит, автомат-пересечение будет содержать $n \times 2^n \times V$ состояний, где V – число вершин в модели. Заметим, что верификатор использует средство *LTL2BA* [12] 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.

1.3.4. 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 $\mathrm{FF_{LTL}} = \sum_{i=1}^{n} \frac{t_i}{T} \Big/ n$, 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. 13. Цифрой «1» выделена часть автомата, на которой не удалось обнаружить контрпример, а цифрой «2» – часть автомата, на которой формула опровергается.

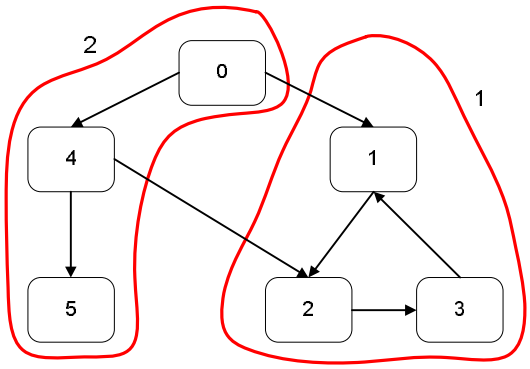


Рис. 13. Пример вычисления вклада LTL-формулы в функцию приспособленности

Таким образом, вклад данной формулы будет $\mathrm{FF_i} = \frac{3}{7}$, 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. 13) мог быть найден сразу. Тогда вклад формулы в функцию приспособленности оказался бы нулевым, так как ни один из переходов не был бы помечен.

1.3.5. Операции скрещивания и мутации

Операция скрещивания может выполняться одним из двух способов: случайное и по тестам. Эти два способа описаны в работе [21].

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.

Операция мутации также осуществляется методом, описанным в работе [21].

1.3.6. Методика построения управляющих конечных автоматов и использованием

Для сравнения методов построения модели на основе тестов и на основе одновременно тестов и темпоральных свойств, сначала приведем методику построения управляющих конечных автоматов только на основе тестов (рис. 14).

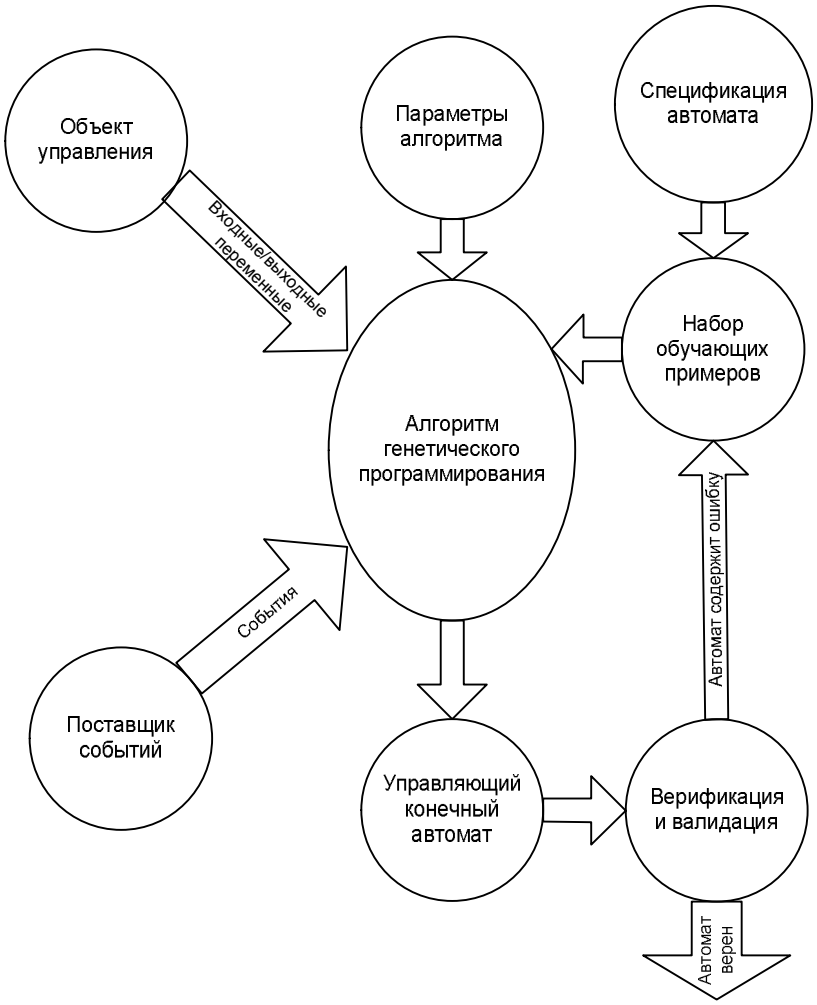


Рис. 14. Методика построения автоматных программ на основе тестов с помощью генетических алгоритмов

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. 15.

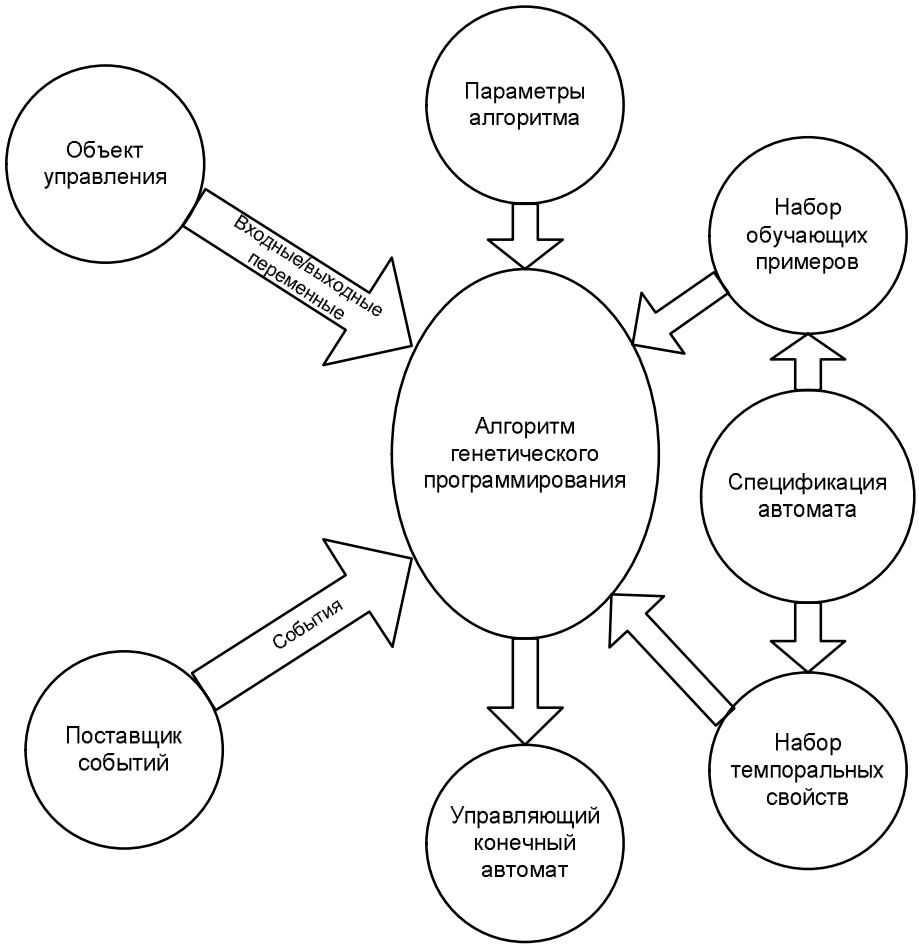


Рис. 15. Методика создания автоматных программ на основе тестов и верификации

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. 15, 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.

1.3.7. Программная реализация

Предлагаемый метод реализован на языке *Java* и является дополнением к реализации из работы [21]. К предыдущей работе были добавлены два пакета: для чтения тестов, формул и для их верификации. Диаграмма пакетов приведена на рис. 16.

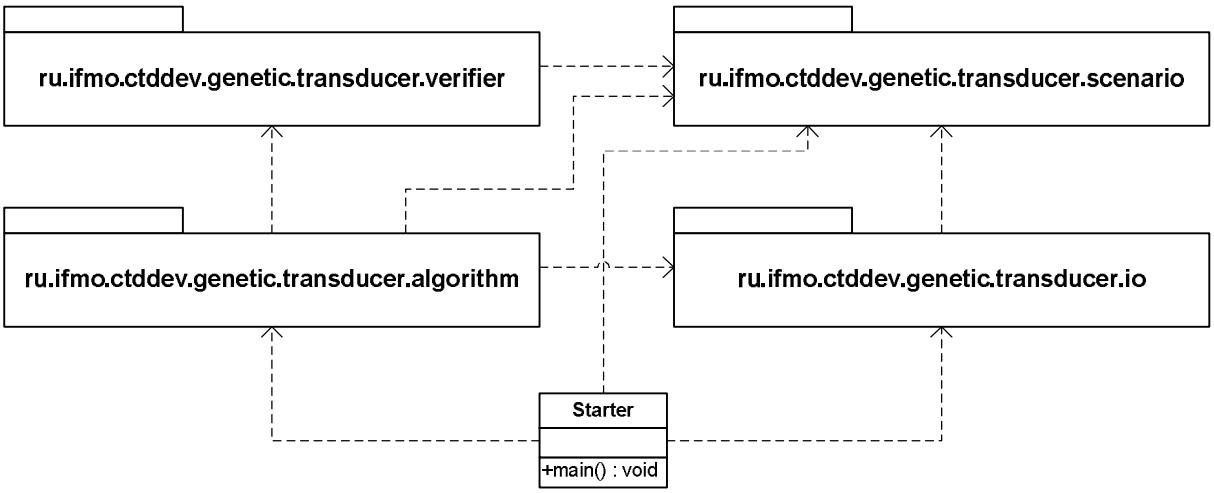


Рис. 16. Диаграмма пакетов программы

В пакет `ru.ifmo.ctddev.genetic.transducer.scenario` добавлен класс `TestGroup`, который хранит набор тестов и *LTL*-формул для конкретной группы. Теперь генетический алгоритм на вход получает не список тестов, а список объектов данного класса.

Класс `FitnessCalculator` реализует вычисление функции приспособленности на основе редакционного расстояния и относительного числа успешно выполненных формул.

```
public double calcFitness(FST fst) {
  fst.doLabelling(this.tests);

  verifier.configureStateMachine(fst.getStates(),
                                 fst.getInitialState());
  int[] verRes = verifier.verify();

  double res = 0;
  int i = 0;
  for (TestGroup group : groups) {
    double sum = 0;
```

```
      int cntOk = 0;

      for (Path p : group.getTests()) {
        String[] output = fst.transform(p.getInput());
        String[] answer = p.getOutput();
        double t;
        if (output == null) {
          t = 1;
        } else {
          t = editDistance(output, answer)
              / Math.max(output.length, answer.length);
        }
        sum += 1 - t;
        if (same(output, answer)) {
          cntOk++;
        }
      }
      int testsSize = group.getTests().size();
      int formulasSize = group.getFormulas().size();

      if (testsSize > 0) {
        res += (cntOk == testsSize)
            ? TESTS_COST
            : 0.5 * TESTS_COST * (sum / testsSize);
      }
      if (formulasSize > 0) {
        res += FORMULAS_COST * verRes[i] / formulasSize;
      }
      i++;
    }
    return res + 0.01 * (100 - fst.getTransitionsCount());
  }
```

Сначала выполняется алгоритм расстановки пометок [21], после этого полученный автомат с проставленными выходными воздействиями верифицируется, и определяется число успешно «пройденных» формул для каждой группы. Также при верификации помечаются переходы, принадлежащие самому длинному контрпримеру. После этого вычисляется редакционное расстояние для тестов в каждой группе, и суммируются вклады тестов и темпоральных свойств.

Класс FST, реализующий конечный автомат Мили, хранит номер начального состояния, число состояний, набор переходов для каждого состояния, набор входных событий и набор выходных воздействий.

```
  private final int initialState;
  private final int stateNumber;
  private Transition[][] states;

  private final String[] setOfInputs;
  private final String[] setOfOutputs;
```

В этом классе также реализованы операции мутации, скрещивания и алгоритм расстановки пометок. Их реализацию можно уточнить в работе [21]. В настоящей же работе в данный класс было добавлено удаление перехода из пути при мутации, опровергающего *LTL*-формулу.

Класс `Transition` представляет собой переход в автомате, он хранит входное воздействие, число выходных воздействий и номер состояния, в которое перейдет автомат по этому переходу. Класс представляет набор методов, но в данной работе интересен тот, который копирует переход в новую особь при скрещивании. Он реализован следующим образом:

```
public Transition copy(String[] setOfInputs,
                       String[] setOfOutputs,
                       int stateNumber) {
  if (isUsedByVerifier()
        && !used
        && (RANDOM.nextDouble() < 0.1)) {
    return new Transition(
        setOfInputs[RANDOM.nextInt(setOfInputs.length)],
        mutateOutputSize(outputSize, setOfOutputs.length),
        RANDOM.nextInt(stateNumber));
  }
  return new Transition(input, outputSize, newState);
}
```

В добавленном пакете `ru.ifmo.ctddev.genetic.transducer.io` расположены классы для чтения тестов и *LTL*-формул из *xml*-файла и записи модели в файл в формате *UniMod* [17].

Класс `TestsReader` позволяет читать из `xml`-файла параметры алгоритма (вероятность мутации, размер популяции, ожидаемое число состояний, число поколений до мутации и т.д.), набор тестов и темпоральных свойств, разбитых по группам. Класс `OneGroupTestsReader` позволяет читать входные данные для метода, игнорируя разбивку по группам (создавая всего одну группу).

Класс `UnimodModelWriter` сохраняет особь в файл в формате *UniMod* `xml`-описания модели. В результате, созданную автоматную модель можно сразу запускать в инструментальном средстве *UniMod*, при условии, что реализованы поставщики событий и объекты управления.

В пакете `ru.ifmo.ctddev.genetic.transducer.verifier` находятся классы, относящиеся к верификации, трансляции формул в автомат Бюхи и преобразованию особи генетического алгоритма в объект-автомат, принимаемый верификатором.

Класс `ModifiableAutomataContext` является представлением верифицируемой автоматной программы. Он содержит объект управления, поставщик событий и конечный автомат модели. Его особенность в том, что конечный автомат может меняться, так как у каждой особи он свой, в то время как поставщик событий и объект управления остаются неизменными и создаются один раз за время генерации конечного автомата. Приведем поля из данного класса, они являются интерфейсами, описанными в используемом верификаторе.

```
private IControlledObject co;
private IEventProvider ep;
private IStateMachine<IState> machine;
```

Класс `VerifierFactory` предоставляет методы для трансляции формул в автомат Бюхи, для представления особи в виде автомата, принимаемого используемым верификатором, и метод для верификации. Причем трансляция формул происходит только один раз при запуске программы, так как формулы в процессе работы генетического алгоритма не меняются, а значит, их можно

преобразовывать в автомат Бюхи только один раз. После верификации модели соответствующий метод возвращает число пройденных тестов для каждой группы и помечает самый длинный путь в автомате, являющийся контрпримером для одной из формул.

```
void prepareFormulas(TestGroup[] groups);
void configureStateMachine(FST fst);
int[] verify();
```

Вызов верификации предполагает, что модель была преобразована в модель, принимаемую на вход верификатором, для этого необходимо вызвать метод `void configureStateMachine(FST fst).` После этого автомат верифицируется методом `int[] verify()`. Который возвращает число верных формул для каждой из группы. Приведем код данного метода:

```
public int[] verify() {
  int[] res = new int[preparedFormulas.length];
  IVerifier<IState> verifier = new SimpleVerifier<IState>(
      context.getStateMashine(null).getInitialState());
  List<IIntersectionTransition> longestList =
      Collections.emptyList();

  for (int i = 0; i < preparedFormulas.length; i++) {
    int formulasOk = 0;

    for (IBuchiAutomata buchi : preparedFormulas[i]) {
      List<IIntersectionTransition> list =
          verifier.verify(buchi, predicates);
      if ((list == null) || (list.size() == 0)) {
        formulasOk++;
      } else {
        if (longestList.size() < list.size()) {
          longestList = list;
        }
      }
    }
    res[i] = formulasOk;
  }

  for (IIntersectionTransition t : longestList) {
    AutomataTransition trans =
        (AutomataTransition) t.getTransition();
    if ((trans != null)
        && (trans.getAlgTransition() != null)) {
      trans.getAlgTransition().setUsedByVerifier(true);
    }

  return res;
}
```

Класс `AutomataTransition` является представлением перехода для верификатора. Также данный класс сохраняет ссылку на переход в особи, чтобы по контрпримеру верификатора совершить обратное преобразование в переход модели.

2. РЕЗУЛЬТАТЫ ТЕОРЕТИЧЕСКИХ И ЭКСПЕРИМЕНТАЛЬНЫХ ИССЛЕДОВАНИЙ

В настоящей главе описываются результаты теоретических и экспериментальных исследований, проводившихся в рамках II этапа работ по настоящему государственному контракту. Перечислим основные направления теоретических и экспериментальных исследований:

- разработка и программная реализация метода построения управляющих конечных автоматов на основе обучающих примеров с помощью генетического программирования:
  - разработка метода представления управляющих конечных автоматов в виде хромосом алгоритма генетического программирования;
  - разработка алгоритмов реализации операций мутации и скрещивания;
  - разработка алгоритма расстановки пометок на переходах автомата на основе обучающих примеров;
- разработка и программная реализация метода двухэтапного генетического программирования:
  - разработка методов генетического программирования, осуществляющих построение репозитория состояний;
  - разработка методов генетического программирования, осуществляющих построение автоматов на основе репозитория состояний.
- разработка и программная реализация метода применения верификации моделей при вычислении функции приспособленности;
- разработка программной модели беспилотного летательного аппарата и экспериментальное исследование разработанных методов на задаче построения системы управления моделью беспилотного летательного аппарата.

Экспериментальное исследование метода построения управляющих конечных автоматов на основе обучающих примеров с помощбю генетических алгоритмов проводилось на двух задачах – генерация автомата, управляющего разгоном самолета, и автомата, выполняющего «мертвую петлю».

2.1. Выбор авиасимулятора для моделирования беспилотного летательного аппарата

Существует множество авиасимуляторов, довольно правдоподобно моделирующих полет самолета. Каждый из них обладает своими преимуществами и недостатками. Авторами был выбран свободный кроссплатформенный авиасимулятор *FlightGear* (http://www.flightgear.org), который позволяет осуществлять программное управление самолетом, взаимодействуя с системой управления через сетевое соединение. На рис. 17 представлен снимок экрана авиасимулятора *FlightGear*.

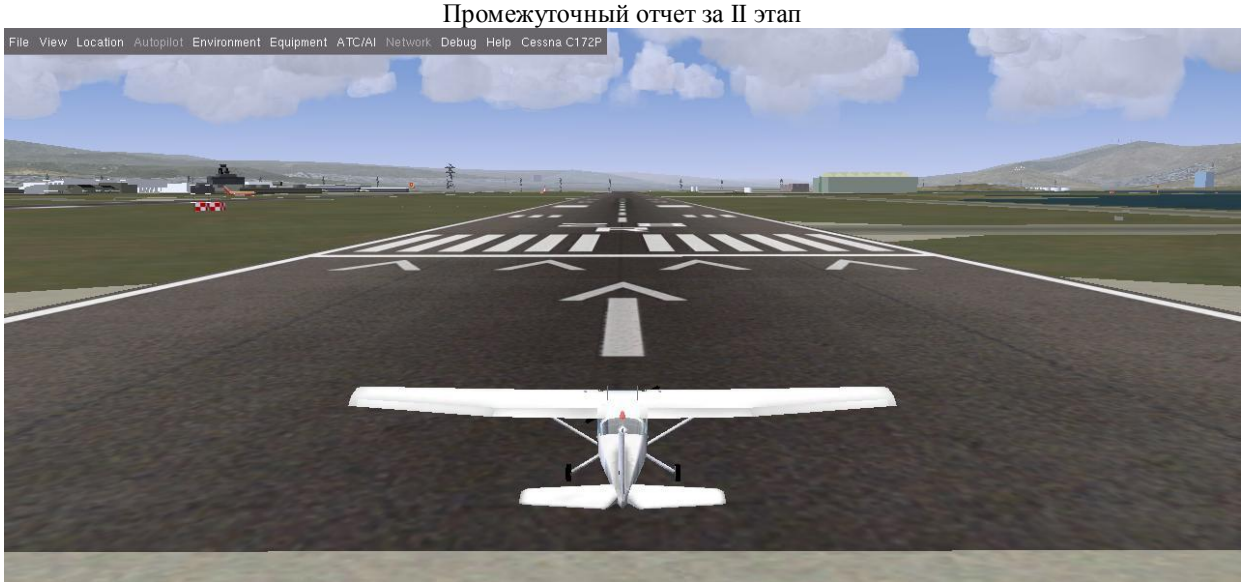


Рис. 17. Снимок экрана авиасимулятора *FlightGear*

Также этот симулятор позволяет осуществлять сохранение параметров полета (скорость, направление полета и т.д.) и параметров состояния самолета (положение руля, элеронов, состояние стартера и т.п.). Параметры полета будем далее называть входными (для системы управления) параметрами, а параметры состояния – управляющими (за счет их изменения осуществляется управление самолетом).

Таким образом, схема взаимодействия системы моделирования с системой управления выглядит так, как показано на рис. 18.

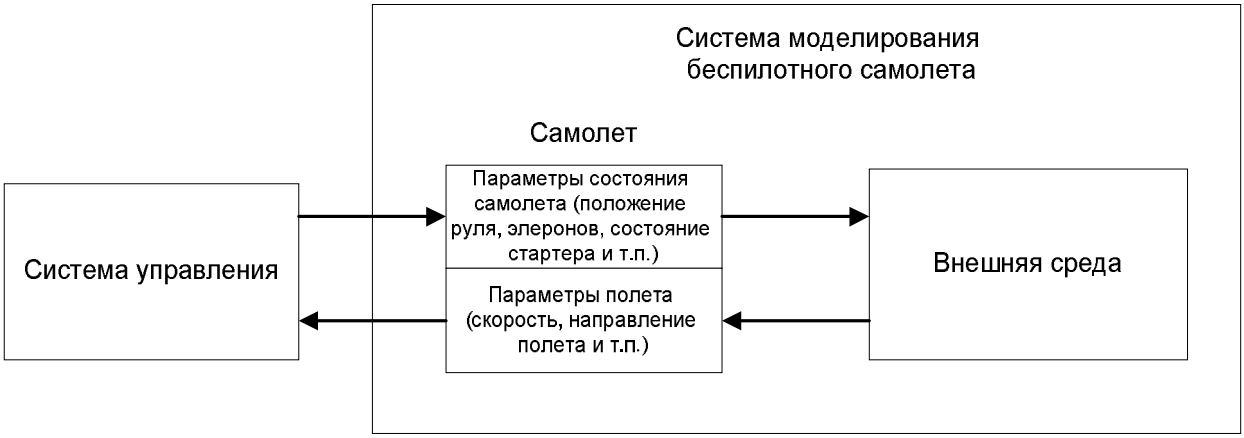


Рис. 18. Схема взаимодействия системы моделирования беспилотного самолета с системой управления

Отметим, что некоторые параметры могут принимать лишь конечное множество значений (такие параметры будем называть дискретными), а другие параметры характеризуются вещественными значениями (их будем называть непрерывными). Будем также называть непрерывными действия над непрерывными управляющими параметрами и дискретными – над дискретными.

При этом непрерывные действия изменяют соответственные параметры на некоторую вещественную величину, а дискретные – устанавливают соответствующий параметр в конкретное значение. Заметим, что последовательное выполнение действий с одним параметром эквивалентно сумме действий в случае непрерывного параметра и последнему действию – в случае дискретного.

Например, одним из непрерывных управляющих параметров является угол поворота руля. Непрерывным действием над ним является изменение угла поворота руля на некоторое значение. Тогда последовательность поворотов руля на x и y градусов эквивалентна повороту руля на $(x+y)$ градусов.

В свою очередь, хорошим примером дискретного управляющего параметра является состояние стартера. Дискретным действием над ним является его включение или выключение. Тогда последовательность включений и выключений стартера эквивалентна последнему совершенному над ним действию.

2.2. Экспериментальное исследование на задаче построения автомата, управляющего разгоном самолета

Задача генерации автомата для режима «разгон самолета» на самом деле не является простой, как может показаться на первый взгляд. Автомат, который «держит штурвал прямо», поставленную задачу не решает, так как самолет при этой стратегии разгоняется не ровно – его сносит влево, и он довольно быстро съезжает с взлетной полосы из-за параметров внешней среды. Такое поведение можно наблюдать на рис. 19–23.

Протоколы экспериментов по построению автоматов, управляющих разгоном самолета, приведены в Приложении 1. Было проведено 10 экспериментов (запусков генетического алгоритма) по построению автомата, который управляет разгоном самолета. В каждом из запусков генетический алгоритм использовал один и тот же набор обучающих примеров, состоящий из 15 тестов. Каждый из запусков работал примерно по 5 часов. Десять полученных управляющих автоматов, которые были лучшими в своих поколениях, были проверены путем запуска на самолете в авиасимуляторе и просмотра их поведения. Запуск каждого автомата производился по 3-6 раз. По результатам этих запусков каждому из них авторами была поставлена оценка по десятибалльной шкале. Три из них получили девять из десяти и были практически идеальными при разгоне самолета, иногда выполняя разгон даже лучше, чем это умеют делать авторы. Только три из автоматов получили оценку три и менее и правильно выполняли разгон менее чем в половине запусков. Полученные результаты работы генетического алгоритма авторы признают успешными.

Рис. 19. Поведение автомата, который «держит штурвал прямо» (4-я секунда разгона)

Рис. 20. Поведение автомата, который «держит штурвал прямо» (6-я секунда разгона)

Рис. 21. Поведение автомата, который «держит штурвал прямо» (8-я секунда разгона)

Рис. 22. Поведение автомата, который «держит штурвал прямо» (11-я секунда разгона)

Рис. 23. Поведение автомата, который «держит штурвал прямо» (14-я секунда разгона)

2.2.1. Проделанные для решения задачи шаги

Для решения поставленной задачи были сделаны следующие шаги:

- найден необходимый и достаточный набор утверждений о состоянии самолета;
- записано пять наборов тестов (по 10 обучающих примеров в каждом), которые рассматривались независимо друг от друга;
- запущен генетический алгоритм для каждого набора параметров и каждого набора тестов. Параметры выбирались так:
  - размер поколения – 100 особей;
  - число состояний автомата варьировалось от двух до 10;
  - вероятность мутации менялась от 0,02 до 0,5;
  - в качестве методов отбора использовались турнирный метод и метод рулетки;
  - размер элиты составлял от нуля до двух особей.

В среднем время работы генетического алгоритма составляло несколько дней для одного набора параметров и одного набора тестов.

Были выбраны следующие утверждения о состоянии самолета:

1. Начало такта (истинно всегда).
2. Отклонение от курса невелико (меньше заранее выбранного значения, в данной работе – один градус).
3. Отклонение от курса вправо.
4. Скорость изменения направления движения больше нуля (изменяем направление вправо).
5. Ускорение изменения направления движения больше нуля.

2.2.2. Записанные обучающие примеры

На рис. рис. 24–32 приведены кадры видеозаписи одного из обучающих примеров.

Рис. 24. Кадр видеозаписи одного из обучающих примеров (4-я секунда)

Рис. 25. Кадр видеозаписи одного из обучающих примеров (15-я секунда)

Рис. 26. Кадр видеозаписи одного из обучающих примеров (18-я секунда)

Рис. 27. Кадр видеозаписи одного из обучающих примеров (19-я секунда)

Рис. 28. Кадр видеозаписи одного из обучающих примеров (20-я секунда)

Рис. 29. Кадр видеозаписи одного из обучающих примеров (23-я секунда)

Рис. 30. Кадр видеозаписи одного из обучающих примеров (25-я секунда)

Рис. 31. Кадр видеозаписи одного из обучающих примеров (28-я секунда)

Рис. 32. Кадр видеозаписи одного из обучающих примеров (32-я секунда)

Также видеозапись одного из обучающих примеров доступна в сети Интернет по адресу http://www.youtube.com/watch?v=BfG90yFiGUo.

2.2.3. Полученные результаты

В результате работы генетического алгоритма было получено порядка 100 автоматов, из которых был выбран лучший путем запуска каждого автомата на самолете в авиасимуляторе и просмотра поведения самолета. Лучший автомат имеет четыре состояния. Управляющий автомат выравнивается после начального смещения влево и ровно разгоняется.

График зависимости максимального значения функции приспособленности автоматов от номера поколения (для процесса выращивания лучшего автомата) приведены на рис. 33.

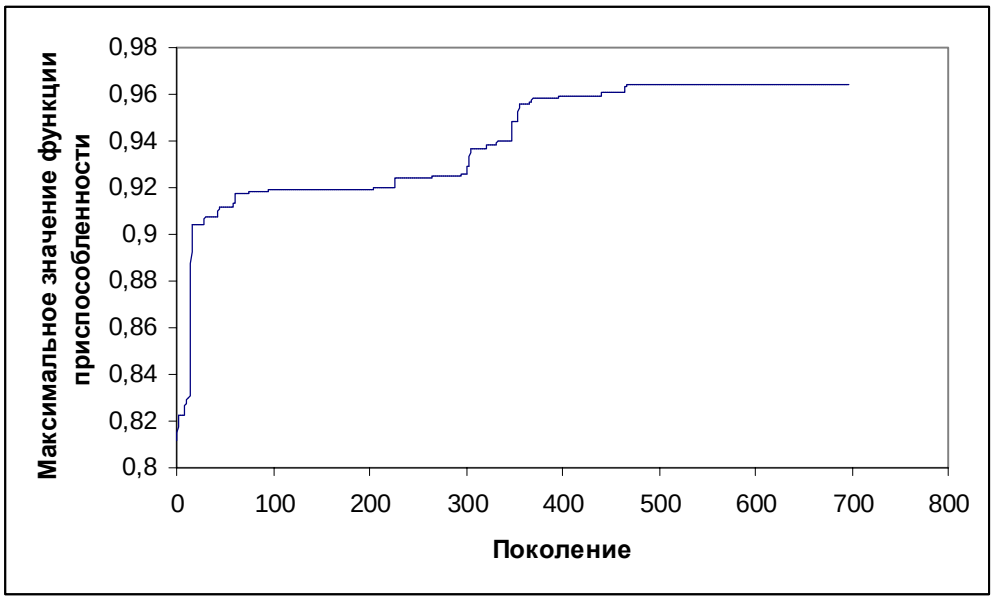


Рис. 33. График зависимости максимального значения функции приспособленности от номера поколения для задачи разгона самолета

2.2.4. Лучший из построенных автоматов

На рис. 34–40 можно увидеть кадры видеозаписи разгона лучшего автомата.

Рис. 34. Кадр видеозаписи разгона лучшего автомата (5-я секунда)

Рис. 35. Кадр видеозаписи разгона лучшего автомата (8-я секунда)

Рис. 36. Кадр видеозаписи разгона лучшего автомата (11-я секунда)

Рис. 37. Кадр видеозаписи разгона лучшего автомата (14-я секунда)

Рис. 38. Кадр видеозаписи разгона лучшего автомата (18-я секунда)

Рис. 39. Кадр видеозаписи разгона лучшего автомата (21-я секунда)

Рис. 40. Кадр видеозаписи разгона лучшего автомата (24-я секунда)

Видеозапись разгона, выполняемого лучшим автоматом, доступна по адресу http://www.youtube.com/watch?v=AiosTU7HWb4.

2.3. Экспериментальное исследование на задаче построения автомата, управляющего выполнением «мертвой петли»

Данная задача более формально может быть сформулирована следующим образом: требуется построить автомат, который делает мертвую петлю и выравнивается. Ввиду того, что понятие «выравнивается» строго не определено, в тестах было записано 2 – 3 секунды нормального выровненного полета после выполнения трюка. Поэтому простым методом «вытянуть штурвал на себя» ее не решить.

Протоколы экспериментов по построению автоматов, управляющих выполнением «мертвой петли», приведены в Приложении 2. Также было проведено 10 экспериментов по построению автомата, выполняющего трюк «мертвая петля». В каждом из запусков генетический алгоритм использовал 11 обучающих примеров и работал около 10 часов. Полученные управляющие автоматы были проверены путем запуска на самолете в авиасимуляторе и просмотра поведения самолета. Запуск каждого автомата производился по 2-5 раз. Как и ранее, каждому автомату авторами была поставлена оценка по десятибалльной шкале по результатам запусков. Три из них получили оценку восемь и отлично выполняли трюк чуть больше, чем в половине случаев. Каждый из этих трех автоматов не смог справиться с управлением в одном из запусков после частично выполненного трюка. В остальных случаях трюк выполнялся, но с замечаниями. Только один из автоматов получил оценку три и из двух раз только в одном выполнил трюк наполовину, в другом же не выполнил вовсе.

По сравнению с экспериментами первой задачи результаты получились немного хуже, что, по мнению авторов, обусловлено сложностью поставленной задачи – выполнения трюка «мертвая петля». Несмотря на это, авторы считают полученные решения для задачи выполнения трюка хорошими.

2.3.1. Проделанные для решения задачи шаги

Для решения поставленной задачи были сделаны следующие шаги:

- найден необходимый и достаточный набор утверждений о состоянии самолета;
- записано три набора тестов (по 10 обучающих примеров в каждом), которые рассматривались независимо друг от друга;
- запущен генетический алгоритм для каждого набора параметров и каждого набора тестов. Параметры выбирались так:
  - размер поколения – 100 особей;
  - число состояний автомата варьировалось от двух до пяти;
  - вероятность мутации не менялась и была равна 0,5;
  - в качестве метода отбора использовался турнирный метод;
  - размер элиты составлял две особи.

В среднем время работы генетического алгоритма, как и раньше, составляло несколько часов для одного набора параметров и одного набора тестов.

Выбранные утверждения были следующие:

1. Начало такта (истинно всегда).
2. Тангаж (угол наклона) самолета маленький (меньше одного градуса).
3. Тангаж больше нуля (самолет смотрит вниз).
4. Скорость изменения угла наклона больше нуля.
5. Ускорение изменения угла наклона больше нуля.
6. Крен (угол наклона) самолета маленький (меньше одного градуса).

7. Крен больше нуля (самолет завалился на правый бок).
8. Скорость изменения крена больше нуля.
9. Ускорение изменения крена больше нуля.

Углы наклона – тангаж и крен – можно увидеть на рис. 41.

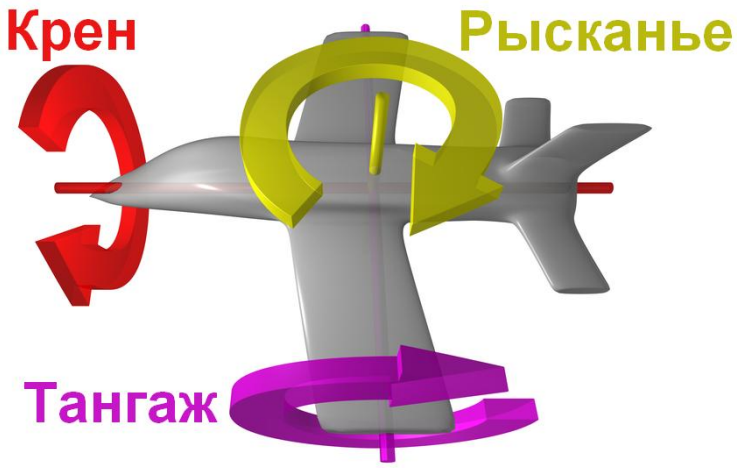


Рис. 41. Углы наклона самолета

2.3.2. Записанные обучающие примеры

На рис. 42–53 приведены кадры видеозаписи одного из обучающих примеров выполнения «мертвой петли».

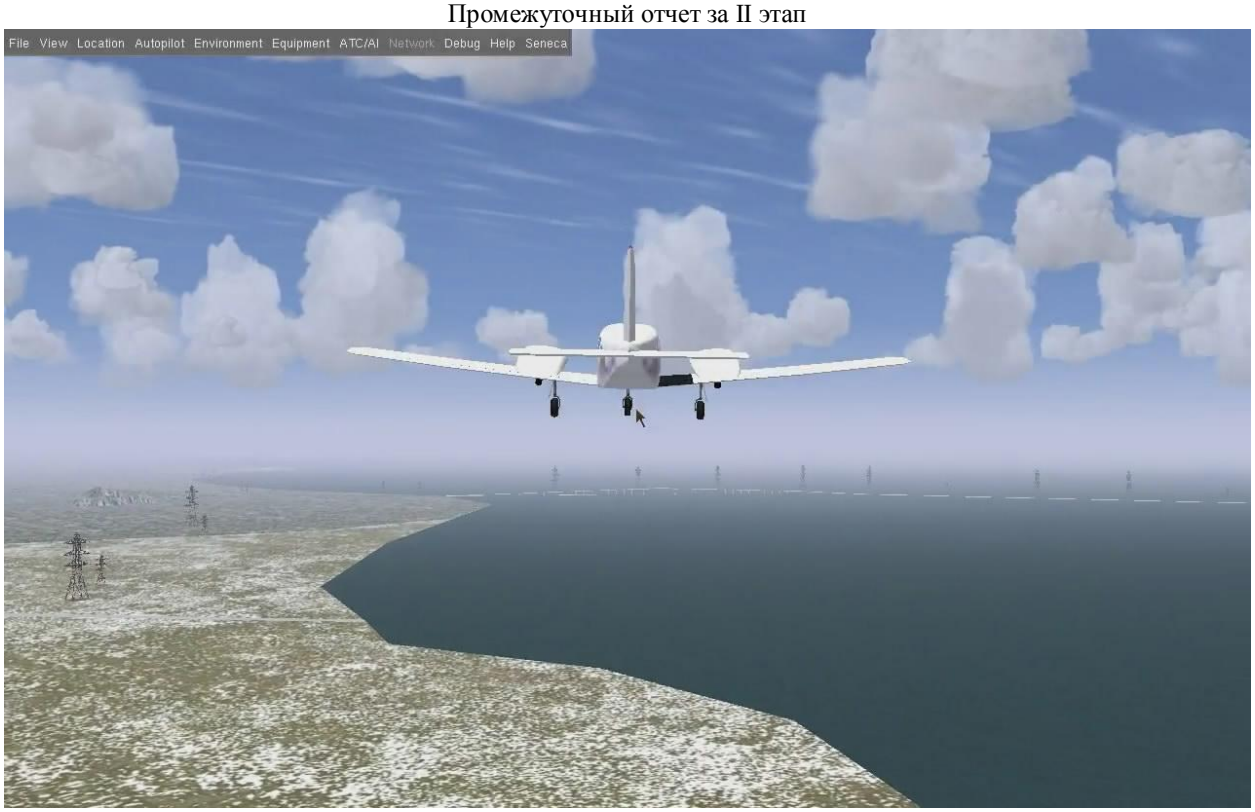


Рис. 42. Кадр видеозаписи одного из обучающих примеров выполнения «мертвой петли» (1-я секунда)

Рис. 43. Кадр видеозаписи одного из обучающих примеров выполнения «мертвой петли» (3-я секунда)

Рис. 44. Кадр видеозаписи одного из обучающих примеров выполнения «мертвой петли» (4-я секунда)

Рис. 45. Кадр видеозаписи одного из обучающих примеров выполнения «мертвой петли» (6-я секунда)

Рис. 46. Кадр видеозаписи одного из обучающих примеров выполнения «мертвой петли» (8-я секунда)

Рис. 47. Кадр видеозаписи одного из обучающих примеров выполнения «мертвой петли» (10-я секунда)

Рис. 48. Кадр видеозаписи одного из обучающих примеров выполнения «мертвой петли» (12-я секунда)

Рис. 49. Кадр видеозаписи одного из обучающих примеров выполнения «мертвой петли» (15-я секунда)

Рис. 50. Кадр видеозаписи одного из обучающих примеров выполнения «мертвой петли» (16-я секунда)

Рис. 51. Кадр видеозаписи одного из обучающих примеров выполнения «мертвой петли» (18-я секунда)

Рис. 52. Кадр видеозаписи одного из обучающих примеров выполнения «мертвой петли» (22-я секунда)

Рис. 53. Кадр видеозаписи одного из обучающих примеров выполнения «мертвой петли» (24-я секунда)

Видеозаписи некоторых успешно выполненных обучающих примеров:

- http://www.youtube.com/watch?v=G5Kcx0ohpNo;
- http://www.youtube.com/watch?v=C1a7_eDp1BI.

Видеозаписи неудачно выполненного трюка «мертвая петля», которые не были включены в набор тестов:

- http://www.youtube.com/watch?v=JQH3VNiIjV8;
- http://www.youtube.com/watch?v=OGVTch-a97A.

2.3.3. Полученные результаты

Было получено порядка 50 автоматов, из которых, как и раньше, был выбран лучший. Он тоже имеет четыре состояния. Было замечено, что полученный управляющий автомат может при благоприятных условиях выполнять несколько мертвых петель с некоторым интервалом. Такое поведение весьма логично ввиду выбранной структуры тестов. При некоторых других начальных условиях автомат «не хочет» выполнять более одного трюка и просто летит дальше. Такое поведение тоже схоже с тестами.

График зависимости максимального значения функции приспособленности автоматов от номера поколения (для процесса выращивания лучшего автомата) приведен на рис. 54.

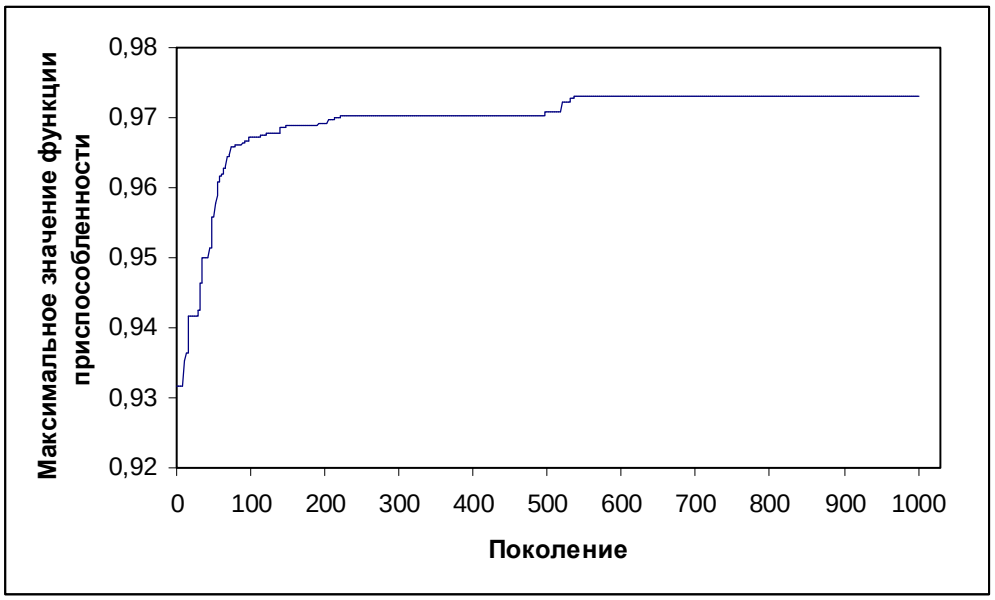


Рис. 54. График зависимости максимального значения функции приспособленности от номера поколения для задачи выполнения «мертвой петли»

2.3.4. Лучший из построенных автоматов

На рис. 55–65 можно увидеть кадры видеозаписи выполнения трюка «мертвая петля» лучшим автоматом.

Рис. 55. Кадр видеозаписи выполнения трюка «мертвая петля» лучшим автоматом (2-я секунда)

Рис. 56. Кадр видеозаписи выполнения трюка «мертвая петля» лучшим автоматом (8-я секунда)

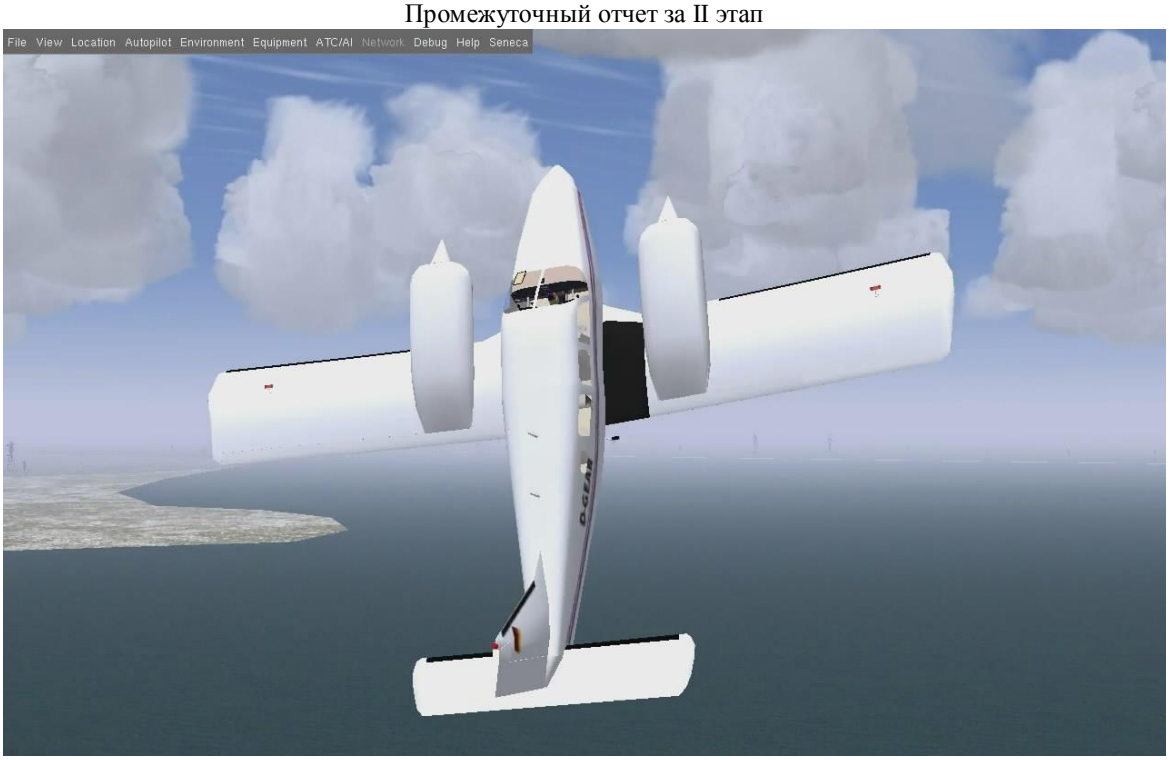


Рис. 57. Кадр видеозаписи выполнения трюка «мертвая петля» лучшим автоматом (11-я секунда)

Рис. 58. Кадр видеозаписи выполнения трюка «мертвая петля» лучшим автоматом (12-я секунда)

Рис. 59. Кадр видеозаписи выполнения трюка «мертвая петля» лучшим автоматом (13-я секунда)

Рис. 60. Кадр видеозаписи выполнения трюка «мертвая петля» лучшим автоматом (15-я секунда)

Рис. 61. Кадр видеозаписи выполнения трюка «мертвая петля» лучшим автоматом (17-я секунда)

Рис. 62. Кадр видеозаписи выполнения трюка «мертвая петля» лучшим автоматом (20-я секунда)

Рис. 63. Кадр видеозаписи выполнения трюка «мертвая петля» лучшим автоматом (22-я секунда)

Рис. 64. Кадр видеозаписи выполнения трюка «мертвая петля» лучшим автоматом (25-я секунда)

Рис. 65. Кадр видеозаписи выполнения трюка «мертвая петля» лучшим автоматом (27-я секунда)

Видеозаписи выполнения мертвой петли лучшим автоматом:

- http://www.youtube.com/watch?v=C6WV7x2bqE8;
- http://www.youtube.com/watch?v=TzrLoJjjVTA;
- http://www.youtube.com/watch?v=yFiG4yz67Ks.

3. ПУБЛИКАЦИИ РЕЗУЛЬТАТОВ НИР

По результатам НИР опубликованы две статьи:

- 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.
- Царев Ф. Н. Метод построения управляющих конечных автоматов на основе тестовых примеров с помощью генетического программирования // Информационно-управляющие системы. 2010. № 5 (48), с. 31–36.

Копии статей приведены в приложении 3. Копия экспертного заключения о возможности опубликования приведена в приложении 4.

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

- Александров А. В., Казаков С. В., Сергушичев А. А., Царев Ф. Н. Гнетическое программирование на основе обучающих примеров для построения конечных автоматов управления моделью беспилотного самолета /Сборник докладов международной конференции по мягким вычислениям и измерениям (SCM`2010). СПбГЭТУ «ЛЭТИ». Т. 1, с. 263–267.
- 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.
- Александров А. В., Казаков С. В., Парфенов В. Г., Сергушичев А. А., Царев Ф. Н. Применение генетических алгоритмов на основе обучающих примеров для построения конечных автоматов для управления моделью беспилотного самолета /Труды XVII Всероссийской научно-методической конференции «Телематика`2010». Т. 2. СПбГУ ИТМО. 2010, с. 343.

ЗАКЛЮЧЕНИЕ

В результате исследований, проведенных на втором этапе работ по контракту, были выполнены следующие работы:

- разработка и программная реализация метода построения управляющих конечных автоматов на основе обучающих примеров с помощью генетического программирования:
  - разработка метода представления управляющих конечных автоматов в виде хромосом алгоритма генетического программирования;
  - разработка алгоритмов реализации операций мутации и скрещивания;
  - разработка алгоритма расстановки пометок на переходах автомата на основе обучающих примеров;
- разработка и программная реализация метода двухэтапного генетического программирования:
  - разработка методов генетического программирования, осуществляющих построение репозитория состояний;
  - разработка методов генетического программирования, осуществляющих построение автоматов на основе репозитория состояний.
- разработка и программная реализация метода применения верификации моделей при вычислении функции приспособленности;
- разработка программной модели беспилотного летательного аппарата и экспериментальное исследование разработанных методов на задаче построения системы управления моделью беспилотного летательного аппарата.

Кроме этого, по результатам НИР опубликованы две статьи в журналах «Известия РАН. Теория и системы управления» и «Информационно-управляющие системы» (входят в перечень ВАК), а также сделаны три доклада на трех конференциях.

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

ИСТОЧНИКИ

1. *Angeline P., Pollack J.* Evolutionary Module Acquisition / Proceedings of the Second Annual Conference on Evolutionary Programming. Cambridge: MIT Press. 1993, pp.154–163. http://www.demo.cs.brandeis.edu/papers/ep93.pdf
2. *Chambers L.* Practical Handbook of Genetic Algorithms. Complex Coding Systems. Volume III. CRC Press, 1999.
3. *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.
4. *Eisenstein J.* Evolving robot tank fighters Technical Report AIM-2003-023, AI Lab, MIT. 2003.
5. *Gade M., Knudsen M., and et al.* Applying machine learning to robocode. 2003.
6. *Gastin P., Oddoux D.* Fast LTL to Büchi Automata Translation / 13th Conference on Computer Aided Verification (CAV'01). 2001, pp. 53–65.
7. *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.
8. *Hoffman L.* Talking Model-Checking Technology // Communications of the ACM. 2008. V. 51. № 7, pp. 110–112.
9. *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
10. *Jefferson D., Collins R., Cooper C., Dyer M., Flowers M., Korf R., Taylor C., Wang A.* The Genesys System: Evolution as a Theme in Artificial Life /Proceedings of Second Conference on Artificial Life. MA: Addison-Wesley. 1992, pp.549–578. www.cs.ucla.edu/~dyer/Papers/AlifeTracker/Alife91Jefferson.html
11. *Koza J. R.* Genetic programming: on the programming of computers by means of natural selection. MIT Press, 1992.
12. LTL2BA project. http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/
13. *Бедный Ю.* Применение генетических алгоритмов для генерации автоматов при построении модели максимального правдоподобия и в задачах управления. http://is.ifmo.ru/papers/bednij/masters.pdf
14. *Бедный Ю. Д., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов в задаче «Умный муравей». http://is.ifmo.ru/works/ant
15. *Данилов В.* Технология генетического программирования для генерации автоматов управления системами со сложным поведением. http://is.ifmo.ru/download/danilov_bachelor.pdf
16. *Гладков Л. А., Курейчик В. В., Курейчик В. М.* Генетические алгоритмы. М.: Физматлит, 2006.
17. *Гуров В. С., Мазим М. А., Нарвский А. С., Шалыто А. А.* UML. SWITCH-технология. Eclipse // Информационно-управляющие системы. 2004. № 6. с. 12–17.
18. *Егоров К. В., Шалыто А. А.* Методика верификации автоматных программ // Информационно-управляющие системы. СПб: Политехника, 2008, № 5, с. 15–21.
19. *Кларк Э., Грамберг О., Пелед Д.* Верификация моделей программ: Model Checking. М.: МЦНМО, 2002. 416 с.
20. *Левенштейн В. И.* Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академии Наук СССР 163.4, с. 845–848.
21. 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.

22. *Паращенко Д. А., Царев Ф. Н., Шалыто А. А.* Технология моделирования одного класса мультиагентных систем на основе автоматного программирования на примере игры «Соревнование летающих тарелок». Проектная документация. СПбГУ ИТМО. 2006. http://is.ifmo.ru/unimod-projects/plates/
23. *Поликарпова Н. И., Точилин В. Н., Шалыто А. А.* Применение генетического программирования для генерации автоматов с большим числом входных переменных // Научно-технический вестник СПбГУ ИТМО. Выпуск 53. Автоматное программирование, с. 24–42.
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. *Рассел С., Норвиг П.* Искусственный интеллект: современный подход. М.: Вильямс, 2006.
26. *Царев Ф. Н., Шалыто А. А.* Применение генетического программирования для генерации автомата в задаче об «Умном муравье» / Сборник трудов IV-ой Международной научно-практической конференции «Интегрированные модели и мягкие вычисления в искусственном интеллекте». Том 2. М.: Физматлит. 2007, с. 590–597. http://is.ifmo.ru/genalg/_ant_ga.pdf
27. *Царев Ф. Н.* Построение автоматов управления системами со сложным поведением на основе тестов с помощью генетического программирования / Сборник докладов XII Международной конференции по мягким вычислениям и измерениям (SCM'2009). Том 1. СПбГЭТУ «ЛЭТИ». 2009, с. 231–234.
28. *Царев Ф. Н., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов с минимальным числом состояний для задачи об «Умном муравье». / Материал опубликован в тезисах научно-технической конференции «Научно-программное обеспечение в образовании и научных исследованиях». СПбГУ ПУ. 2008, с. 209–215.
29. *Шалыто А. А.* Switch-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
30. *Яминов Б. Р.* Генетические алгоритмы. http://rain.ifmo.ru/cat/view.php/theory/unsorted/genetic-2005/

ПРИЛОЖЕНИЕ 1. ПРОТОКОЛЫ ЭКСПЕРИМЕНТОВ ПО ПОСТРОЕНИЮ АВТОМАТА, УПРАВЛЯЮЩЕГО РАЗГОНОМ САМОЛЕТА

Протокол эксперимента №1

Задача: построить управляющий автомат, который управляет разгоном самолета.
Используется 15 обучающих примеров.
Параметры генетического алгоритма:

- размер поколения – 100 особей;
- число состояний автомата – 4;
- вероятность мутации – 0,5;
- в качестве метода отбора используется турнирный метод;
- размер элиты составляет две особи.

Номер поколения	Максимальное значение функции приспособленности	Время от начала эксперимента (в секундах)
1	0.8882	0-5
2-4	0.8890	5-28
5-7	0.8904	28-49
8	0.8906	49-55
9-10	0.8970	55-68
11-18	0.9022	68-120
19-22	0.9025	120-146
23-27	0.9034	146-178
28-34	0.9055	178-225
35	0.9064	225-233
36-37	0.9077	233-248
38	0.9086	248-255
39	0.9088	255-262
40-42	0.9118	262-286
43-59	0.9142	286-429
60-61	0.9143	429-443
62-64	0.9146	443-469
65-76	0.9147	469-576
77-95	0.9148	576-768
96-195	0.9149	768-2016
196-296	0.9150	2016-3542
297-483	0.9158	3542-5852

Результат: полученный автомат был проверен путем запуска на самолете в авиасимуляторе и просмотра поведения самолета. Запуск производился шесть раз. В двух запусках самолет не стал разгоняться: автомат не включил стартер самолета в первом случае и не поставил дроссель на максимум во втором. В остальных запусках самолет разогнался, но сделал это не ровно, и даже в одном из запусков съехал с взлетной полосы.

Оценка авторов этому автомату (по десятибалльной шкале) – 2.

Протокол эксперимента №2

Задача: построить управляющий автомат, который управляет разгоном самолета.

Используется 15 обучающих примеров.

Параметры генетического алгоритма:

- размер поколения – 100 особей;
- число состояний автомата – 4;
- вероятность мутации – 0,5;
- в качестве метода отбора используется турнирный метод;
- размер элиты составляет две особи.

Номер поколения	Максимальное значение функции приспособленности	Время от начала эксперимента (в секундах)
1	0.8923	0-5
2-5	0.8930	5-28
6-7	0.8948	28-39
8-12	0.8970	39-72
13-14	0.9018	72-85
15-17	0.9046	85-101
18	0.9052	101-106
19-26	0.9061	106-148
27-31	0.9090	148-176
32-55	0.9115	176-345
56	0.9139	345-353
57-59	0.9140	353-378
60-66	0.9164	378-437
67-79	0.9165	437-548
80-240	0.9166	548-2403
241-789	0.9167	2403-10425

Результат: полученный автомат был проверен путем запуска на самолете в авиасимуляторе и просмотра поведения самолета. Запуск производился пять раз. В каждом из запусков самолет под управлением автомата отлично разогнался и не делал значительных отклонений от «идеальной» траектории.

Оценка авторов этому автомату (по десятибалльной шкале) – 9.

Протокол эксперимента №3

Задача: построить управляющий автомат, который управляет разгоном самолета.

Используется 15 обучающих примеров.

Параметры генетического алгоритма:

- размер поколения – 100 особей;
- число состояний автомата – 4;
- вероятность мутации – 0,5;
- в качестве метода отбора используется турнирный метод;
- размер элиты составляет две особи.

Номер поколения	Максимальное значение функции приспособленности	Время от начала эксперимента (в секундах)
1-3	0.8911	0-20
4-15	0.8966	20-91
16-19	0.9055	91-114
20-32	0.9057	114-190
33-50	0.9081	190-317
51-55	0.9098	317-358
56-72	0.9107	358-504
73-78	0.9110	504-550
79	0.9111	550-557
80-83	0.9112	557-587
84-91	0.9116	587-649
92-106	0.9117	649-784
107-108	0.9149	784-802
109-149	0.9166	802-1229
150-368	0.9167	1229-4236
369-477	0.9168	4236-5861

Результат: полученный автомат был проверен путем запуска на самолете в авиасимуляторе и просмотра поведения самолета. Запуск производился три раза. В одном из запусков автомат не включил стартер самолета и поэтому самолет не стал разгоняться, в двух других самолет хорошо разогнался, но в одном из запусков самолет отклонился от «идеальной» траектории на небольшую, но заметную величину.

Оценка авторов этому автомату (по десятибалльной шкале) – 5.

Протокол эксперимента №4

Задача: построить управляющий автомат, который управляет разгоном самолета.

Используется 15 обучающих примеров.

Параметры генетического алгоритма:

- размер поколения – 100 особей;
- число состояний автомата – 4;
- вероятность мутации – 0,5;
- в качестве метода отбора используется турнирный метод;
- размер элиты составляет две особи.

Номер поколения	Максимальное значение функции приспособленности	Время от начала эксперимента (в секундах)
1-2	0.8882	0-12
3	0.8927	12-18
4	0.8987	18-23
5-17	0.8994	23-103
18-25	0.8997	103-152
26-32	0.9020	152-202
33-34	0.9042	202-217
35-37	0.9048	217-237
38	0.9078	237-244
39-43	0.9093	244-273
44	0.9125	273-279
45-51	0.9126	279-321
52-56	0.9151	321-354
57-72	0.9156	354-473
73-82	0.9157	473-571
83-234	0.9158	571-2578
235-774	0.9175	2578-10396

Результат: полученный автомат был проверен путем запуска на самолете в авиасимуляторе и просмотра поведения самолета. Запуск производился три раза. В двух случаях из трех автомат не включил стартер самолета и поэтому самолет не стал разгоняться, в другом запуске самолет отлично выполнил разгон.

Оценка авторов этому автомату (по десятибалльной шкале) – 3.

Протокол эксперимента №5

Задача: построить управляющий автомат, который управляет разгоном самолета.

Используется 15 обучающих примеров.

Параметры генетического алгоритма:

- размер поколения – 100 особей;
- число состояний автомата – 4;
- вероятность мутации – 0,5;
- в качестве метода отбора используется турнирный метод;
- размер элиты составляет две особи.

Номер поколения	Максимальное значение функции приспособленности	Время от начала эксперимента (в секундах)
1-9	0.8974	0-43
10-11	0.9018	43-53
12-35	0.9039	53-156
36-47	0.9068	156-206
48	0.9076	206-210
49-50	0.9088	210-218
51-54	0.9108	218-234
55	0.9115	234-238
56-63	0.9130	238-273
64	0.9148	273-277
65-87	0.9151	277-399
88-91	0.9152	399-423
92-93	0.9153	423-435
94-115	0.9155	435-583
116-134	0.9157	583-728
135-221	0.9158	728-1493
222-1253	0.9159	1493-11986

Результат: полученный автомат был проверен путем запуска на самолете в авиасимуляторе и просмотра поведения самолета. Запуск производился шесть раз. В каждом из запусков самолет под управлением автомата хорошо разогнался, но в двух случаях было замечено небольшое отклонение от «идеальной» траектории.

Оценка авторов этому автомату (по десятибалльной шкале) – 9.

Протокол эксперимента №6

Задача: построить управляющий автомат, который управляет разгоном самолета.

Используется 15 обучающих примеров.

Параметры генетического алгоритма:

- размер поколения – 100 особей;
- число состояний автомата – 4;
- вероятность мутации – 0,5;
- в качестве метода отбора используется турнирный метод;
- размер элиты составляет две особи.

Номер поколения	Максимальное значение функции приспособленности	Время от начала эксперимента (в секундах)
1	0.8874	0-5
2-14	0.9013	5-71
15	0.9035	71-76
16-18	0.9049	76-90
19-20	0.9050	90-100
21-22	0.9064	100-110
23-25	0.9065	110-125
26-27	0.9088	125-134
28-36	0.9092	134-171
37-39	0.9135	171-182
40-52	0.9140	182-242
53-57	0.9141	242-271
58-67	0.9159	271-337
68-105	0.9163	337-665
106-116	0.9165	665-768
117-127	0.9166	768-873
128-1212	0.9167	873-12030

Результат: полученный автомат был проверен путем запуска на самолете в авиасимуляторе и просмотра поведения самолета. Запуск производился шесть раз. В двух случаях самолет при выполнении разгона выехал за пределы взлетной полосы, в остальных случаях самолет хорошо выполнил разгон.

Оценка авторов этому автомату (по десятибалльной шкале) – 6.

Протокол эксперимента №7

Задача: построить управляющий автомат, который управляет разгоном самолета.

Используется 15 обучающих примеров.

Параметры генетического алгоритма:

- размер поколения – 100 особей;
- число состояний автомата – 4;
- вероятность мутации – 0,5;
- в качестве метода отбора используется турнирный метод;
- размер элиты составляет две особи.

Номер поколения	Максимальное значение функции приспособленности	Время от начала эксперимента (в секундах)
1	0.8859	0-13
2	0.8869	13-27
3-4	0.8921	27-54
5-6	0.8927	54-82
7	0.8930	82-95
8-10	0.9014	95-134
11-12	0.9041	134-160
13-28	0.9044	160-326
29-32	0.9067	326-361
33-35	0.9090	361-389
36-37	0.9102	389-408
38-42	0.9106	408-455
43-55	0.9118	455-594
56-77	0.9142	594-890
78-121	0.9151	890-1635
122-139	0.9155	1635-1969
140-143	0.9156	1969-2045
144-163	0.9157	2045-2461
164-1007	0.9158	2461-25440

Результат: полученный автомат был проверен путем запуска на самолете в авиасимуляторе и просмотра поведения самолета. Запуск производился шесть раз. В каждом из запусков самолет хорошо разогнался, но в двух случаях значительно отклонился от «эталонной» траектории.

Оценка авторов этому автомату (по десятибалльной шкале) – 7.

Протокол эксперимента №8

Задача: построить управляющий автомат, который управляет разгоном самолета.

Используется 15 обучающих примеров.

Параметры генетического алгоритма:

- размер поколения – 100 особей;
- число состояний автомата – 4;
- вероятность мутации – 0,5;
- в качестве метода отбора используется турнирный метод;
- размер элиты составляет две особи.

Номер поколения	Максимальное значение функции приспособленности	Время от начала эксперимента (в секундах)
1	0.8852	0-5
2-3	0.8912	5-15
4-5	0.8949	15-25
6-8	0.8991	25-41
9-13	0.9041	41-65
14-22	0.9059	65-108
23-38	0.9084	108-178
39	0.9106	178-182
40-42	0.9107	182-195
43-44	0.9109	195-204
45-50	0.9130	204-231
51-57	0.9133	231-267
58-75	0.9142	267-368
76-125	0.9151	368-644
126-138	0.9154	644-724
139-157	0.9155	724-856
158-178	0.9157	856-1015
179-309	0.9158	1015-2039
310-1339	0.9159	2039-13223

Результат: полученный автомат был проверен путем запуска на самолете в авиасимуляторе и просмотра поведения самолета. Запуск производился шесть раз. В каждом из запусков самолет под управлением автомата отлично выполнил разгон и не сделал значительных отклонений от «идеальной» траектории.

Оценка авторов этому автомату (по десятибалльной шкале) – 9.

Протокол эксперимента №9

Задача: построить управляющий автомат, который управляет разгоном самолета.

Используется 15 обучающих примеров.

Параметры генетического алгоритма:

- размер поколения – 100 особей;
- число состояний автомата – 4;
- вероятность мутации – 0,5;
- в качестве метода отбора используется турнирный метод;
- размер элиты составляет две особи.

Номер поколения	Максимальное значение функции приспособленности	Время от начала эксперимента (в секундах)
1-2	0.8873	0-7
3	0.8877	7-10
4-8	0.8994	10-27
9-11	0.8999	27-36
12-27	0.9041	36-79
28-37	0.9060	79-103
38-43	0.9062	103-118
44-46	0.9084	118-126
47-82	0.9108	126-242
83-85	0.9120	242-252
86-91	0.9132	252-274
92-95	0.9140	274-288
96-109	0.9156	288-340
110-111	0.9157	340-347
112-115	0.9159	347-363
116-203	0.9160	363-758
204-241	0.9185	758-946
242-2295	0.9211	946-10897

Результат: полученный автомат был проверен путем запуска на самолете в авиасимуляторе и просмотра поведения самолета. Запуск производился шесть раз. В двух случаях автомат не включил самолету стартер и поэтому самолет не стал разгоняться, в трех случаях самолет сильно отклонился вправо и в двух случаях из этих трех в конце разгона выехал за пределы взлетной полосы, оставшийся один раз самолет разогнался хорошо.

Оценка авторов этому автомату (по десятибалльной шкале) – 3.

Протокол эксперимента №10

Задача: построить управляющий автомат, который управляет разгоном самолета.
Используется 15 обучающих примеров.
Параметры генетического алгоритма:

- размер поколения – 100 особей;
- число состояний автомата – 4;
- вероятность мутации – 0,5;
- в качестве метода отбора используется турнирный метод;
- размер элиты составляет две особи.

Номер поколения	Максимальное значение функции приспособленности	Время от начала эксперимента (в секундах)
1	0.8859	0-3
2	0.8912	3-6
3-6	0.8918	6-21
7-15	0.8964	21-49
16-20	0.9030	49-63
21-24	0.9033	63-73
25-26	0.9038	73-78
27-43	0.9080	78-112
44-45	0.9095	112-115
46-47	0.9104	115-119
48-50	0.9108	119-125
51	0.9110	125-127
52	0.9113	127-130
53	0.9115	130-132
54-57	0.9132	132-143
58-66	0.9135	143-168
67-68	0.9139	168-175
69	0.9143	175-178
70-73	0.9146	178-191
74	0.9147	191-195
75-88	0.9164	195-251
89-105	0.9165	251-321
106-131	0.9166	321-450
132-514	0.9167	450-3077
515-884	0.9174	3077-5563
885-1088	0.9175	5563-6717
1089-1883	0.9176	6717-11993

Результат: полученный автомат был проверен путем запуска на самолете в авиасимуляторе и просмотра поведения самолета. Запуск производился шесть раз. В каждом из запусков самолет выполнил разгон, но в двух случаях отклонился вправо от «идеальной» траектории.

Оценка авторов этому автомату (по десятибалльной шкале) – 7.

ПРИЛОЖЕНИЕ 2. ПРОТОКОЛЫ ЭКСПЕРИМЕНТОВ ПО ПОСТРОЕНИЮ АВТОМАТА, УПРАВЛЯЮЩЕГО ВЫПОЛНЕНИЕМ «МЕРТВОЙ ПЕТЛИ»

Протокол эксперимента №1

Задача: построить управляющий автомат, который выполняет трюк «мертвая петля».
Используется 11 обучающих примеров.
Параметры генетического алгоритма:

- размер поколения – 100 особей;
- число состояний автомата – 4;
- вероятность мутации – 0,5;
- в качестве метода отбора используется турнирный метод;
- размер элиты составляет две особи.

Номер поколения	Максимальное значение функции приспособленности	Время от начала эксперимента (в секундах)
1	0.9463	0-10
2-3	0.9471	10-31
4-6	0.9477	31-65
7-10	0.9501	65-113
11-13	0.9509	113-150
14-25	0.9542	150-309
26-32	0.9545	309-415
33-37	0.9557	415-491
38-39	0.9561	491-523
40-41	0.9565	523-553
42-48	0.9572	553-673
49-51	0.9573	673-724
52	0.9599	724-741
53-56	0.9605	741-812
57	0.9609	812-828
58-65	0.9621	828-978
66-67	0.9625	978-1016
68	0.9628	1016-1036
69-71	0.9630	1036-1095
72	0.9632	1095-1114
73-75	0.9635	1114-1170
76-80	0.9639	1170-1263
81-82	0.9643	1263-1301
83-88	0.9648	1301-1416
89-90	0.9649	1416-1455
91-94	0.9650	1455-1528
95-103	0.9652	1528-1696
104-107	0.9653	1696-1777
108-111	0.9654	1777-1856
112-113	0.9655	1856-1897
114-116	0.9656	1897-1959

117-124	0.9659	1959-2135
125-140	0.9660	2135-2503
141-147	0.9661	2503-2661
148-158	0.9662	2661-2902
159-165	0.9664	2902-3063
166-167	0.9665	3063-3111
168-876	0.9666	3111-19488
877-886	0.9667	19488-19723
887-904	0.9670	19723-20137
905-958	0.9673	20137-21376
959-979	0.9674	21376-21881
980-1000	0.9676	21881-22388

Результат: полученный автомат был проверен путем запуска на самолете в авиасимуляторе и просмотра поведения самолета. Запуск производился два раза. В одном из запусков автомат начал выполнять трюк, но в заключительной стадии не смог справиться с управлением, в другом – выполнил трюк и выровнялся, но при выполнении трюка значительно «завалился» набок.

Оценка авторов этому автомату (по десятибалльной шкале) – 5.

Протокол эксперимента №2

Задача: построить управляющий автомат, который выполняет трюк «мертвая петля».
Используется 11 обучающих примеров.
Параметры генетического алгоритма:

- размер поколения – 100 особей;
- число состояний автомата – 4;
- вероятность мутации – 0,5;
- в качестве метода отбора используется турнирный метод;
- размер элиты составляет две особи.

Номер поколения	Максимальное значение функции приспособленности	Время от начала эксперимента (в секундах)
1	0.9474	0-9
2-8	0.9495	9-85
9	0.9500	85-98
10-28	0.9540	98-370
29-32	0.9542	370-437
33-38	0.9544	437-536
39-43	0.9562	536-623
44-47	0.9566	623-699
48	0.9570	699-718
49-51	0.9578	718-773
52-54	0.9610	773-831
55-64	0.9622	831-1038
65	0.9626	1038-1059
66-68	0.9636	1059-1120
69-71	0.9644	1120-1182
72	0.9646	1182-1203
73-77	0.9652	1203-1304
78-80	0.9653	1304-1368
81-84	0.9654	1368-1448
85	0.9660	1448-1469
86-92	0.9663	1469-1613
93-94	0.9664	1613-1653
95-97	0.9673	1653-1716
98-103	0.9674	1716-1836
104-105	0.9677	1836-1878
106-110	0.9679	1878-1986
111-112	0.9681	1986-2030
113	0.9684	2030-2052
114-115	0.9686	2052-2098
116-124	0.9688	2098-2305
125-129	0.9689	2305-2420
130-132	0.9690	2420-2489
133-134	0.9697	2489-2535
135-150	0.9698	2535-2894
151-152	0.9699	2894-2938

153-154	0.9700	2938-2983
155-170	0.9701	2983-3370
171-197	0.9702	3370-4044
198-223	0.9703	4044-4654
224-231	0.9704	4654-4839
232-236	0.9706	4839-4956
237	0.9707	4956-4978
238-259	0.9708	4978-5471
260-1000	0.9709	5471-23882

Результат: полученный автомат был проверен путем запуска на самолете в авиасимуляторе и просмотра поведения самолета. Запуск производился два раза. В каждом из запусков автомат выполнил трюк, однако при первом запуске он не смог выровняться после выполнения трюка, а при втором – не справился с управлением в завершающей стадии.

Оценка авторов этому автомату (по десятибалльной шкале) – 7.

Протокол эксперимента №3

Задача: построить управляющий автомат, который выполняет трюк «мертвая петля».
Используется 11 обучающих примеров.
Параметры генетического алгоритма:

- размер поколения – 100 особей;
- число состояний автомата – 4;
- вероятность мутации – 0,5;
- в качестве метода отбора используется турнирный метод;
- размер элиты составляет две особи.

Номер поколения	Максимальное значение функции приспособленности	Время от начала эксперимента (в секундах)
1-7	0.9500	0-73
8-12	0.9511	73-134
13-18	0.9527	134-210
19	0.9530	210-224
20-26	0.9540	224-326
27-30	0.9559	326-388
31-34	0.9569	388-455
35-39	0.9577	455-543
40-42	0.9589	543-598
43-45	0.9605	598-655
46-47	0.9618	655-695
48-49	0.9619	695-736
50-51	0.9631	736-776
52-55	0.9635	776-858
56-58	0.9640	858-920
59-60	0.9647	920-964
61	0.9649	964-985
62-66	0.9650	985-1089
67-69	0.9654	1089-1156
70-73	0.9655	1156-1246
74-75	0.9656	1246-1292
76-80	0.9657	1292-1400
81-83	0.9658	1400-1466
84-92	0.9659	1466-1670
93-95	0.9662	1670-1742
96-117	0.9663	1742-2256
118-122	0.9665	2256-2375
123-145	0.9666	2375-2891
146-148	0.9667	2891-2959
149-154	0.9668	2959-3088
155-158	0.9669	3088-3180
159-161	0.9670	3180-3244
162-168	0.9671	3244-3401
169	0.9675	3401-3422
170-174	0.9676	3422-3534

175-187	0.9679	3534-3824
188-194	0.9681	3824-3988
195-212	0.9682	3988-4407
213-231	0.9687	4407-4866
232-272	0.9688	4866-5897
273-656	0.9689	5897-15361
657-665	0.9690	15361-15580
666-678	0.9691	15580-15917
679-897	0.9692	15917-21792
898-903	0.9693	21792-21945
904-906	0.9694	21945-22021
907	0.9695	22021-22046
908-1000	0.9696	22046-24414

Результат: полученный автомат был проверен путем запуска на самолете в авиасимуляторе и просмотра поведения самолета. Запуск производился два раза. В каждом из запусков автомат выполнил трюк, однако в первом запуске самолет немного отклонился вбок, а во втором случае не смог справиться с управлением в заключительной стадии.

Оценка авторов этому автомату (по десятибалльной шкале) – 5.

Протокол эксперимента №4

Задача: построить управляющий автомат, который выполняет трюк «мертвая петля».
Используется 11 обучающих примеров.
Параметры генетического алгоритма:

- размер поколения – 100 особей;
- число состояний автомата – 4;
- вероятность мутации – 0,5;
- в качестве метода отбора используется турнирный метод;
- размер элиты составляет две особи.

Номер поколения	Максимальное значение функции приспособленности	Время от начала эксперимента (в секундах)
1-2	0.9471	0-21
3-5	0.9500	21-55
6	0.9502	55-67
7-10	0.9506	67-117
11	0.9511	117-130
12-17	0.9535	130-216
18-22	0.9537	216-293
23-27	0.9559	293-371
28-33	0.9564	371-476
34-38	0.9566	476-567
39-40	0.9568	567-603
41-47	0.9587	603-739
48-49	0.9597	739-780
50-51	0.9600	780-821
52-54	0.9621	821-881
55-56	0.9626	881-923
57-61	0.9629	923-1028
62-63	0.9633	1028-1069
64-68	0.9639	1069-1175
69-72	0.9644	1175-1259
73	0.9646	1259-1280
74	0.9649	1280-1302
75-76	0.9651	1302-1347
77	0.9654	1347-1368
78-81	0.9656	1368-1455
82	0.9658	1455-1475
83-85	0.9659	1475-1542
86-90	0.9661	1542-1648
91-95	0.9662	1648-1755
96	0.9665	1755-1778
97-98	0.9666	1778-1820
99-111	0.9668	1820-2104
112-117	0.9669	2104-2238
118-139	0.9670	2238-2720
140-164	0.9671	2720-3275

165-186	0.9672	3275-3755
187-195	0.9673	3755-3955
196-218	0.9688	3955-4491
219-223	0.9689	4491-4610
224-275	0.9690	4610-5901
276-314	0.9692	5901-6870
315-321	0.9693	6870-7040
322-343	0.9694	7040-7578
344-370	0.9695	7578-8255
371-373	0.9696	8255-8329
374-388	0.9700	8329-8713
389-434	0.9701	8713-9929
435-436	0.9704	9929-9982
437-1000	0.9705	9982-23828

Результат: полученный автомат был проверен путем запуска на самолете в авиасимуляторе и просмотра поведения самолета. Запуск производился два раза. В одном из запусков автомат «не захотел» выполнять трюк, в другом выполнил трюк, но незначительно отклонился вбок.

Оценка авторов этому автомату (по десятибалльной шкале) – 4.

Протокол эксперимента №5

Задача: построить управляющий автомат, который выполняет трюк «мертвая петля».
Используется 11 обучающих примеров.
Параметры генетического алгоритма:

- размер поколения – 100 особей;
- число состояний автомата – 4;
- вероятность мутации – 0,5;
- в качестве метода отбора используется турнирный метод;
- размер элиты составляет две особи.

Номер поколения	Максимальное значение функции приспособленности	Время от начала эксперимента (в секундах)
1–4	0,9479	0–75
5	0,9483	75–94
6	0,9489	94–115
7–9	0,9497	115–179
10	0,9510	179–202
11–27	0,9576	202–632
28–34	0,9581	632–833
35–36	0,9590	833–891
37–41	0,9595	891–1049
42	0,9599	1049–1084
43–45	0,9610	1084–1183
46	0,9613	1183–1216
47–49	0,9629	1216–1316
50–52	0,9633	1316–1415
53–56	0,9634	1415–1553
57–62	0,9635	1553–1759
63–68	0,9636	1759–1971
69–76	0,9638	1971–2257
77	0,9640	2257–2296
78–80	0,9642	2296–2403
81–88	0,9643	2403–2693
89–94	0,9644	2693–2917
95–97	0,9645	2917–3032
98–109	0,9646	3032–3492
110–114	0,9648	3492–3687
115–116	0,9649	3687–3765
117	0,9650	3765–3806
118	0,9651	3806–3847
119–126	0,9652	3847–4181
127–133	0,9653	4181–4462
134–139	0,9656	4462–4711
140–155	0,9657	4711–5396
156–158	0,9659	5396–5525

159–161	0,9662	5525–5653
162–168	0,9663	5653–5952
169–181	0,9664	5952–6521
182–187	0,9665	6521–6788
188–203	0,9666	6788–7475
204–306	0,9667	7475–12048
307–362	0,9668	12048–14583
363–373	0,9672	14583–15072
374	0,9678	15072–15116
375–388	0,9679	15116–15722
389–396	0,9690	15722–16065
397–398	0,9691	16065–16152
399–408	0,9692	16152–16572
409–412	0,9693	16572–16748
413–415	0,9694	16748–16882
416–429	0,9695	16882–17490
430–432	0,9696	17490–17620
433–436	0,9697	17620–17798
437–456	0,9698	17798–18667
457–461	0,9699	18667–18883
462–474	0,9700	18883–19417
475–480	0,9701	19417–19660
481–536	0,9702	19660–21924
537–554	0,9703	21924–22692
555–637	0,9704	22692–26633
638–651	0,9705	26633–27297
652–1000	0,9706	27297–44046

Результат: полученный автомат был проверен путем запуска на самолете в авиасимуляторе и просмотра поведения самолета. Запуск производился три раза. В одном из них автомат не смог справиться с управлением самолета при выравнивании после выполнения трюка, ещё в одном запуске самолет немного отклонился вбок. Последний запуск был выполнен на «отлично».

Оценка авторов этому автомату (по десятибалльной шкале) – 8.

Протокол эксперимента №6

Задача: построить управляющий автомат, который выполняет трюк «мертвая петля».
Используется 11 обучающих примеров.
Параметры генетического алгоритма:

- размер поколения – 100 особей;
- число состояний автомата – 4;
- вероятность мутации – 0,5;
- в качестве метода отбора используется турнирный метод;
- размер элиты составляет две особи.

Номер поколения	Максимальное значение функции приспособленности	Время от начала эксперимента (в секундах)
1-2	0.9484	0-21
3-23	0.9523	21-278
24-40	0.9550	278-526
41-48	0.9555	526-647
49-53	0.9560	647-725
54-61	0.9562	725-856
62	0.9566	856-872
63-64	0.9567	872-907
65	0.9568	907-924
66-70	0.9577	924-1013
71-72	0.9579	1013-1050
73	0.9580	1050-1068
74-75	0.9585	1068-1105
76-77	0.9586	1105-1143
78-81	0.9588	1143-1216
82	0.9590	1216-1236
83-86	0.9592	1236-1315
87-91	0.9596	1315-1415
92	0.9597	1415-1434
93	0.9599	1434-1453
94-96	0.9605	1453-1511
97-99	0.9621	1511-1569
100	0.9622	1569-1591
101-109	0.9631	1591-1774
110	0.9633	1774-1795
111-112	0.9635	1795-1838
113	0.9636	1838-1860
114	0.9637	1860-1882
115	0.9638	1882-1902
116-117	0.9639	1902-1943
118-121	0.9640	1943-2026
122-123	0.9642	2026-2069
124-127	0.9643	2069-2155
128	0.9644	2155-2177
129-132	0.9645	2177-2262

133-135	0.9646	2262-2326
136-139	0.9651	2326-2408
140	0.9652	2408-2430
141-149	0.9654	2430-2620
150-168	0.9655	2620-3022
169-185	0.9656	3022-3377
186-189	0.9657	3377-3456
190-202	0.9658	3456-3727
203-247	0.9659	3727-4737
248-730	0.9660	4737-14997
731-1000	0.9661	14997-20579

Результат: полученный автомат был проверен путем запуска на самолете в авиасимуляторе и просмотра поведения самолета. Запуск производился два раза. В одном из запусков автомат «не захотел» выполнять трюк, в другом начал выполнять трюк, но в заключительной стадии не смог справиться с управлением.

Оценка авторов этому автомату (по десятибалльной шкале) – 3.

Протокол эксперимента №7

Задача: построить управляющий автомат, который выполняет трюк «мертвая петля».
Используется 11 обучающих примеров.
Параметры генетического алгоритма:

- размер поколения – 100 особей;
- число состояний автомата – 4;
- вероятность мутации – 0,5;
- в качестве метода отбора используется турнирный метод;
- размер элиты составляет две особи.

Номер поколения	Максимальное значение функции приспособленности	Время от начала эксперимента (в секундах)
1-14	0.9520	0-173
15	0.9522	173-187
16-18	0.9544	187-235
19-29	0.9577	235-411
30-35	0.9589	411-513
36-37	0.9590	513-549
38	0.9592	549-565
39-41	0.9603	565-617
42-43	0.9610	617-653
44-48	0.9620	653-744
49-50	0.9622	744-781
51	0.9623	781-799
52	0.9626	799-817
53-55	0.9637	817-871
56-59	0.9641	871-942
60	0.9647	942-961
61	0.9649	961-978
62-63	0.9652	978-1015
64	0.9657	1015-1033
65-68	0.9668	1033-1109
69-70	0.9672	1109-1149
71-72	0.9673	1149-1189
73-74	0.9675	1189-1227
75-82	0.9677	1227-1394
83-85	0.9678	1394-1455
86-88	0.9682	1455-1517
89-95	0.9688	1517-1663
96-101	0.9689	1663-1789
102-104	0.9691	1789-1850
105-110	0.9692	1850-1975
111-115	0.9694	1975-2084
116-119	0.9695	2084-2175
120-128	0.9696	2175-2382
129-140	0.9697	2382-2669
141-151	0.9698	2669-2947

152-179	0.9700	2947-3669
180-183	0.9701	3669-3772
184-191	0.9702	3772-3972
192-197	0.9703	3972-4127
198-201	0.9704	4127-4226
202-914	0.9712	4226-21815
915-956	0.9714	21815-22854
957-1000	0.9715	22854-23918

Результат: полученный автомат был проверен путем запуска на самолете в авиасимуляторе и просмотра поведения самолета. Запуск производился пять раз. Из них только в одном автомат не смог справиться с управлением самолета в середине трюка, и в одном – незначительно отклонился вбок. В остальных трех запусках самолет под управлением автомата выполнил трюк «мертвая петля» на «отлично».

Оценка авторов этому автомату (по десятибалльной шкале) – 8.

Протокол эксперимента №8

Задача: построить управляющий автомат, который выполняет трюк «мертвая петля».
Используется 11 обучающих примеров.
Параметры генетического алгоритма:

- размер поколения – 100 особей;
- число состояний автомата – 4;
- вероятность мутации – 0,5;
- в качестве метода отбора используется турнирный метод;
- размер элиты составляет две особи.

Номер поколения	Максимальное значение функции приспособленности	Время от начала эксперимента (в секундах)
1	0.9455	0-9
2-4	0.9482	9-40
5-6	0.9494	40-61
7-14	0.9534	61-159
15-22	0.9560	159-284
23-26	0.9572	284-349
27-35	0.9596	349-503
36-43	0.9671	503-643
44-46	0.9673	643-696
47	0.9678	696-714
48-51	0.9680	714-784
52-56	0.9682	784-871
57-61	0.9684	871-957
62	0.9687	957-975
63-64	0.9691	975-1009
65	0.9692	1009-1026
66-67	0.9693	1026-1062
68-72	0.9697	1062-1152
73-76	0.9700	1152-1225
77-78	0.9701	1225-1262
79-85	0.9704	1262-1396
86-88	0.9705	1396-1455
89-92	0.9706	1455-1534
93-106	0.9707	1534-1807
107-117	0.9709	1807-2025
118-120	0.9710	2025-2083
121-126	0.9711	2083-2198
127-133	0.9712	2198-2338
134-154	0.9713	2338-2768
155-215	0.9714	2768-4061
216-710	0.9715	4061-14783
711-768	0.9716	14783-16154
769-1000	0.9717	16154-21530

Результат: полученный автомат был проверен путем запуска на самолете в авиасимуляторе и просмотра поведения самолета. Запуск производился пять раз. В одном из них автомат не смог

справиться с управлением самолета в середине трюка, ещё в одном плохо была выполнена выравнивание, и ещё в одном самолет немного отклонился вбок. В остальных запусках самолет под управлением автомата выполнил трюк без замечаний со стороны авторов.

Оценка авторов этому автомату (по десятибалльной шкале) – 8.

Протокол эксперимента №9

Задача: построить управляющий автомат, который выполняет трюк «мертвая петля».
Используется 11 обучающих примеров.
Параметры генетического алгоритма:

- размер поколения – 100 особей;
- число состояний автомата – 4;
- вероятность мутации – 0,5;
- в качестве метода отбора используется турнирный метод;
- размер элиты составляет две особи.

Номер поколения	Максимальное значение функции приспособленности	Время от начала эксперимента (в секундах)
1-7	0.9502	0-81
8	0.9509	81-94
9-16	0.9519	94-205
17-20	0.9522	205-261
21-25	0.9550	261-337
26-34	0.9551	337-480
35-46	0.9565	480-694
47-52	0.9569	694-803
53	0.9577	803-821
54-59	0.9600	821-933
60-65	0.9604	933-1048
66-67	0.9605	1048-1085
68-69	0.9625	1085-1123
70	0.9636	1123-1140
71-75	0.9644	1140-1231
76-83	0.9647	1231-1387
84-85	0.9651	1387-1430
86-90	0.9652	1430-1539
91-93	0.9653	1539-1601
94-99	0.9654	1601-1728
100-106	0.9655	1728-1876
107-109	0.9656	1876-1937
110-115	0.9657	1937-2066
116-122	0.9658	2066-2214
123-125	0.9659	2214-2276
126-139	0.9660	2276-2586
140-146	0.9661	2586-2742
147	0.9662	2742-2765
148-167	0.9663	2765-3247
168-170	0.9665	3247-3320
171-177	0.9666	3320-3490
178-186	0.9671	3490-3708
187-205	0.9672	3708-4178
206-208	0.9673	4178-4254
209-217	0.9674	4254-4482

218-227	0.9675	4482-4739
228-236	0.9680	4739-4966
237-246	0.9683	4966-5232
247-255	0.9685	5232-5475
256-263	0.9687	5475-5689
264-268	0.9688	5689-5827
269-270	0.9689	5827-5882
271-297	0.9692	5882-6626
298-1000	0.9693	6626-23103

Результат: полученный автомат был проверен путем запуска на самолете в авиасимуляторе и просмотра поведения самолета. Запуск производился два раза. В одном из запусков автомат «не захотел» выполнять трюк, в другом – выполнил трюк, но незначительно отклонился вбок во время выполнения.

Оценка авторов этому автомату (по десятибалльной шкале) – 4.

Протокол эксперимента №10

Задача: построить управляющий автомат, который выполняет трюк «мертвая петля».
Используется 11 обучающих примеров.
Параметры генетического алгоритма:

- размер поколения – 100 особей;
- число состояний автомата – 4;
- вероятность мутации – 0,5;
- в качестве метода отбора используется турнирный метод;
- размер элиты составляет две особи.

Номер поколения	Максимальное значение функции приспособленности	Время от начала эксперимента (в секундах)
1-10	0.9506	0-113
11-21	0.9540	113-267
22-45	0.9568	267-693
46-47	0.9571	693-731
48	0.9574	731-751
49-74	0.9624	751-1267
75-79	0.9637	1267-1370
80-90	0.9650	1370-1600
91	0.9652	1600-1621
92-98	0.9653	1621-1775
99-103	0.9654	1775-1873
104-106	0.9655	1873-1935
107-108	0.9656	1935-1980
109-117	0.9659	1980-2179
118-119	0.9660	2179-2225
120-124	0.9662	2225-2337
125-128	0.9663	2337-2430
129-132	0.9667	2430-2519
133-169	0.9668	2519-3349
170-179	0.9669	3349-3580
180-207	0.9675	3580-4267
208-511	0.9676	4267-11748
512-538	0.9678	11748-12429
539-586	0.9679	12429-13661
587-604	0.9681	13661-14111
605-620	0.9682	14111-14518
621-1000	0.9684	14518-23772

Результат: полученный автомат был проверен путем запуска на самолете в авиасимуляторе и просмотра поведения самолета. Запуск производился два раза. В каждом из запусков автомат выполнил трюк «мертвая петля», однако в первом запуске он значительно «завалился» набок, а во втором – незначительно отклонился набок и не до конца выровнял самолет.

Оценка авторов этому автомату (по десятибалльной шкале) – 6.

ПРИЛОЖЕНИЕ 3. КОПИИ ОПУБЛИКОВАННЫХ В РАМКАХ НИР СТАТЕЙ

ИЗВЕСТИЯ РАН. ТЕОРИЯ И СИСТЕМЫ УПРАВЛЕНИЯ, 2010, № 2, с. 100–117

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ

УДК 004.4'242

МЕТОД СОКРАЩЕННЫХ ТАБЛИЦ ДЛЯ ГЕНЕРАЦИИ АВТОМАТОВ С БОЛЬШИМ ЧИСЛОМ ВХОДНЫХ ПЕРЕМЕННЫХ НА ОСНОВЕ ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ

© 2010 г. Н. И. Поликарпова, В. Н. Точилин, А. А. Шалыто

Санкт-Петербург, Санкт-Петербургский государственный ун-т информационных технологий, механики и оптики

Поступила в редакцию12.08.08 г., после доработки 08.09.09 г.

Известные методы автоматической генерации конечных автоматов на основе генетического программирования неэффективны при большом числе входных переменных автомата. Предложен метод, который не имеет указанного недостатка. Предпочтительность применения метода при большом числе входных переменных обоснована теоретически и подтверждена экспериментально. Метод был использован для автоматизации разработки системы управления самолетом на высоком уровне абстракции.

Введение. В литературе рассматривается ряд методов построения конечных автоматов с помощью генетических алгоритмов [1, 2], генетического программирования [3, 4] и других эволюционных подходов. Большинство работ в этой области посвящено генерации автоматов-*распознавателей*, описывающих грамматику некоторого языка. Задача такого автомата состоит в проверке принадлежности заданной строки языку. Распознаватель не производит выходных воздействий: результат определяется состоянием автомата после обработки входной последовательности. Примером распознавателя может служить синтаксический анализатор, устанавливающий соответствие кода программы грамматике языка программирования. В данном направлении как наиболее значимые можно выделить работы [5–12]. Более сложная форма конечного автомата – *преобразователь* – отображает множество входных строк на множество выходных, возможно, над другим алфавитом. Примером преобразователя является компилятор. Эволюционному построению преобразователей посвящены публикации [13–15].

Для обоих рассмотренных выше классов автоматов входными воздействиями выступают символы некоторого заданного (чаще всего – небольшого) алфавита. Иначе говоря, в качестве условия перехода автомата из одного состояния в другое используется сравнение текущего входного символа с одним из небольшого множества заданных символов.

В области *генетического программирования* в явном виде задача построения конечных автоматов не ставится. Однако в этой области традиционно оптимизируются модели вычислений в виде графов, которые можно интерпретировать как графы переходов конечных автоматов. Построению программ в виде графов посвящено большое число работ, например [16–20]. Применение автоматов в играх описано в [21–23]. Небольшое число работ затрагивают вопросы построения *управляющих автоматов*, описывающих логику сложного поведения сущности или системы. Например, в [24–26] рассматривается автоматическое построение компонент программного обеспечения логических контроллеров в виде автоматов, а в [27] оценивается эффективность автоматных моделей применительно к различным задачам.

Практически во всех упомянутых исследованиях автомат в каждый момент времени обрабатывает только одну входную переменную. Исключения составляют лишь работы [22] (четыре параллельных троичных входа) и [27] (в качестве условия перехода допускается сравнение значений двух регистров). Теоретически любое число параллельных входов можно свести к одному, в качестве воздействий которого выступают комбинации сигналов исходных параллельных входов. Однако размер алфавита полученного таким образом входа растет экспоненциально с увеличением числа исходных параллельных входов. В указанных публикациях параллельные входы не приводят к недопустимо большому алфавиту, но для реальных систем управления эта проблема крайне актуальна. Также в перечисленных работах автомат на каждом шаге может сгенерировать не более одной выходной переменной. Это вызывает экспоненциальный рост выходного алфавита, так как в реальных задачах автомату часто необходимо на каждом шаге реализовывать произвольные комбинации элементарных действий. Отметим, что в [27] допускаются действия с аргументами, а также параллельно выполняемые автоматы, отвечающие за различные действия, что значительно ослабляет проблему.

В настоящей статье исследуется задача автоматического построения управляющих автоматов на основе генетического программирования. В качестве условий переходов в управляющих автоматах выступают булевы функции над произвольным числом входных переменных. Из перечисленных выше источников наилучшие результаты в области построения управляющих автоматов получены в работах [25, 26], посвященных созданию программы управления роботом. Однако и эти публикации не лишены упомянутых выше недостатков, относящихся к размерности входных и выходных воздействий.

Для построения управляющих автоматов в статье предлагается *метод сокращенных таблиц*. При разработке данного метода наибольшее внимание уделялось возможности получения автоматов с произвольным числом параллельных входов и выходов. В отличие от существующих методов генетического программирования, в которых программа описывается на низком уровне абстракции и целиком подвергается оптимизации, в развиваемом подходе программа представляется в виде *автоматизированного объекта управления* [28]: логика сложного поведения выражается автоматом или системой автоматов на высоком уровне абстракции и подвергается оптимизации, в то время как объект управления реализуется вручную (аппаратно или программно) и оптимизации не предполагает. Объект управления может быть произвольным (и, вообще говоря, сколь угодно сложным). Таким образом, предлагаемый метод решает задачу об использовании сложных структур данных в рамках генетического программирования, поставленную основателем генетического программирования *J. Koza* в [29].

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

Объект O может управляться автоматом вида $A = \langle S, s_0, \Delta\rangle$, где S – конечное множество *управляющих состояний*, s_0 – *стартовое состояние*, $\Delta : S \times \{0,1\}^n \to S \times Z^*$ – *управляющая функция*, сопоставляющая управляющему состоянию и *входному воздействию* новое состояние и *выходное воздействие*. Управляющую функцию можно разложить на две компоненты: функцию выходов $\zeta : S \times \{0,1\}^n \to Z^*$ и функцию переходов $\delta : S \times \{0,1\}^n \to S$. Отдельные компоненты входного воздействия, соответствующие предикатам объекта управления, называются *входными переменными*. Отдельные компоненты выходного воздействия, соответствующие действиям объекта управления, именуются *выходными переменными*.

Пусть перед началом работы объект управления находится в вычислительном состоянии v_0, а автомат – в управляющем состоянии s_0. Следующая последовательность операций составляет *шаг работы* автоматизированного объекта:

1) объект управления вызывает все предикаты из множества X и формирует из их значений вектор *входного воздействия* $in \in \{0,1\}^n$;

2) автомат вычисляет значение вектора *выходного воздействия* $out = \zeta(s, in)$, где s – текущее состояние автомата, и переходит в новое управляющее состояние $s_{new} = \delta(s, in)$;

3) объект управления реализует действия $z \in out$, изменяя при этом текущее вычислительное состояние [28].

Задача построения управляющего автомата состоит в том, чтобы найти автомат заданного вида, такой, что за k шагов работы под его управлением объект O перейдет в вычислительное состояние с максимальной приспособленностью ($\varphi(v) \to \max$)). В связи с использованием генетического программирования для этой задачи возникают следующие подзадачи: выбор представления конечного автомата в виде набора хромосом; адаптация генетических операторов (мутации и скрещивания) для этого представления; настройка параметров генетической оптимизации. Для решения первой из подзадач автомат удобно интерпретировать как набор управляющих состояний, каждое из которых задается проекцией управляющей функции $\Delta_s : \{0,1\}^n \to S \times Z^*$, $s \in S$. Таким образом, задача сводится к описанию каждого управляющего состояния в отдельности в виде хромосомы.

2. Представление состояния в виде хромосомы. Естественный способ представления управляющей функции в состоянии – *таблица переходов и выходов*, в которой каждой возможной комбинации значений входных переменных сопоставляется множество значений выходных переменных и новое состояние. Основная проблема, возникающая при этом – экспоненциальный рост размера хромосомы с увеличением числа предикатов объекта управления, так как число строк в таблице составляет 2^n, где n – количество предикатов.

Опыт показывает, что в реальных задачах число переходов в управляющих автоматах, сформированных вручную, не возрастает экспоненциально с увеличением числа предикатов объекта управления. Причина состоит, по-видимому, в том, что в большинстве задач предикаты имеют

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

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

Рис. 1. Хромосома состояния: сокращенная таблица ($n = 6$, $r = 2$)

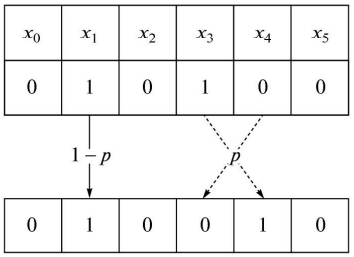


Рис. 2. Пример мутации множества значимых предикатов

"локальную природу" по отношению к управляющим состояниям. В каждом состоянии *значимым* является лишь определенный, небольшой поднабор предикатов, остальные же не влияют на значение управляющей функции. Именно это свойство делает возможным существенное сокращение размера описания состояний. Кроме того, использование этого свойства в процессе оптимизации позволяет получить результат, более похожий на автомат, построенный вручную, а следовательно, и более понятный человеку.

Свойство локальности предикатов можно применять для уменьшения размера описания управляющего состояния разными способами. Авторами выбран один из подходов, при котором число значимых в состоянии предикатов ограничивается некоторой константой r. В этом случае состояние автомата представляется в виде *сокращенной таблицы* переходов и выходов, в которой каждой комбинации значений *значимых* входных переменных сопоставляется множество значений выходных переменных и новое состояние. Кроме того, представление содержит битовый вектор, описывающий множество значимых предикатов (рис. 1).

Число строк сокращенной таблицы составляет 2^r. В автоматах, разработанных вручную, число значимых предикатов в состоянии обычно колеблется от одного до пяти. Это позволяет предположить, что хорошие результаты оптимизации могут быть достигнуты даже при небольших значениях константы r. Таким образом, объем памяти, необходимой процессу оптимизации для метода сокращенных таблиц, можно оценить как $O(g|S|(|Z| + |X|))$, где g – число особей в поколении. Отметим, что в отличие от объема требуемой памяти оценку быстродействия алгоритма генетической оптимизации трудно найти теоретически, так как он является вероятностным. В связи с этим оценка быстродействия была проведена авторами экспериментально (разд. 4).

3. Генетические операторы. Опишем генетические операторы над хромосомами состояний, представленные в виде сокращенных таблиц.

Алгоритм 1. Мутация сокращенных таблиц. При мутации состояния с некоторой вероятностью может измениться как значение в каждой строке сокращенной таблицы, так и множество значимых предикатов. При этом каждый из значимых предикатов с заданной вероятностью заменяется другим, который не принадлежит множеству (рис. 2). Таким образом, число значимых предикатов в состоянии остается постоянным. Описание алгоритма мутации приведено в листинге на рис. 3.

Алгоритм 2. Скрещивание сокращенных таблиц. Основные шаги алгоритма скрещивания отражены в листинге на рис. 4. Поскольку родительские хромосомы, представленные сокращенными таблицами, могут иметь разные множества значимых предикатов, сначала необходимо выбрать, какие из этих предикатов будут присутствовать в хромосомах детей. Функция `ChoosePreds`, осуществляющая этот выбор, отражена в листинге на рис. 5. В результате работы функции `ChoosePreds` предикаты, значимые для обоих родителей, наследуются обоими детьми, а каждый из предикатов, участвующих лишь в одной родительской особи, равновероятно достанется любому из двух детей.

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

Рис. 3. Мутация сокращенных таблиц (листинг)

Пример работы функции для родительских хромосом, присутствующих на рис. 6, проиллюстрирован на рис. 7. После выбора значимых предикатов формируются таблицы обоих детей. Алгоритм заполнения содержится в листинге на рис. 8. Иллюстрация примера заполнения первой строки таблицы одного из детей приведена на рис. 9. В данной реализации оператора скрещивания на значения каждой строки таблицы ребенка влияют величины из нескольких строк родительских таблиц. При этом конкретная величина, помещаемая в ячейку таблицы ребенка, определяется “голосованием” всех влияющих на нее ячеек родительских таблиц.

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

4. Экспериментальная оценка метода сокращенных таблиц. Для оценки характеристик метода сокращенных таблиц был проведен ряд экспериментов, в которых он сравнивался с представлением управляющих автоматов в виде полных таблиц переходов и выходов [30]. Сравнение выполнялось по следующим критериям: объем занимаемой памяти; время, затрачиваемое на обработку каждого поколения; скорость роста функции приспособленности (от номера поколения и от времени).

Как было упомянуто выше, метод сокращенных таблиц решает проблему экспоненциального роста размера описания автомата с увеличением числа входных переменных (предикатов объекта управления). Это свойство метода подтвердилось при экспериментальной проверке. Во время эксперимента описания автоматов хранились в памяти компьютера, поэтому объем занимаемой памяти в этом случае прямо пропорционален размеру описания автомата. Результаты эксперимента

```
pair<State, State> Cross(State statel, State state2)
{
      State child! = statel;
      State child2 = childl;

      ChoosePreds(statel.predicates, state2.predicates,
            childl.predicates, child2.predicates);

      int. crossPoint = случайное число от 0 до tableSize;
      FillChildTable(statel, state2, childl, crossPoint);
      FillChildTable(statel, state2, child2, crossPoint);
      return make_pair(childl, child2);
```

Рис. 4. Скрещивание сокращенных таблиц (листинг)

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

Рис. 5. Выбор значимых предикатов детей при скрещивании сокращенных таблиц (листинг)

приведены на рис. 10, *a*. Из рассмотрения графика следует, что объем занимаемой памяти при использовании метода полных таблиц растет экспоненциально, в то время как для метода сокращенных таблиц этот объем с ростом числа предикатов практически не изменяется. В действительности он зависит от числа предикатов линейно, однако коэффициент линейной связи настолько мал, что в экспериментах она не отражается. Аналогичный характер имеет зависимость времени, требуемого на обработку каждого поколения, от числа предикатов (рис. 10, *b*).

Теперь перейдем к оценке скорости роста функции приспособленности. На рис. 10, *c* содержатся зависимости значения оценочной функции от номера поколения для методов полных и сокращенных таблиц (измерения приводились при небольшом числе предикатов). Из последнего графика следует, что оптимизация методом полных таблиц требует вычисления меньшего числа поколений. Это означает, что при небольшом числе предикатов метод полных таблиц обладает более высоким быстродействием. Однако с ростом числа предикатов стремительно растет время обработки одного поколения таким методом. Кроме того, из-за экспоненциального увеличения объема требуемой памяти применение метода полных таблиц, начиная с некоторого числа предикатов, становится не просто неэффективным, а практически невозможным. В экспериментах авторам не удалось получить методом пол-

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

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

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

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

Рис. 6. Родительские хромосомы, представленные сокращенными таблицами

ных таблиц автомат с более чем 14 входными переменными. Отметим также, что автоматы, которые построены методом полных таблиц, практически невозможно изобразить и понять, так как в них присутствует большое число избыточных переходов, а условия на переходах громоздки. Напротив, автоматы, основанные на методе сокращенных таблиц, могут быть сравнительно легко поняты человеком.

Для того чтобы адекватно оценить быстродействие обоих методов, необходимо установить зависимость значения оценочной функции, достигаемого с помощью каждого из них за определенное время, от числа предикатов. Такая зависимость для промежутка времени, равного 5 мин, присутствует на рис. 10, *d*. Как и ожидалось, приведенные зависимости показывают, что при небольшом числе предикатов метод полных таблиц имеет более высокое быстродействие, однако с ростом числа предикатов его быстродействие резко падает. В то же время быстродействие метода сокращенных таблиц с ростом числа предикатов уменьшается незначительно.

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

при небольшом числе предикатов оптимизация с использованием метода полных таблиц требует меньше времени, однако построенные этим методом автоматы непонятны человеку;

начиная с некоторого числа предикатов применение метода полных таблиц практически невозможно, в то время как метод сокращенных таблиц дает достаточно хорошие результаты относительно требуемого времени и памяти.

Для генерации автоматов на основе описанного метода авторами было разработано инструментальное средство [30, 31].

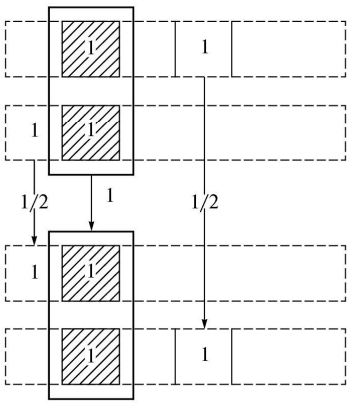


Рис. 7. Пример выбора значимых предикатов детей

5. Применение метода сокращенных таблиц для автоматического управления самолетом. Предложенный метод был применен для проектирования системы автоматического управления моделью самолета на высоком уровне абстракции. Известно, что в настоящее время при управлении самолетами широко используются автопилоты, однако они не обеспечивают полностью автоматического полета. В частности, переключение между режимами автопилота и настройка навигационных приборов производится вручную. В работе была поставлена задача построения конечного автомата, управляющего самолетом, который позволил бы полностью автоматизировать полет. При этом в качестве объекта управления автомат может привлекать стандартный автопилот. Разрабатываемая система не должна предъяв-

```
void FillChildTable(State s1, State s2, States child,int crossPoint) {
      for (для всех i: строк таблицы child) {
            vector<int> linesl = выбрать строки таблицы s1, в которых предикаты,
                  значимые для child, имеют те же значения, что в строке i,
                  причем, если предикат значим для обоих родителей и i >=
                  crossPoint, то его значение не учитывается;
            vector<int> lines2 = выбрать строки таблицы s2, в которых предикаты,
                  значимые для child, имеют те же значения, что в строке i,
                  причем, если предикат значим для обоих родителей и i <
                  crossPoint, то его значение не учитывается;
            vector<Probability> pi(nStates);
            vector<Probability> p2(nStates);
            for (для всех j из linesl) {
                  p1[целевое состояние у s1 в строке j] += 1.0;
            }
            for (для всех j из lines2) {
                  p2[целевое состояние у s2 в строке j] += 1.0;
            }
            Поделить значения p1 на число строк из linesl;
            Поделить значения p2 на число строк из lines2;
            vector<Probability> p = pi + p2;
            child[i].targetState = случайное с распределением вероятностей p;
            for (для всех k: номеров действий) {
                  Probability ql, q2;
                  for (для всех j из linesl) {
                        ql += s1[j].output[k];
                  }
                  for (для всех j из lines2) {
                        q2 += s2[j].output[k];
                  }
                  Поделить ql на число строк из linesl;
                  Поделить q2 на число строк из lines2;
                  child[i].output[k] = 1 с вероятностью (ql + q2)/2 и 0 иначе;
            }
      }
}
```

Рис. 8. Заполнение таблиц детей при скрещивании сокращенных таблиц (листинг)

лять нестандартных требований к наземному оборудованию.

5.1. Методика генерации автомата, управляющего самолетом. Для решения сформулированной выше задачи при помощи генетического программирования на основе метода сокращенных таблиц были выполнены следующие шаги:

выбор авиасимулятора;

реализация интерфейса объекта управления;

задание функции приспособленности;

построение управляющего автомата;

анализ результатов эксперимента.

5.2. Выбор авиасимулятора. В рассматриваемой задаче объектом управления является самолет, находящийся в некоторой среде, включающей воздух (возможно, движущийся), взлетно-посадочные полосы, ландшафт, службу управления полетами, радиомаяки и другое. Требуется эмулировать аэродинамику, механику и работу оборудования самолета. Ввиду трудоемкости разработки необходимого эмулятора представляется естественным использование эмулятора стороннего производителя. Для этой цели был выбран авиационный симулятор *X-Plane* [32], особенности которого – точность физического моделирования и документированный интерфейс взаимодействия с другими программами (API), что сделало возможным его применение в проводимом эксперименте.

5.3. Выбор интерфейса объекта управления. Симулятор *X-Plane* позволяет получать от самолета и передавать ему большое количество данных. В настоящем эксперименте не представлялось возможным задействовать в

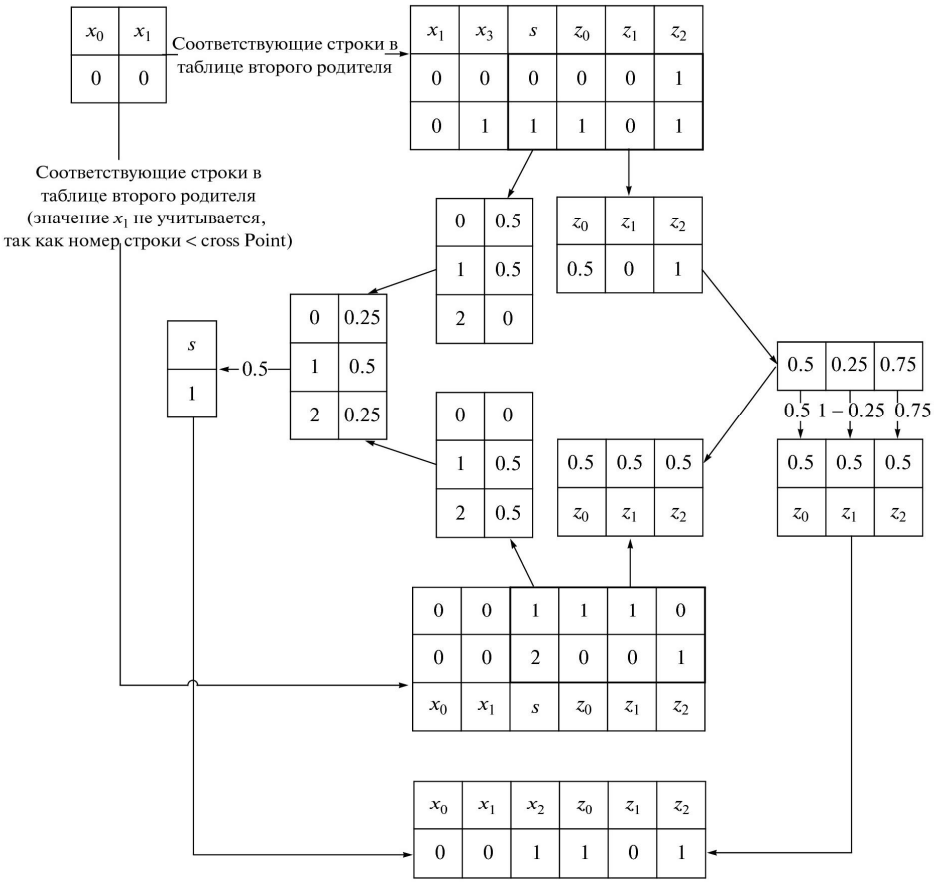


Рис. 9. Пример заполнения строки таблицы ребенка при скрещивании сокращенных таблиц

управляющем автомате все эти данные, так как процесс оптимизации в этом случае был бы слишком долгим. Как было упомянуто выше, реалистичность предлагаемого метода генерации управляющих автоматов обеспечивается высоким уровнем абстракции логики. При этом предикаты и действия объекта управления выбираются и реализуются вручную.

Таблица 1. Входные воздействия автомата

Идентификатор	Описание значения
x_1	Самолет движется
x_2	Скорость самолета достаточна для полета с выпущенными закрылками
x_3	Скорость самолета достаточна для полета с убранными закрылками
x_4	Самолет летит
x_5	Самолет находится рядом с поверхностью земли
x_6	Высота полета соответствует рекомендованной высоте эшелона
x_7	Самолет находится рядом с опорной точкой *GPS*-приемника

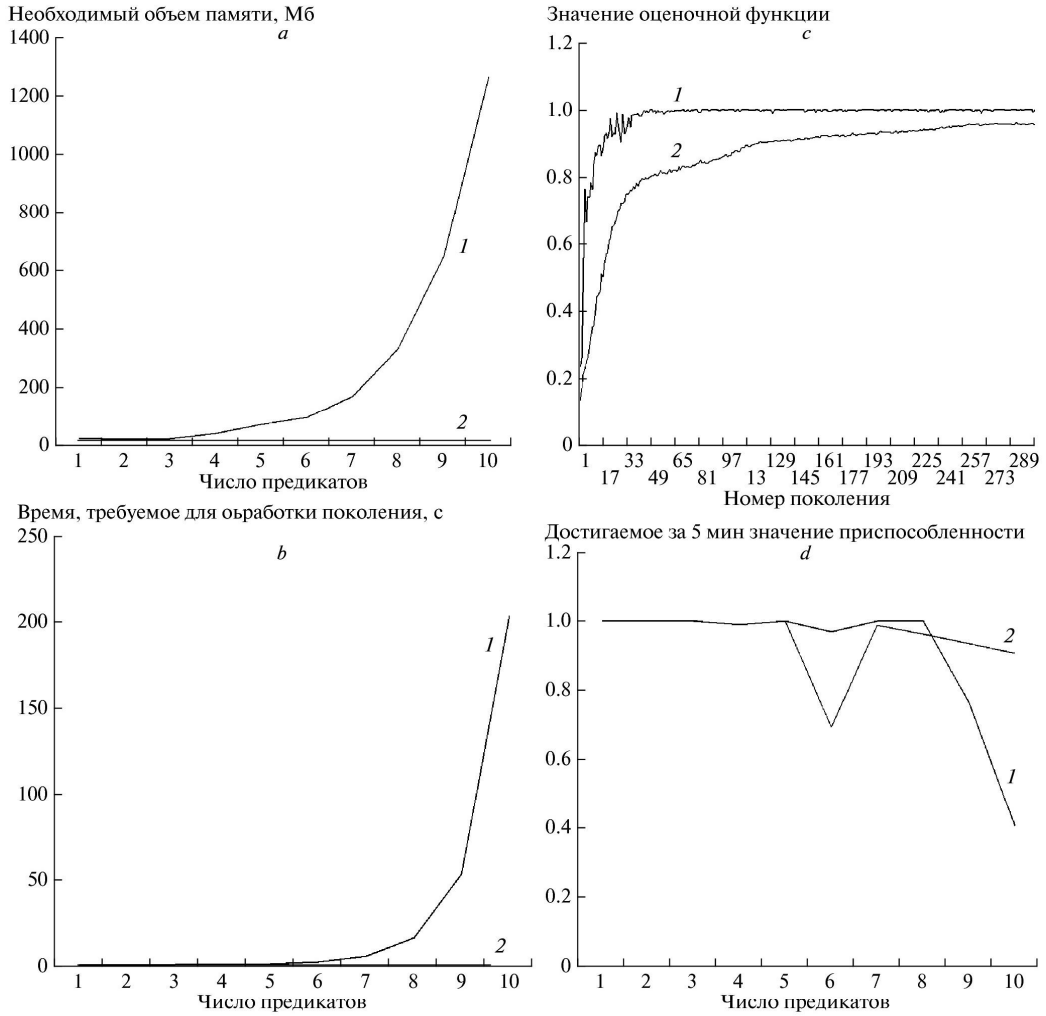


Рис. 10. Зависимости объема занимаемой памяти *a*, времени обработки поколения *b*, значения оценочной функции для заданной продолжительности работы метода *d* от числа предикатов и значения оценочной функции от номера поколения *c*; *1* – полные таблицы, *2* – сокращенные таблицы

В исследуемой задаче были использованы 7 входных (табл. 1) и 24 выходных переменных (табл. 2), которых, по мнению авторов, было достаточно для управления самолетом на высоком уровне абстракции. Для каждой из выбранных переменных была разработана подпрограмма, соответствующая предикату или действию объекта управления. Эти подпрограммы взаимодействуют с авиасимулятором, считывая показания приборов и воздействуя на органы управления самолетом. Состав приборов и органов управления приведен на рис. 11.

5.4. Функция приспособленности. Переформулируем задачу в виде функции приспособленности. От автопилота требуется провести самолет по маршруту и не разбить его. Следовательно, можно выделить два значимых фактора: отклонение от маршрута и сохранность самолета. Кроме того, важно, чтобы после про-

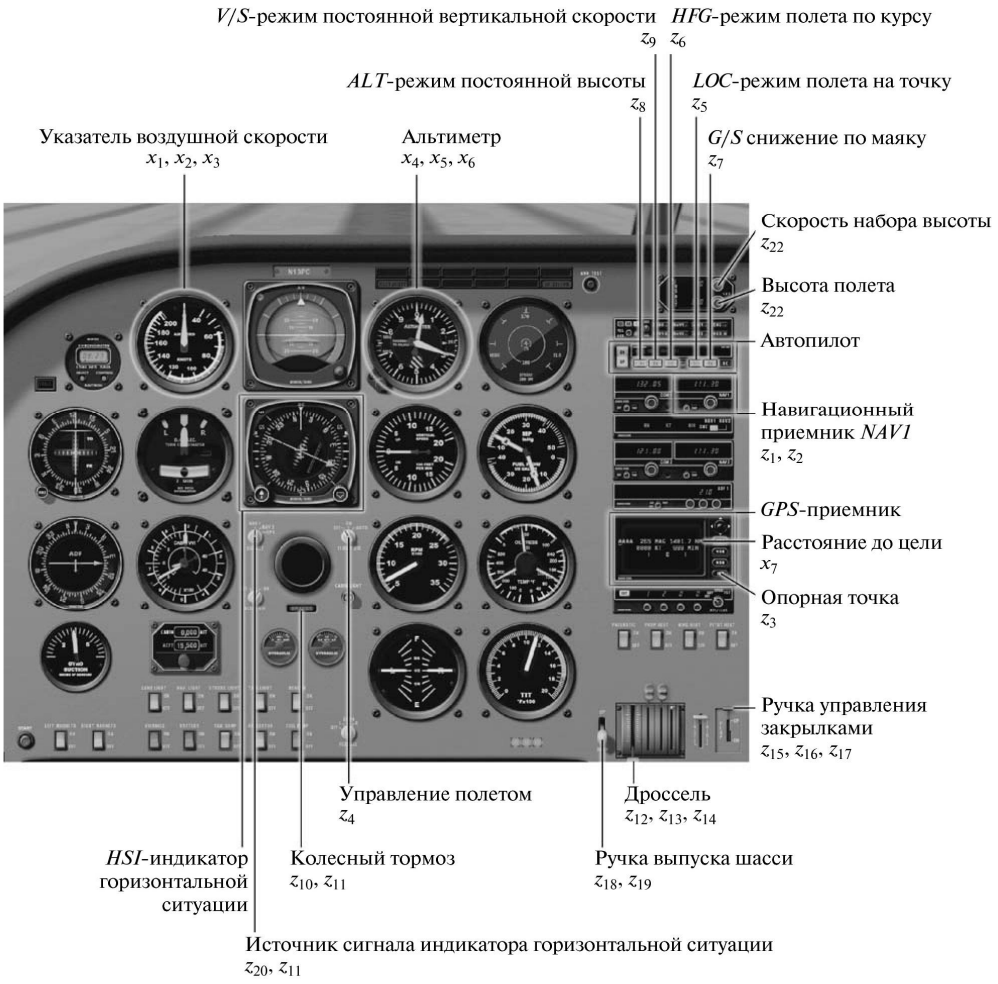


Рис. 11. Используемые приборы и органы управления самолетом

хождения маршрута самолет остановился (или снизил скорость до безопасной) на взлетно-посадочной полосе. Совокупное отклонение от маршрута как по положению, так и по скорости будем вычислять в виде интегралов модулей соответствующих отклонений по времени

$$P = \int_{t_0}^{t_1} |p(t)|\,dt,$$

где P – совокупное отклонение от маршрута по положению, t_0 – время начала эмуляции, t_1 – время окончания эмуляции, $p(t)$ – отклонение по положению в момент времени t;

$$V = \int_{t_0}^{t_1} |v(t)|\,dt,$$

где V – совокупное отклонение от оптимальной скорости, а $v(t)$ – отклонение от оптимальной скорости в момент времени t. Обозначим через C_α, P_α и V_α весовые коэффициенты аварии, отклонения по положению и отклонения по скорости. Коэффициенты определяют относительную

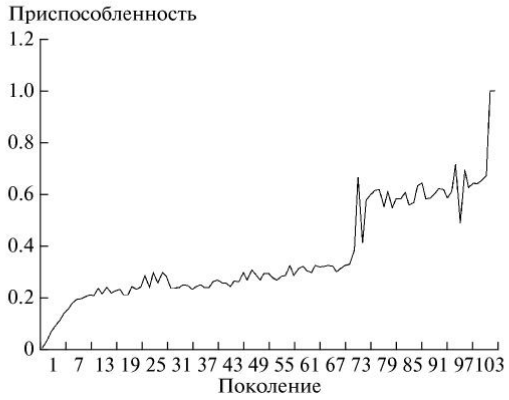


Рис. 12. Изменение приспособленности в процессе эволюции

значимость соответствующих дефектов поведения автопилота. Для них были установлены значения $C_\alpha = 9$, $P_\alpha = 0.01$ 1/м и $V_\alpha = 0.01$ с/м. В результате авторами была выбрана функция приспособленности вида

$$f = \frac{t_1 - t_0}{t_i\left(1 + \dfrac{PP_\alpha + VV_\alpha}{t_1 - t_0}\right)(1 + bC_\alpha)},$$

где b – переменная, равная нулю, если самолет цел, и единице, если он разбит; $t_i = 77$ с – время, за которое при точном следовании маршруту (что физически невозможно) автопилот набрал бы то же значение функции приспособленности, которое "идеальный" физически возможный автопилот получает за весь перелет. Функция нормирована к интервалу $(0,1)$ по сравнению с "идеальным" автопилотом. Таким образом, при значении функции единица цель оптимизации достигнута.

Таблица 2. Выходные воздействия автомата

Идентификатор	Описание действия
z_1	Настройка навигационного приемника на частоту посадочного маяка аэропорта отправления
z_2	Настройка навигационного приемника на частоту посадочного маяка аэропорта прибытия
z_3	Перевод *GPS*-приемника в режим следования к опорной точке
z_4	Установка переключателя управления полетом в положение "АВТО"
z_5	Переключение автопилота в режим полета на точку в горизонтальной плоскости
z_6	Переключение автопилота в режим полета по курсу в горизонтальной плоскости
z_7	Переключение автопилота в режим снижения по маяку
z_8	Переключение автопилота в режим постоянной высоты
z_9	Переключение автопилота в режим постоянной вертикальной скорости
z_{10}	Включение колесного тормоза
z_{11}	Выключение колесного тормоза
z_{12}	Установка максимальной подачи топлива
z_{13}	Установка средней подачи топлива
z_{14}	Установка минимальной подачи топлива
z_{15}	Убирание закрылков
z_{16}	Установка закрылков во взлетное положение
z_{17}	Установка закрылков в посадочное положение
z_{18}	Выпуск шасси
z_{19}	Убирание шасси
z_{20}	Использование навигационного приемника в качестве источника сигнала индикатора горизонтальной ситуации
z_{21}	Использование *GPS*-приемника в качестве источника сигнала индикатора горизонтальной ситуации
z_{22}	Установка скорости набора высоты и высоты полета на автопилоте
z_{23}	Управление рулем направления в соответствии с сигналами автопилота
z_{24}	Установка руля направления в центральное положение

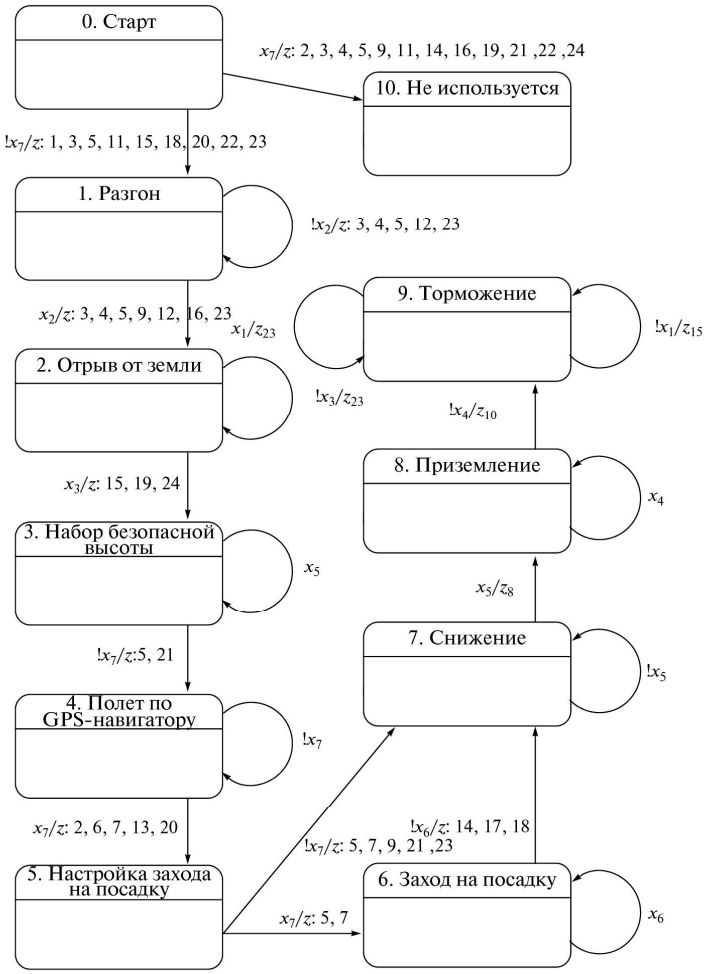


Рис. 13. Граф переходов полученного автомата

5.5. Построение управляющего автомата. Алгоритм построения управляющего автомата состоит из следующих шагов:

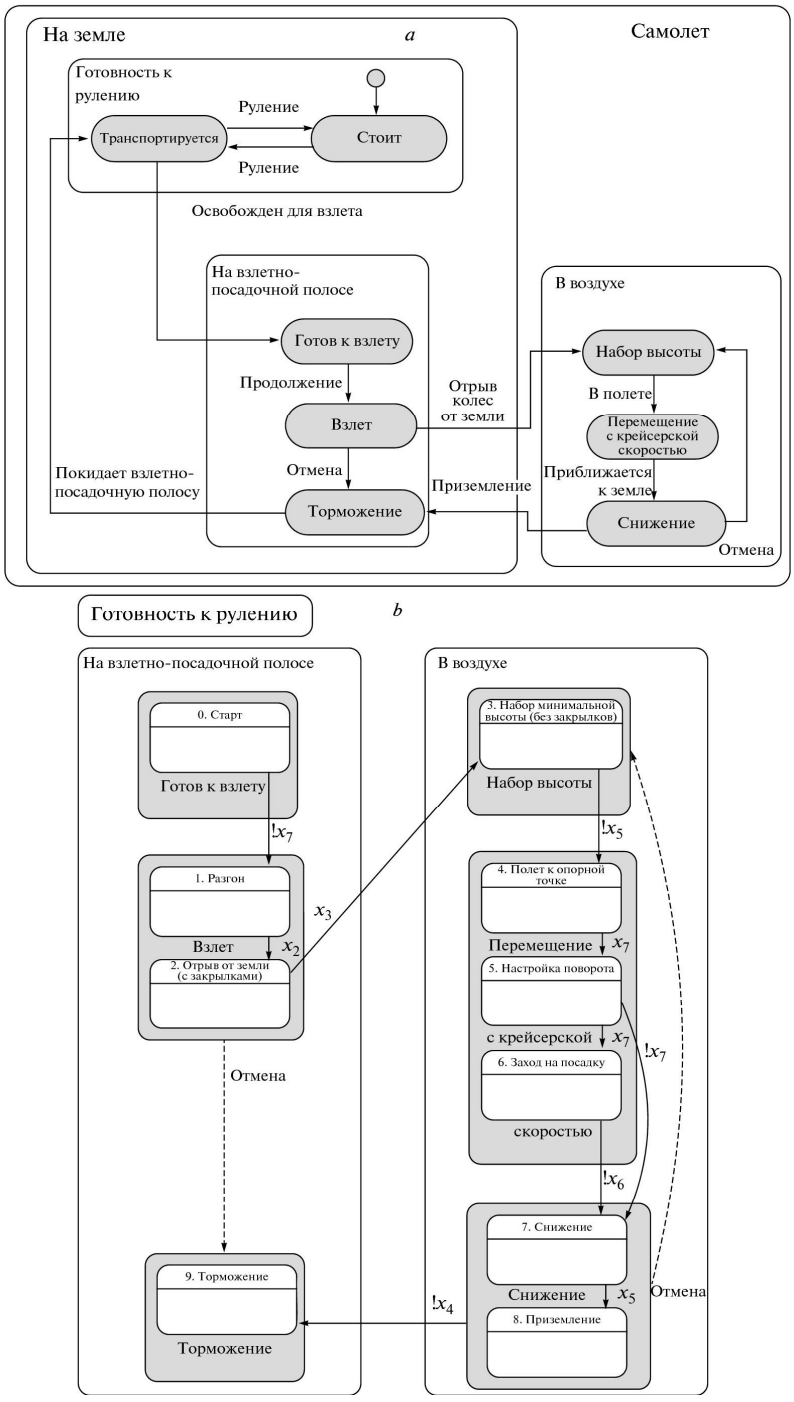
1) исходная популяция автоматов генерируется случайным образом;

2) выполняется оценка каждого автомата в популяции. Для этого производится эмуляция полета в авиасимуляторе под управлением оцениваемого автомата и записываются данные о маршруте и состоянии самолета. Далее на основе этих данных вычисляется значение функции приспособленности;

3) если в популяции имеется автомат со значением функции приспособленности, равным единице, он выбирается в качестве результата и процесс оптимизации завершается;

4) в противном случае путем применения операторов мутации и скрещивания строится новое поколение, после чего процесс возвращается к шагу 2.

Рис. 14. Диаграмма состояний для полета самолета из книги [33] *a* и ее сопоставление с автоматически построенным автоматом *b*


На земле
a
Самолет
Готовность к рулению
Руление
Транспортируется
Стоит
Руление
Освобожден для взлета
На взлетно-посадочной полосе
В воздухе
Готов к взлету
Набор высоты
Продолжение
Отрыв колес от земли
В полете
Взлет
Перемещение с крейсерской скоростью
Покидает взлетно-посадочную полосу
Отмена
Приземление
Приближается к земле
Торможение
Снижение
Отмена
Готовность к рулению
b
На взлетно-посадочной полосе
В воздухе
0. Старт
Готов к взлету
3. Набор минимальной высоты (без закрылков)
Набор высоты
!x7
!x5
1. Разгон
Взлет
2. Отрыв от земли (с закрылками)
x3
x2
4. Полет к опорной точке
Перемещение
x7
5. Настройка поворота
с крейсерской
x7
!x7
6. Заход на посадку
скоростью
Отмена
!x6
7. Снижение
Снижение
x5
Отмена
8. Приземление
!x4
9. Торможение
Торможение

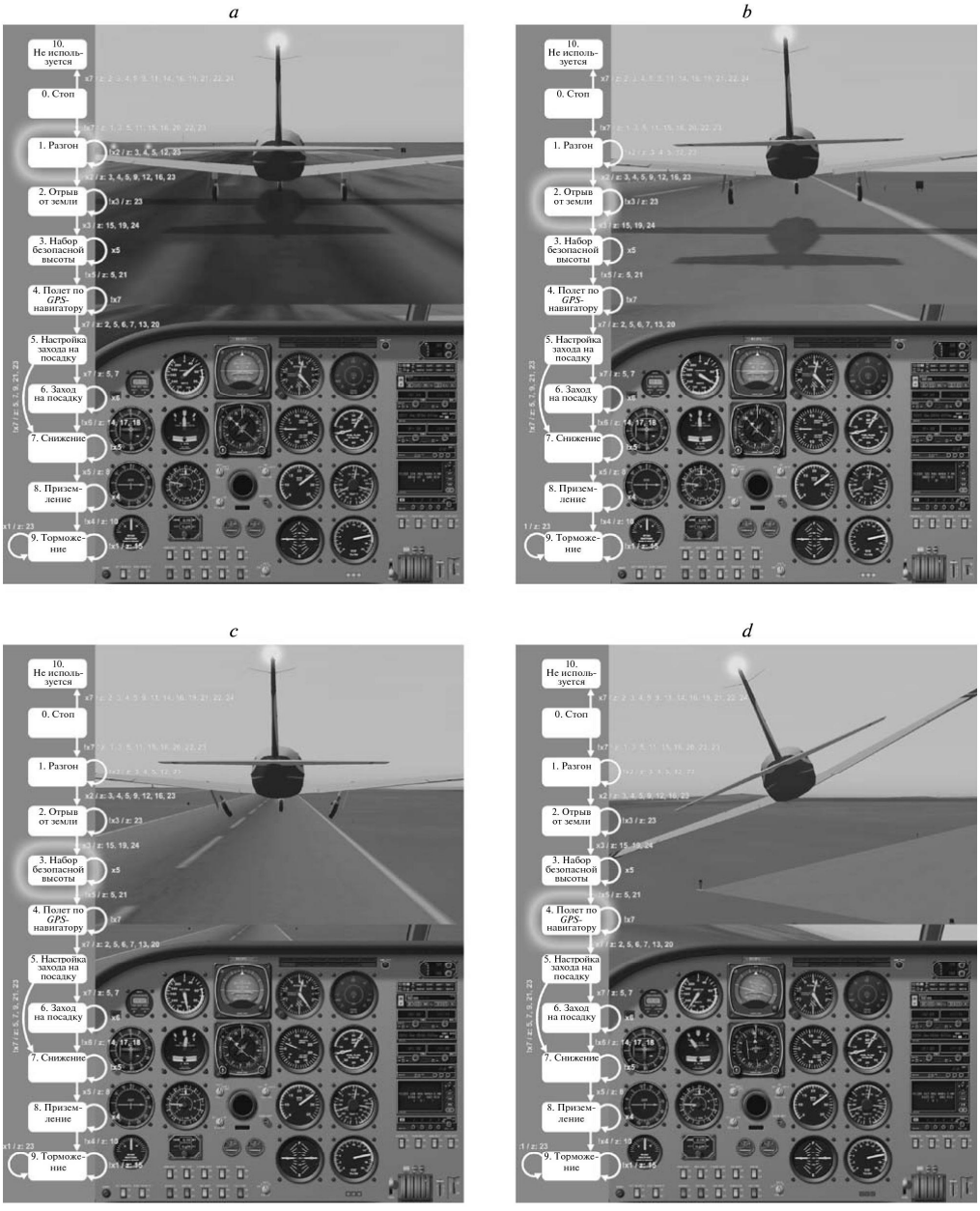


Рис. 15. Автомат и самолет в различных состояниях: разгон *a*, отрыв от земли *b*, набор безопасной высоты *c*, полет по GPS-навигатору *d*, настройка захода на посадку *e*, заход на посадку *f*, снижение *g*, приземление *h*, торможение *i*

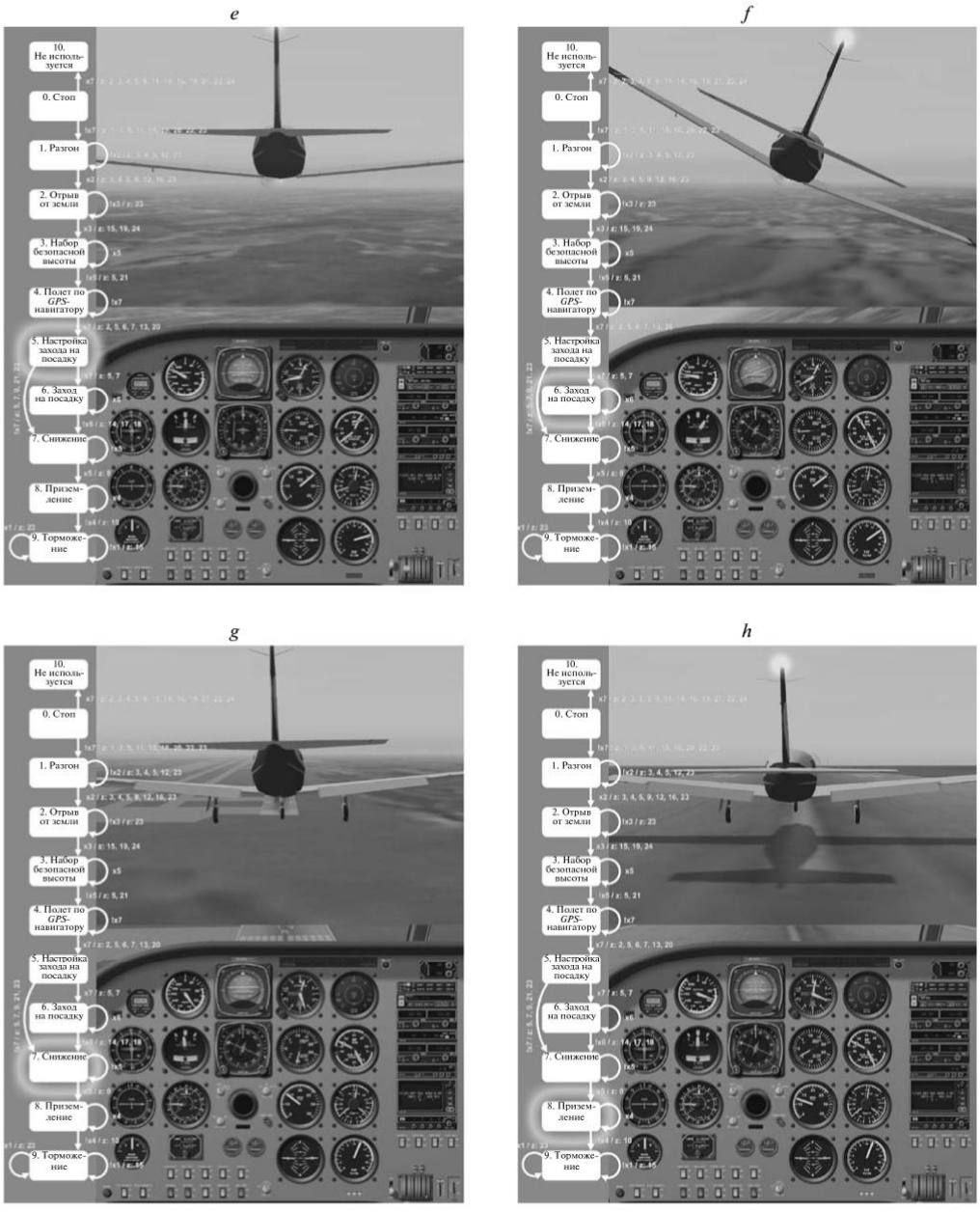


Рис. 15. (Продолжение)

График изменения значения функции приспособленности с ростом номера поколения в процессе работы описанного алгоритма приведен на рис. 12. Максимальное значение оценочной функции, равное единице, для двух последних поколений свидетельствует об успешном завершении процесса оптимизации.

При генерации автомата было наложено ограничение на число значимых предикатов в каждом состоянии $r = 1$. Даже при таком ограничении процесс оптимизации продолжался около месяца. Отметим, что большую часть времени занимало вычисление функции приспособленности: эмуляция, необходимая для оценки одного автомата, выполнялась около 5 мин. Полученный в результате оптимизации автомат имеет небольшое число состояний и переходов. Однако он достаточно сложен для восприятия человеком ввиду значительного числа действий, вызываемых на переходах. В целях упрощения автомата действия, отменяющие друг друга или не влияющие на поведение самолета в данном состоянии, были удалены вручную. Кроме того, состояния автомата были пронумерованы и снабжены названиями. Окончательный вариант автомата приведен на рис. 13. Поскольку число значимых предикатов в каждом состоянии управляющего автомата было установлено равным единице, из каждого состояния могло быть не более двух переходов. Благодаря тому, что предикаты и действия объекта управления были выбраны на достаточно высоком уровне абстракции, для управления оказалось достаточно автомата с настолько простой структурой.

Автомат, изображенный на рис. 13, может быть сравнительно легко построен эвристически (без использования генетического программирования). Например, в [33] в качестве примера диаграммы состояний приведен очень похожий граф переходов, описывающий полет самолета (рис. 14, *a*). На рис. 14, *b* состояниям и переходам этого автомата сопоставляются состояния и переходы автомата, полученного с помощью генетической оптимизации в настоящей работе. Отличия между двумя графами переходов объясняются различиями в постановке задачи. Однако часто построить управляющий автомат вручную нелегко. Такой автомат в большинстве случаев будет неоптимальным и неоправданно сложным. Кроме того, оптимизационный подход характеризуется лучшей масштабируемостью.

5.6. Анализ результатов эксперимента. После завершения процесса оптимизации была проведена тестовая эмуляция полета под управлением построенного оптимального автомата. На рис. 15 представлены автомат, положение самолета и показания приборов на разных этапах тестовой эмуляции. В процессе тестовой эмуляции фиксировались отклонения от маршрута и рекомендованной скорости, высота полета

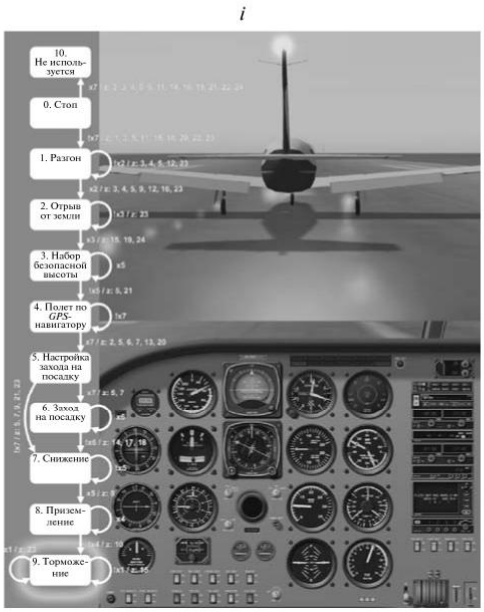


Рис. 15. (Окончание)

(позволяющая определить моменты отрыва от земли и касания земли), действующие перегрузки. Записанные данные отражены на рис. 16.

Из графика следует, что максимальное отклонение самолета от маршрута за все время полета составило 314 м. При этом оно не превышало 60 м на протяжении всего полета, кроме минутного участка в середине графика, когда отклонения наименее критичны. По графику высоты полета можно определить время отрыва и касания земли: 35 и 271 с соответственно. Из установленного времени и графика отклонений от маршрута следует, что максимальное отклонение при разгоне составило 12, а при торможении – 10 м. Существенные, но не выходящие за пределы взлетно-посадочных полос отклонения позволяют считать, что сгенерированный автомат провел самолет по маршруту с достаточной точностью.

Из графика рекомендованной скорости получается, что скоростной режим соблюден за исключением превышения скорости на 20.34 м/с в момент касания земли, которое было компенсировано при торможении и не привело к существенному превышению тормозного пути. Совокупное отклонение положения самолета после остановки, включающее поперечное отклонение от центра полосы и превышение тормозного пути, составляет 6.5 м. Из графика перегрузок следует, что ускоренное торможение не привело к су-

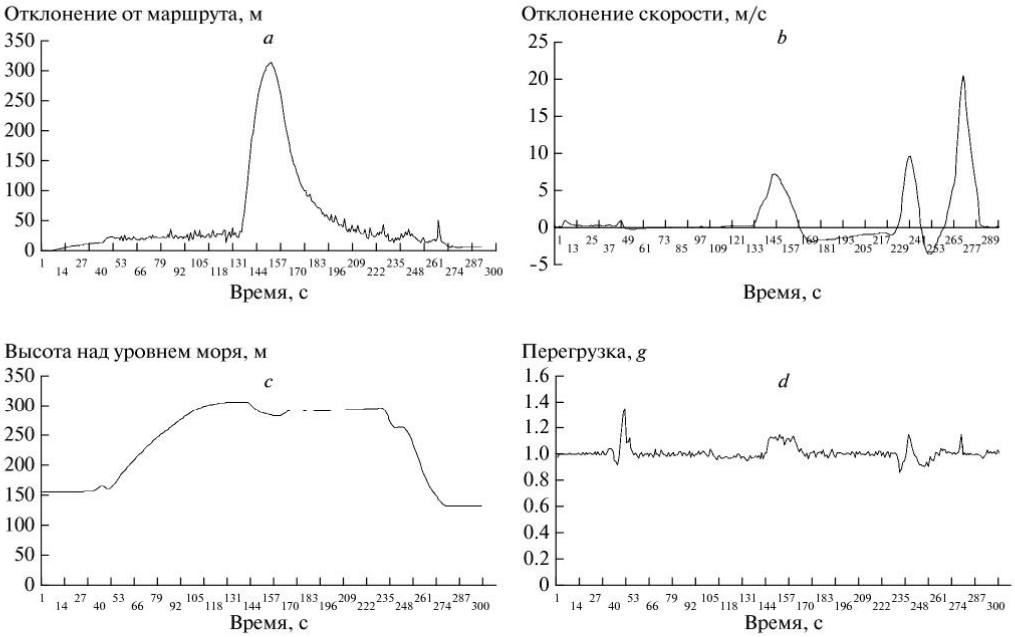


Рис. 16. Результаты тестирования: отклонения от маршрута *a*, отклонения от рекомендованной скорости *b*, высота полета над уровнем моря *c*, перегрузки *d*

щественному дискомфорту пассажиров. Проведенный анализ позволяет сделать заключение об удовлетворительном качестве автоматически построенного автомата.

Заключение. Показаны недостатки существующих методов генерации конечных автоматов с помощью генетических алгоритмов и предложен метод, который не имеет указанных недостатков. Была проведена экспериментальная оценка быстродействия предложенного метода и его требований к памяти. Метод был успешно апробирован в задаче автоматической генерации системы управления самолетом, что показало его применимость для решения практических задач.

СПИСОК ЛИТЕРАТУРЫ

1. *Курейчик В.М.* Генетические алгоритмы. Состояние. проблемы. перспективы // Изв. РАН. ТиСУ. 1999. № 1. С. 144–160.
2. *Курейчик В.В., Курейчик В.М., Сороколетов П.В.* Анализ и обзор моделей эволюции // Изв. РАН. ТиСУ. 2007. № 5. С. 114–126.
3. *Koza J.R.* Genetic Programming: On the Programming of Computers by Means of Natural Selection. The MIT Press., 1992.
4. *Курейчик В.М., Родзин С.И.* Эволюционные алгоритмы: генетическое программирование // Изв. РАН. ТиСУ. 2002. № 1. С. 127–137.
5. *Gold E.M.* Language Identification in the Limit // Information and Control. 1967. № 10. P. 447–474.
6. *Belz A.* Computational Learning of Finite-State Models for Natural Language Processing. PhD thesis. University of Sussex, 2000.
7. *Clelland C.H., Newlands D.A.* Pfsa modelling of behavioural sequences by evolutionary programming // Complex'94 – 2nd Australian Conf. on Complex Systems. Rockhampton, Queensland, Australia, 1994. P. 165–172.
8. *Das S., Mozer M.C.* A Unified Gradient-Descent/Clustering Architecture for Finite State Machine Induction // Advances in Neural Information Processing Systems. 1994. V. 6.
9. *Lankhorst M.M.* A Genetic Algorithm for the Induction of Nondeterministic Pushdown Automata // Computing Science Report. University of Groningen Department of Computing Science, 1995.
10. *Belz A., Eskikaya B.* A genetic algorithm for finite state automata induction with an application to phonotactics // ESSLLI-98 Workshop on Automated Acquisition of Syntax and Parsing. Saarbruecken, Germany, 1998. P. 9–17.
11. *Ashlock D., Wittrock A., Wen T.-J.* Training finite state machines to improve PCR primer design // Congress

on Evolutionary Computation (CEC'02). Honolulu, Hawaii, USA, 2002. P. 13–18.

12. *Ashlock D.A., Emrich S.J., Bryden K.M., et al.* A comparison of evolved finite state classifiers and interpolated markov models for improving PCR primer design // 2004 IEEE Sympos. on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB'04). Jolla, California, USA, 2004. P. 190–197.
13. *Lucas S.M.* Evolving Finite State Transducers: Some Initial Explorations // Genetic Programming: 6th Europ. Conf. (EuroGP'03). Berlin: Springer, 2003. P. 130–141.
14. *Лобанов П.Г., Шалыто А.А.* Использование генетических алгоритмов для автоматического построения конечных автоматов в задаче о флибах // Изв. РАН. ТиСУ. 2007. № 5. С. 127–136.
15. *Царев Ф.Н., Шалыто А.А.* Применение генетических алгоритмов для построения автоматов с минимальным числом состояний для задачи об "умном муравье" // Научно-программное обеспечение в образовании и научных исследованиях. СПб: СПбГУ ПУ, 2008. С. 209–215.
16. *Teller A., Veloso M.* PADO: A New Learning Architecture for Object Recognition. Symbolic Visual Learning. N.Y.: Oxford University Press, 1996. P. 81–116.
17. *Banzhaf W., Nordin P., Keller R.E., et al.* Genetic Programming – An Introduction. On the automatic Evolution of Computer Programs and its Application. San Francisco: Morgan Kaufmann Publishers, 1998.
18. *Kantschik W., Dittrich P., Brameier M.* Empirical Analysis of Different Levels of Meta-Evolution // Congress on Evolutionary Computation. Washington DC, USA, 1999.
19. *Kantschik W., Dittrich P., Brameier M., et al.* Meta-Evolution in Graph GP // Genetic Programming: Second European Workshop (EuroGP'99). Goeteborg, Sweden, 1999.
20. *Teller A., Veloso M.* Internal Reinforcement in a Connectionist Genetic Programming Approach // Artificial Intelligence. North-Holland Pub. Co, 1970. P.161.
21. *Miller J.H.* The Coevolution of Automata in the Repeated Prisoner's Dilemma. Working Paper. Santa Fe Institute, 1989.
22. *Spears W.M., Gordon D.F.* Evolving Finite-State Machine Strategies for Protecting Resources // Internat. Sypos. on Methodologies for Intelligent Systems. Charlotte, North Carolina, USA, 2000.
23. *Ashlock D.* Evolutionary Computation for Modeling and Optimization. N.Y.: Springer, 2006.
24. *Frey C., Leugering G.* Evolving Strategies for Global Optimization. A Finite State Machine Approach // Genetic and Evolutionary Computation Conf. (GECCO-2001). San Francisco, California, USA, 2001. P. 27–33.
25. *Petrovic P.* Simulated evolution of distributed FSA behaviour-based arbitration // The Eighth Scandinavian Conf. on Artificial Intelligence (SCAI'03). Bergen, Norway, 2003.
26. *Petrovic P.* Evolving automatons for distributed behavior arbitration. Technical Report. Trondheim, Norway: Norwegian University of Science and Technology, 2005.
27. *Petrovic P.* Comparing Finite-State Automata Representation with GP-trees. Technical report. Trondheim, Norway: Norwegian University of Science and Technology, 2006.
28. *Поликарпова Н.И., Шалыто А.А.* Автоматное программирование. СПб: Питер, 2009.
29. *Koza J.R.* Future Work and Practical Applications of Genetic Programming. Handbook of Evolutionary Computation. Bristol: IOP Publishing Ltd, 1997.
30. *Поликарпова Н.И., Точилин В.Н., Шалыто А.А.* Применение генетического программирования для реализации систем со сложным поведением // Сб. тр. IV Междунар. научно-практической конф. "Интегрированные модели и мягкие вычисления в искусственном интеллекте". Т. 2. М.: Физматлит, 2007. С. 598–604. http: // is.ifmo.ru/genalg/_polikarpova.pdf.
31. *Поликарпова Н.И., Точилин В.Н., Шалыто А.А.* Разработка библиотеки для генерации управляющих автоматов методом генетического программирования // Сб. докл. X Междунар. конф. по мягким вычислениям и измерениям. Т. 2. СПб: СПбГЭТУ "ЛЭТИ", 2007. С. 84–87. http: // is.ifmo.ru/download/polikarpova(LETI).pdf.
32. http: // www.x-plane.com/.
33. *Халл Э., Джексон К., Дик Д.* Разработка и управление требованиями. Практическое руководство пользователя. 2005.

УДК 004.4'242

МЕТОД ПОСТРОЕНИЯ УПРАВЛЯЮЩИХ КОНЕЧНЫХ АВТОМАТОВ НА ОСНОВЕ ТЕСТОВЫХ ПРИМЕРОВ С ПОМОЩЬЮ ГЕНЕТИЧЕСКОГО ПРОГРАММИРОВАНИЯ

***Ф. Н. Царев*[1],**
аспирант
Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Предлагается метод построения управляющих конечных автоматов на основе тестовых примеров с помощью генетического программирования. Приводятся описания структуры особи алгоритма генетического программирования, операций мутации и скрещивания, а также генетического алгоритма. Предлагаемый метод в качестве примера используется для построения автомата управления часами с будильником.

Ключевые слова *— генетическое программирование, автоматное программирование, машинное обучение.*

Введение

Автоматное программирование — парадигма программирования, в рамках которой программные системы предлагается строить в виде набора взаимодействующих автоматизированных объектов управления [1]. Автоматизированный объект управления состоит из управляющего конечного автомата и объекта управления. Таким образом, поведение каждого автоматизированного объекта управления во многом описывается детерминированным конечным автоматом.

Для большинства задач автоматы удается строить эвристически вручную. Однако в ряде случаев такое построение слишком трудоемко или приводит к неоптимальным результатам. К таким задачам относятся, например, задачи «Умный муравей» [2, 3], «Умный муравей-3» [4] и задача об управлении моделью беспилотного летательного аппарата [5]. Для построения автоматов в таких задачах можно применять генетические алгоритмы (ГА) [6–8].

Традиционный метод построения конечных автоматов с помощью ГА [3, **9–11] использует вы**числение функции приспособленности на основе моделирования работы системы со сложным поведением в некоторой внешней среде. Главным недостатком этого метода является то, что при его применении функцию приспособленности необходимо «с нуля» реализовывать для каждой задачи. Кроме того, такой подход к вычислению функции приспособленности связан с большими затратами вычислительных ресурсов.

Целью настоящей работы является разработка метода построения конечных автоматов на основе генетического программирования, в котором устранены указанные недостатки. Для достижения этой цели предлагается осуществлять построение конечных автоматов на основе тестовых примеров.

Постановка задачи

При применении парадигмы автоматного программирования для реализации сущности со сложным поведением выделяется система управления и объект управления. На начальном этапе проектирования программы выделяются события (e1, e2, ...), входные переменные (x1, x2, ...) и выходные воздействия (z1, z2, ...). После этого проектирование программы может идти разными путями. Один из них состоит в написании сценария работы программы, по которому далее эвристически строится автомат. Пример построения автомата таким способом приведен в работе [12].

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

Другой подход, который практически не применяется для построения автоматных программ, но достаточно широко распространен при традиционной разработке программ, называется «разработкой на основе тестов» (*test-driven development*) [13]. При этом методе процесс написания кода на языке программирования идет параллельно с написанием тестов для программы, а добавление функциональности в программу осуществляется только после того, как создан тест для проверки этой функциональности. Таким образом, функциональность программы описывается набором тестов для нее.

В случае применения автоматного программирования в качестве тестов для управляющего конечного автомата естественно рассматривать пары последовательностей, одна из которых описывает события и входные переменные, поступающие на вход автомату, а вторая — выходные воздействия, которые должен вырабатывать автомат при обработке входных воздействий. Таким образом, задача построения управляющего конечного автомата становится похожей на задачу построения конечного преобразователя, для решения которой успешно используются ГА [14].

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 Tests, каждый из которых содержит последовательность Input[i] событий, поступающих на вход конечному автомату, и соответствующую ей эталонную последовательность Answer[i] выходных воздействий.

Отметим, что при использовании описываемого метода входные переменные явным образом не задаются. Для учета входных переменных необходимо добавить в список событий новые события, объединяющие исходные события и логические формулы, содержащие входные переменные. Например, если в списке событий присутствует событие e1, а в списке входных переменных — x1, то новым событием может быть e1 [x1] (произошло событие e1 и переменная x1 истинна).

Отметим также, что для тестов, которые задаются для построения конечного автомата, справедливо свойство, которое можно сформулировать следующим образом: «префиксы тестов являются тестами» — если из входной последовательности событий удалить часть событий, находящихся в ее конце, то результат обработки автоматом этой последовательности будет префиксом исходной выходной последовательности.

Представление конечного автомата в виде хромосомы ГА. 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. 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 [15].

Выбор представления графа переходов автомата с помощью списков ребер (в отличие от работы [15], в которой применялись полные таблицы переходов) обоснован тем, что, как правило, в автоматах управления системами со сложным поведением не в каждом состоянии определена реакция на каждое событие.

Алгоритм расстановки пометок. Опишем алгоритм расстановки пометок на переходах, применяемый в настоящей работе. Как было сказано выше, для каждого перехода в особи ГА записано, сколько выходных воздействий должно вырабатываться при его выборе. 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.

Для каждого перехода T и каждой последовательности выходных воздействий zs вычисляется величина C[T][zs] — число раз, когда при обработке входной последовательности, соответствующей одному из тестов, на переходе T должны быть выработаны выходные воздействия, образующие последовательность zs. Далее, каждый переход помечается той последовательностью zs_0, для которой величина $C[T][zs_0]$ максимальна.

Функция приспособленности. Функция приспособленности основана на редакционном расстоянии (расстоянии Левенштейна) [16]. Редакционным расстоянием между двумя последовательностями символов называется минимальное число операций замены символа, вставки символа и удаления символа, которые необходимо вы-

полнить над первой последовательностью для того, чтобы она совпала со второй.

Для вычисления функции приспособленности выполняются следующие действия: на вход автомату подается каждая из последовательностей Input[i]. Обозначим последовательность выходных воздействий, которую сгенерировал автомат при входе Input[i], как Output[i]. После этого вычисляется величина

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

где $ED(A, B)$ — редакционное расстояние между строками A и B. Отметим, что значения функции FF_1 лежат в пределах от 0 до 1. При этом, чем «лучше» автомат соответствует тестам, тем больше значение функции приспособленности.

Функция приспособленности зависит не только от того, насколько «хорошо» автомат работает на тестах, но и от числа переходов, которые он содержит. Эта функция вычисляется следующим образом:

$$FF_2 = \begin{cases} 0{,}5 \cdot T \cdot FF_1 + \frac{1}{M}(\text{M} - cnt), FF_1 < 1 \\ T + \frac{1}{M}(M - cnt), FF_1 = 1 \end{cases},$$

где T — «стоимость» прохождения всех тестов; M — произвольное целое число, большее максимального числа переходов в автомате; cnt — число переходов в автомате. При проведении вычислительных экспериментов были выбраны следующие значения: $T = 20$, $M = 100$.

Эта функция приспособленности устроена таким образом, что при одинаковом значении функции FF_1, отражающей «прохождение» тестов автоматом, преимущество имеет автомат, содержащий меньшее число переходов. Кроме этого, автомат, который «идеально» проходит все тесты, оценивается выше, чем автомат, проходящий тесты не идеально.

Операция мутации. При выполнении операции мутации с заданной вероятностью (по умолчанию, она равна 0,05) выполняется каждое из действий:

— изменение начального состояния;

— изменение описания каждого из переходов;

— удаление или добавление перехода для каждого из состояний.

После выполнения операции мутации в автомате может возникнуть ситуация, когда в автомате из одного состояния присутствуют два перехода по одному и тому же событию. Для устранения таких переходов применяется операция удаления дублирующихся переходов.

Операция удаления дублирующихся переходов. В целях удаления дублирующихся переходов для каждого состояния выполняются следующие операции: последовательно просматривается список переходов из этого состояния, при этом запоминаются события, переходы по которым определены для этого состояния. Если очередной переход T происходит по событию, для которого в списке уже есть переход, то переход T удаляется из списка.

Операция скрещивания. Скрещивание описаний автоматов производится следующим образом. Обозначим как P1 и P2 — «родительские» особи, а S1 и S2 — особи-«потомки». Для начальных состояний S1.is и S2.is автоматов S1 и S2 будет верно одно из двух соотношений:

S1.is = P1.is и S2.is = P2.is;

S1.is = P2.is и S2.is = P1.is.

Опишем, как устроены переходы автоматов S1 и S2. Скрещивание описаний автоматов производится отдельно для каждого состояния. Обозначим список переходов из состояния номер i автомата P1 как P1.T[i], а список переходов из состояния номер i автомата P2 как P2.T[i]. Для выполнения «скрещивания переходов» с равной вероятностью может быть выбран один из двух методов.

При использовании *традиционного метода скрещивания* списки переходов S1.T[i] и S2.T[i] строятся следующим образом.

1. Строится общий список переходов, в который помещаются переходы, входящие как в P1.T[i], так и в P2.T[i].

2. К полученному списку применяется случайная перестановка.

3. Далее возможны два равновероятных варианта:

• либо в S1.T[i] помещаются первые |P1.T[i]| переходов из полученного списка, а в S2.T[i] — оставшиеся переходы;

• либо в S1.T[i] помещаются первые |P2.T[i]| переходов из полученного списка, а в S2.T[i] — оставшиеся переходы.

При использовании *метода скрещивания с учетом тестов* списки переходов S1.T[i] и S2.T[i] строятся следующим образом.

1. В автоматах P1 и P2 помечаются те переходы, которые выполняются при обработке 10 % тестов, для которых возрастает нормированное редакционное расстояние между «правильным ответом» Answer и последовательностью Output выходных воздействий, генерируемой автоматом, а значение выражения $\frac{\text{ED}(\text{Output}[i], \text{Answer}[i])}{\max(|\text{Output}[i]|, |\text{Answer}[i]|)}$ минимально.

2. *Помеченные* переходы копируются в S1.*T*[*i*] и S2.*T*[*i*] напрямую.

3. Строится общий список переходов, в который помещаются *непомеченные* переходы, входящие как в P1.*T*[*i*], так и в P2.*T*[*i*].

4. К полученному списку L применяется случайная перестановка.

5. Список S1.*T*[*i*] дополняется первыми переходами из списка L до размера |P1.*T*[*i*]|, а список S2.*T*[*i*] дополняется оставшимися переходами.

В обоих случаях к получившимся в результате скрещивания автоматам S1 и S2 применяется операция удаления дублирующихся переходов.

Пример применения предлагаемого метода

Применение предлагаемого метода иллюстрируется на примере построения автомата управления часами с будильником [1]. Эти часы имеют три кнопки (помеченные буквами «А», «Н», «М»), которые предназначены для изменения режима их работы и для настройки текущего времени или времени срабатывания будильника. Если будильник выключен, то кнопки «Н» и «М» служат для установки текущего времени, а кнопка «А» включает его и переводит часы в режим «Настройка будильника», в котором кнопки «Н» и «М» устанавливают не текущее время, а время срабатывания будильника. Повторное нажатие кнопки «А» включает будильник. После этого если текущее время совпадает со временем срабатывания будильника, то включается звонок, который отключается либо нажатием кнопки «А», либо самопроизвольно через минуту. Кроме этого, нажатие кнопки «А» приводит к выключению будильника.

Рассматриваемые часы с будильником являются системой со сложным поведением, так как в ответ на одни и те же входные события (нажатия кнопок) в зависимости от режима работы генерируются различные выходные воздействия. Поведение этих часов может быть описано с помощью конечного автомата [1], который содержит три состояния (рис. 1).

Система управления часами с будильником имеет четыре события:

- H — нажата кнопка «Н»;
- M — нажата кнопка «М»;
- A — нажата кнопка «А»;
- T — генерируется таймером каждую секунду.

Кроме этого, она содержит две входные переменные:

- x1 — верно ли, что текущее время совпадает со временем срабатывания будильника;
- x2 — верно ли, что текущее время на минуту больше времени срабатывания будильника?

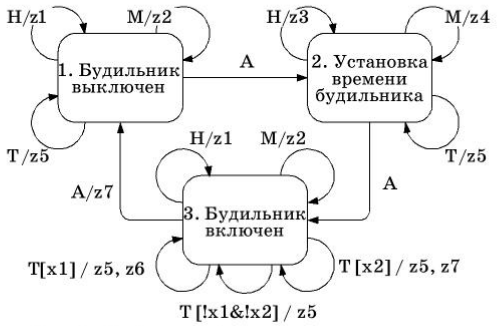


***Рис. 1.** Граф переходов автомата управления часами с будильником*

Число выходных воздействий равно семи:

- z1 — увеличить на единицу число часов в текущем времени;
- z2 — увеличить на единицу число минут в текущем времени;
- z3 — увеличить на единицу число часов во времени срабатывания будильника;
- z4 — увеличить на единицу число минут во времени срабатывания будильника;
- z5 — прибавить минуту к текущему времени;
- z6 — включить звонок будильника;
- z7 — выключить звонок будильника.

В систему тестов для построения автомата управления часами с будильником было составлено 38 тестов, описывающих его работу во всех трех режимах. В качестве примера приведем тесты для состояния «Будильник выключен» (таблица).

Построение конечного автомата управления часами с будильником проводилось при следующих параметрах алгоритма генетического программирования:

- размер поколения — 2000 особей;
- доля «элиты» — наиболее приспособленных особей, напрямую переходящих в следующее поколение, — 10 %;
- число поколений до малой «мутации поколения» — 100 поколений;
- число поколений до большой «мутации поколения» — 150 поколений;
- размер автоматов в начальном поколении — четыре состояния.

Было проведено 1000 запусков алгоритма с указанными параметрами. Цель в каждом из них состояла в том, чтобы построить автомат, содержащий 14 переходов и соответствующий всем тестам (значение функции приспособленности, соответствующее такому автомату, — 20.86). На каждом из запусков алгоритма генетического программирования был построен автомат (рис. 2), в котором из начального (нулевого) достижимы

Тесты для состояния «Будильник выключен»

	Входная последовательность	Выходная последовательность	Комментарий
1	*T, T, T, T*	z5, z5, z5, z5	Описывает обработку часами события «Сработал таймер». При возникновении этого события текущее время должно быть увеличено на минуту
2	*H, H, H, H*	z1, z1, z1, z1	Описывает обработку часами нажатия кнопки «H». При нажатии на эту кнопку число часов в текущем времени должно быть увеличено на единицу
3	*M, M, M, M*	z2, z2, z2, z2	Описывает обработку часами нажатия кнопки «M». При нажатии на эту кнопку число минут в текущем времени должно быть увеличено на единицу
4	*T, M, H, T, T, T, M, T, H, H, T, M*	z5, z2, z1, z5, z5, z5, z2, z5, z1, z1, z5, z2	Описывает обработку событий H, M и T в состоянии «Будильник выключен»
5	*A, A, A, T, T, T, T*	z7, z5, z5, z5, z5	После трех нажатий кнопки «A» часы должны находиться в состоянии «Будильник выключен». Аналог первого теста
6	*A, A, A, H, H, H, H*	z7, z1, z1, z1, z1	После трех нажатий кнопки «A» часы должны находиться в состоянии «Будильник выключен». Аналог второго теста
7	*A, A, A, M, M, M, M*	z7, z2, z2, z2, z2	После трех нажатий кнопки «A» часы должны находиться в состоянии «Будильник выключен». Аналог третьего теста

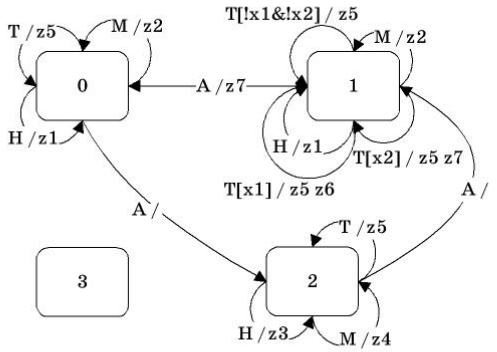


***Рис. 2.** Граф переходов автомата, построенного с помощью алгоритма генетического программирования*

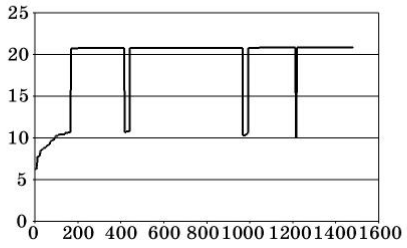


***Рис. 3.** Зависимость максимального значения функции приспособленности от номера поколения*

только три состояния из четырех. Если удалить недостижимое состояние, то этот граф переходов будет изоморфен графу переходов, построенному вручную.

График зависимости максимального значения функции приспособленности от номера поколения представлен на рис. 3 при одном из запусков алгоритма.

Как видно из графика, автоматы, входящие в начальное поколение, проходят тесты примерно наполовину. Примерно к двухсотому поколению был построен автомат, полностью проходящий все тесты, однако содержащий достаточно большое число переходов. Далее шел процесс уменьшения числа переходов, во время которого три раза (в районе 400-, 1000- и 1200-го поколений) к популяции применялась операция «большой мутации», в результате чего значение функции приспособленности уменьшалось примерно до десяти. В итоге в 1482-м поколении был построен автомат, полностью проходящий все тесты и содержащий 14 переходов.

Для каждого из запусков запоминалось число вычислений функции приспособленности (которое равно числу просмотренных во время работы авто-

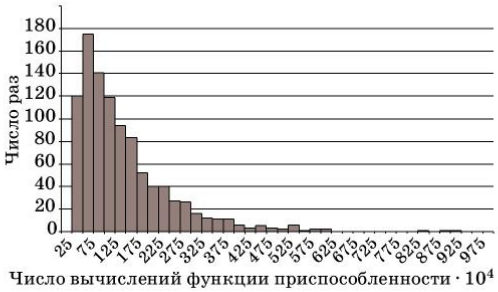


***Рис. 4.** Распределение числа вычислений функции приспособленности в проведенных вычислительных экспериментах*

матов) в процессе построения автомата. На рис. 4 показано распределение этой величины, полученное в результате вычислительных экспериментов.

Минимальное значение числа вычислений функции приспособленности составило 256 063, максимальное — 9 239 523. В предположении, что эта величина распределена по логнормальному распределению, ее среднее значение составляет 1 443 351.20 (стандартное отклонение — 1 103 401.82).

Заключение

В работе предложен метод построения автоматов управления системами со сложным поведением с учетом тестов с помощью генетического программирования. При использовании разработанного метода для построения управляющих конечных автоматов необходимо задать только тестовые примеры, а вычисление функции приспособленности требует существенно меньше вычислительных ресурсов, чем при использовании моделирования для вычисления функции приспособленности. Рассмотрен пример применения разработанного метода.

Исследования проводятся по государственному контракту, выполняемому в рамках Федеральной целевой программы «Научные и научно-педагогические кадры инновационной России на 2009–2013 годы».

Литература

1. **Поликарпова Н. И., Шалыто А. А.** Автоматное программирование. — СПб.: Питер, 2010. — 176 с.
2. **Jefferson D. et. al.** The Genesys System: Evolution as a Theme in Artificial Life // Proc. of Second Conf. on Artificial Life. MA: Addison-Wesley, 1992. P. 549–578. www.cs.ucla.edu/~dyer/Papers/AlifeTracker/Alife91Jefferson.html (дата обращения: 19.03.2010).
3. **Царев Ф. Н., Шалыто А. А.** Применение генетического программирования для генерации автомата в задаче об «Умном муравье» // Интегрированные модели и мягкие вычисления в искусственном интеллекте: Сб. тр. IV Междунар. науч.-практ. конф. Т. 2. М.: Физматлит. 2007. С. 590–597. http://is.ifmo.ru/genalg/_ant_ga.pdf (дата обращения: 19.03.2010).
4. **Бедный Ю. Д., Шалыто А. А.** Применение генетических алгоритмов для построения автоматов в задаче «Умный муравей». http://is.ifmo.ru/works/ant (дата обращения: 19.03.2010).
5. **Паращенко Д. А., Царев Ф. Н., Шалыто А. А.** Технология моделирования одного класса мультиагентных систем на основе автоматного программирования на примере игры «Соревнование летающих тарелок»: Проектная документация / СПбГУ ИТМО. 2006. http://is.ifmo.ru/unimod-projects/plates (дата обращения: 19.03.2010).
6. **Гладков Л. А., Курейчик В. В., Курейчик В. М.** Генетические алгоритмы. — М.: Физматлит, 2006. — 320 с.
7. **Рассел С., Норвиг П.** Искусственный интеллект: современный подход. — М.: Вильямс, 2006. — 1407 с.
8. **Koza J. R.** Genetic programming: on the programming of computers by means of natural selection. — MIT Press, 1992. — 835 с.
9. **Поликарпова Н. И., Точилин В. Н., Шалыто А. А.** Применение генетического программирования для генерации автоматов с большим числом входных переменных // Науч.-техн. вестник СПбГУ ИТМО. 2008. Вып. 53. Автоматное программирование. С. 24–42.
10. **Данилов В. Р.** Метод представления автоматов деревьями решений для использования в генетическом программировании // Науч.-техн. вестник СПбГУ ИТМО. 2008. Вып. 53. Автоматное программирование. С. 103–108.
11. **Давыдов А. А., Соколов Д. О., Царев Ф. Н.** Применение генетических алгоритмов для построения автоматов Мура и систем взаимодействующих автоматов Мили на примере задачи об «Умном муравье» // Науч.-техн. вестник СПбГУ ИТМО. 2008. Вып. 53. Автоматное программирование. С. 108–114.
12. **Мазин М. А., Парфенов В. Г., Шалыто А. А.** Разработка интерактивных приложений Macromedia Flash на базе автоматной технологии: Проектная документация / СПбГУ ИТМО. 2003. http://is.ifmo.ru/projects/flash/ (дата обращения: 19.03.2010).
13. **Бек К.** Экстремальное программирование: разработка через тестирование. — СПб.: Питер, 2003. — 224 с.
14. **Lucas S., Reynolds T.** Learning Finite State Transducers: Evolution versus Heuristic State Merging // IEEE Transactions on Evolutionary Computation. 2007. Vol. 11. Is. 3. P. 308–325.
15. **Lucas S., Reynolds T.** Learning Deterministic Finite Automata with a Smart State Labeling Evolutionary Algorithm // IEEE Transactions on Evolutionary Computation. 2005. Vol. 27. Is. 7. P. 1063 –1074.
16. **Левенштейн В. И.** Двоичные коды с исправлением выпадений, вставок и замещений символов // Докл. Академии наук СССР. 1963. № 4. С. 845–848.

ПРИЛОЖЕНИЕ 4. 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 №П1188 от 27 августа 2009 года по проекту «Разработка методов совместного применения генетического и автоматного программирования для построения систем управления беспилотными летательными аппаратами»
(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 _____ / Новиков В.В. /