Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

На правах рукописи

Царев Федор Николаевич

Методы построения конечных автоматов на основе эволюционных алгоритмов

Специальность 05.13.11 – Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Диссертация на соискание ученой степени
кандидата технических наук

Научный руководитель:
доктор технических наук,
профессор А. А. Шалыто

Санкт-Петербург – 2012 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность проблемы. В последнее время при разработке программного обеспечения (ПО) для управляющих систем все шире применяется *автоматное программирование* [22, 27] – парадигма программирования, при использовании которой программу предлагается строить как совокупность *автоматизированных объектов управления*, каждый из которых содержит *систему управления* (один или несколько взаимодействующих *управляющих конечных автоматов*) и *объект управления*.

Объект управления характеризуется множеством ***вычислительных состояний***, а также двумя наборами функций: множеством *предикатов* (они же являются *входными переменными* или *входными воздействиями* автомата), отображающих вычислительное состояние в логическое значение (истина или ложь), и множеством *воздействий*, позволяющих изменять вычислительное состояние, которые являются *выходными воздействиями* автомата. Кроме этого, внешняя среда может генерировать *события*, которые поступают на вход управляющего конечного автомата.

Управляющий конечный автомат определяется конечным множеством ***управляющих состояний***, *функцией переходов* и *функцией* выходных воздействий (*выходов*). Функция переходов зависит от события, набора значений входных переменных и текущего состояния автомата. Значением этой функции является номер состояния, в которое должен перейти автомат. Функция выходных воздействий в общем случае также зависит от события, набора значений входных переменных и текущего состояния автомата. Значением этой функции является набор выходных воздействий на объект управления.

Главное отличие управляющих конечных автоматов от других типов конечных автоматов (конечных преобразователей и распознавателей)

состоит в том, что в них в пометки переходов входят не отдельные входные воздействия, а *булевы формулы* из них. Именно такие автоматы и рассматриваются в настоящей диссертации.

Взаимодействие между автоматами может осуществляться различными способами: за счет вложенности автоматов, с помощью обмена сообщениями, с помощью обмена номерами состояний и т. д.

При использовании автоматного программирования существенно *упрощается* (по сравнению с программами, написанными традиционными методами) *верификация программ* с использованием метода *Model checking* [6, 9, 125, 56], так как построение модели Крипке по автоматной программе может быть автоматизировано [125]. Кроме этого, при использовании инструментальных средств для поддержки автоматного программирования таких, как, например, *UniMod* [3], до 70% исходного кода автоматной программы может быть сгенерировано автоматически [27, 137]. Уровень автоматизации программирования этого класса программ станет значительно выше, если удастся автоматизировать процесс построения управляющих конечных автоматов, что и является предметом исследования в настоящей работе.

В последнее десятилетие активно развивается область исследований, называемая *поисковая инженерия ПО* (*Search-Based Software Engineering*, *SBSE*) [45, 69 – 71], в рамках которой для решения задач *программной инженерии* (включая анализ требований [37, 121], прогнозирование хода разработки [30], проектирование [110], тестирование [34, 67, 93] и *рефакторинг* [68, 106]) предлагается применять *алгоритмы поисковой оптимизации*. В число методов, которые нашли применение в поисковой инженерии ПО [69], входят *эволюционные алгоритмы* [7] (*генетические алгоритмы* [2, 11, 12, 43], *генетическое программирование* [84] и *эволюционные стратегии* [40]), а также *муравьиные алгоритмы* [49], *метод роя частиц* [82], *метод имитации отжига* [94], *метод спуска* [66],

алгоритмы оценки распределений [86], *поиск с запретами* [58], *меметические алгоритмы* [51], *метод рассеянного поиска* [59], *квадратичное программирование* [105], *целочисленное программирование* [113], *искусственные иммунные системы* [47]. При этом в настоящее время наибольшее распространение получили эволюционные алгоритмы [71].

Как указано выше, при использовании автоматного программирования возникает задача, для решения которой можно, а иногда и необходимо, применять методы поисковой инженерии ПО. Это определяется тем, что построение управляющих конечных автоматов вручную может представлять существенную сложность, а в ряде случаев построить автоматы вручную и вовсе не удается.

Примерами таких задач являются: управление командой беспилотных летательных аппаратом в соревнованиях с другой командой [21, 136], итерированная дилемма узника [98], задача о «Флибах» [1, 53], задача «Умный муравей» [79]. Полный перебор управляющих конечных автоматов даже при их небольших размерах крайне трудоемок, а эвристическое их построение, как отмечено выше, не всегда дает приемлемые результаты.

Разработка методов решения указанной задачи является одним из шагов к автоматическому построению программ и позволит повысить уровень автоматизации построения автоматных программ (как отмечалось выше, до 70% исходного кода программ может быть построено автоматически) и снизить влияние человеческого фактора на их качество.

Как отмечается в работе [69], б*о*льшая часть работ в области поисковой инженерии ПО основана на использовании эволюционных алгоритмов, а для решения задач проектирования ПО (к которым относится задача построения конечных автоматов) применялись только эволюционные и муравьиные алгоритмы, методы имитации отжига и

спуска. Так как метод имитации отжига не дает существенного улучшения результатов по сравнению с генетическими алгоритмами [8, 124], а муравьиные алгоритмы более приспособлены для задач, в которых решением является путь в графе, то для решения задачи построения конечных автоматов обычно применялись эволюционные алгоритмы и метод спуска.

Подобные идеи возникали у ряда исследователей. В 1962 г. Л. Фогель занялся изучением интеллектуального поведения индивида и его развития в процессе эволюции [52]. При этом поведение индивида задавалось конечным автоматом. Продолжая данные исследования, Л. Фогель, А. Оуэнс и М. Уолш предложили в 1966 г. схему эволюции конечных автоматов, решающих задачи предсказания [53].

При решении задачи каким-либо из методов поисковой оптимизации необходимо описать задачу в терминах *множества допустимых решений* (пространства поиска) и *функции приспособленности*. Для задачи построения управляющих конечных автоматов множеством допустимых решений является множество автоматов с заданными событиями, входными переменными и выходными воздействиями и числом состояний не больше заданного. Функция приспособленности зависит от задачи, которую должен решать автомат, и должна характеризовать качество ее решения.

Генетические алгоритмы ведут поиск оптимальных решений параллельно в нескольких точках пространства поиска. Вначале случайным образом генерируется некоторое число решений (особей), образующих начальное *поколение*. Далее, особи этой популяции *скрещиваются* и *мутируют,* формируя новое поколение. Скрещивание (*кроссовер, рекомбинация*) – фундаментальная операция в генетических алгоритмах, позволяющая производить обмен «генетическим материалом» между особями. Мутация – не менее важная составляющая, она позволяет

получать новый «генетический материал», а также предотвращать «застревание» в области локального оптимума.

В классическом генетическом алгоритме особь кодируется строкой над небольшим алфавитом (как правило, это битовая строка), по аналогии с хромосомой, кодирующей наследственную информацию в живых организмах. По этой причине особь генетического алгоритма также называют хромосомой. Для битовых строк известно несколько стандартных операций скрещивания и мутации, что, однако, не ограничивает возможные варианты выбора этих операций.

Функция приспособленности выражает насколько решение, представленное данной особью, удовлетворяет задаче. Она может также содержать дополнительные слагаемые, выражающие, например, штраф за слишком большое число переходов в автомате. Эти слагаемые используются для направления процесса поиска оптимального решения.

Особи нового поколения выбирается на основе критерия отбора. Этот критерий отдает предпочтение более приспособленным особям, в то же время, давая шанс и менее приспособленным. Таким образом, в популяции поддерживается необходимое разнообразие особей, в то же время лучшие особи выживают гораздо чаще. Можно сказать, что идея генетического алгоритма берет свои истоки в учении об естественном отборе. Однако, эволюция Дарвина в некотором смысле «слепа» – улучшения в генотипе популяции происходят случайно. В генетических же алгоритмах улучшение особей в популяции является основной целью.

Генетическое программирование – разновидность генетических алгоритмов, в которой вместо низкоуровневого представления объектов в виде битовых строк используется высокоуровневое представление: деревья разбора программ, диаграммы переходов конечных автоматов и т. д. С помощью генетического программирования эффективно решаются задачи

автоматического построения программ, конечных автоматов и клеточных автоматов.

Эволюционные стратегии имеют два отличия от генетических алгоритмов: в них *не используется операция скрещивания* и, как правило, *каждое поколение состоит только из одной особи*. Процесс поиска начинается со случайно выбранной особи, задающей некоторое допустимое решение задачи. На каждой итерации к текущей особи применяется операция мутации. Она обычно устроена таким образом, чтобы в результате могло получиться (пусть и с весьма малой вероятностью) любое допустимое решение.

Если результат этой операции представляет собой лучшее решение, то оно становится текущим на следующей итерации. Работа алгоритма считается оконченной, когда будет достигнуто необходимое значение функции приспособленности.

Метод спуска устроен следующим образом. Процесс поиска начинается со случайно выбранной точки в пространстве решений задачи. На каждой итерации рассматриваются точки, соседние с текущей. Если некоторая соседняя точка представляет собой улучшенное решение, то она становится текущей на следующей итерации. В противном случае поиск считается оконченным, а текущая точка – оптимальным решением.

В предыдущих работах, связанных с автоматным программированием, рассматривались, прежде всего, методы, использующие **моделирование для вычисления функции приспособленности** [5, 14, 15, 17, 35, 53, 73, 76, 79, 95, 112, 114, 129 – 132, 135].

Эти методы обладают следующими недостатками:

- для решения новой задачи необходимо заново выполнять программную реализацию вычисления функции приспособленности, что может быть весьма трудоемким;

- каждое вычисление функции приспособленности может требовать значительного времени, так как связано с моделированием поведения управляющего конечного автомата в некоторой внешней среде.

Поэтому исследования, направленные на разработку лишенных указанных недостатков эволюционных алгоритмов для построения *управляющих* конечных автоматов, **являются актуальными.**

В соответствии с паспортом специальности 05.13.11 – «Математическое обеспечение вычислительных машин, комплексов и компьютерных сетей» областями исследования, к которым относится настоящая диссертация, в частности, являются «5. Теория построения программ, пакетов прикладных программ (ППП), программных комплексов (ПК), а также сетевых программ (СП), в том числе, поддерживающих сетевые протоколы» и «9. Модели и методы разработки программных средств обработки данных и знаний в ВМ, ВК и КС».

Цель диссертационной работы – разработка методов построения *управляющих* конечных автоматов (в дальнейшем автоматов) на основе эволюционных алгоритмов.

Основные задачи диссертационной работы состоят в следующем:

1. Разработать метод построения автоматов *по обучающим примерам* на основе эволюционных алгоритмов.
2. Разработать метод выполнения операции скрещивания для генетических алгоритмов, учитывающий поведение автоматов на обучающих примерах.
3. Разработать метод построения автоматов по обучающим примерам и темпоральным формулам на основе эволюционных алгоритмов и верификации.
4. Разработать технологию построения автоматов по обучающим примерам и темпоральным формулам.

5. Разработать инструментальное средство для автоматизации построения автоматов.
6. Внедрить разработанные методы при построении автомата управления моделью беспилотного самолета и в учебный процесс.

Научная новизна. В работе получены следующие новые научные результаты, которые выносятся на защиту:

1. Метод построения автоматов по обучающим примерам на основе эволюционных алгоритмов. Его основное отличие от известных состоит в том, что в предлагаемые алгоритмы добавлен новый шаг «Расстановка выходных воздействий», который выполняется перед вычислением функции приспособленности.
2. Метод выполнения операции скрещивания для генетических алгоритмов, учитывающий поведение автоматов на обучающих примерах. Показано, что генетический алгоритм, использующий разработанный метод выполнения операции скрещивания, осуществляет построение автоматов по обучающим примерам быстрее, чем генетический алгоритм, использующий традиционный метод выполнения операции скрещивания, эволюционная стратегия и метод спуска на основе случайных мутаций.
3. Метод построения автоматов по обучающим примерам и темпоральным формулам на основе эволюционных алгоритмов и верификации. Его основное отличие от известных состоит в том, что для вычисления функции приспособленности совместно применяются обучающие примеры и метод *Model Checking*.

Методы исследования. В работе используются методы теории автоматов, дискретной математики и эволюционных алгоритмов.

Достоверность научных положений, выводов и практических рекомендаций, полученных в диссертации, подтверждается корректным обоснованием постановок задач, точной формулировкой критериев, а также результатами экспериментов по использованию предложенных в диссертации методов.

Практическое значение работы состоит в том, что разработана технология автоматизированного построения управляющих конечных автоматов на основе эволюционных алгоритмов и создано инструментальное средство для автоматизации построения автоматов, поддерживающее эту технологию. По постренным автоматам, как отмечено выше, автоматически может быть сгенерирован программный код. Предложенные в работе эволюционные алгоритмы позволяют решить задачи построения автоматов, которые не удается решить вручную, а для других автоматов – существенно сократить затраты времени на их построение по сравнению с известными методами, что подтверждается результатами экспериментальных исследований, приведенными в работе.

Внедрение результатов работы. Результаты диссертации были получены при выполнении научно-исследовательских работ на кафедрах «Компьютерные технологии» и «Технологии программирования» НИУ ИТМО по следующим государственным контрактам: «*Технология генетического программирования для генерации автоматов управления системами со сложным поведением*» (государственный контракт № 02.514.11.4044 от 18.05.2007 г. по Федеральной целевой программе «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007–2013 годы»), «*Разработка методов совместного применения генетического и автоматного программирования для построения систем управления*

беспилотными летательными аппаратами» (государственный контракт № П1188 от 27.08.2009 г. по Федеральной целевой программе «Научные и научно-педагогические кадры инновационной России» на 2009–2013 годы), «*Разработка методов машинного обучения на основе генетических алгоритмов для построения управляющих конечных автоматов*» (государственный контракт № П2174 от 09.11.2009 г. по Федеральной целевой программе «Научные и научно-педагогические кадры инновационной России» на 2009–2013 годы), «*Применение методов искусственного интеллекта в разработке управляющих программных систем*» (государственный контракт № П2236 от 11.11.2009 г. по Федеральной целевой программе «Научные и научно-педагогические кадры инновационной России» на 2009–2013 годы), в рамках проекта «Подготовка и переподготовка профильных специалистов на базе центров образования и разработок в сфере информационных технологий в Северо-Западном федеральном округе» (Государственный контракт № 07.Р20.11.0028 от 07.09.2011 г.) и были внедрены на примере построения автомата управления моделью беспилотного самолета, а также в учебном процессе на кафедре «Компьютерные технологии» в рамках курса «Теория автоматов и программирование».

Апробация результатов работы. Основные положения диссертационной работы докладывались на следующих научных и научно-практических конференциях: IV Международная научно-практическая конференция «Интегрированные модели и мягкие вычисления в искусственном интеллекте» (Коломна, 2007), научно-техническая конференция «Научное программное обеспечение в образовании и научных исследованиях» (СПбГПУ, 2008), XII Всероссийская конференция по проблемам науки и высшей школы «Фундаментальные исследования и инновации в технических университетах» (СПбГПУ, 2008), Second Spring Young Researchers' Colloquium on Software Engineering –

SYRCoSE'2008 (SPbSU, 2008), III и IV Международная научно-практическая конференция «Современные информационные технологии и ИТ-образование» (ВМК МГУ, 2008, 2009), научно-практическая конференция студентов, аспирантов, молодых ученых и специалистов «Интегрированные модели, мягкие вычисления, вероятностные системы и комплексы программ в искусственном интеллекте (ИММВИИ-2009)» (Коломна, 2009), VI и VII межвузовская конференция молодых ученых (СПбГУ ИТМО, 2009, 2010), X, XI и XII Международная конференции по мягким вычислениям и измерениям (СПбГЭТУ (ЛЭТИ), 2009, 2010, 2011), Международная научная конференция «Компьютерные науки и информационные технологии» памяти А. М. Богомолова (Саратовский государственный университет имени Н. Г. Чернышевского, 2009), 32-я конференция молодых ученых и специалистов Института проблем передачи информации им. А. А. Харкевича РАН «Информационные технологии и системы» (2009), XL научная и учебно-методическая конференция профессорско-преподавательского и научного состава СПбГУ ИТМО (2011), 14-th Annual Graduate Students Workshop (part of the «Genetic and Evolutionary Computation Conference» GECCO – 2011, Dublin, ACM SIGEVO, 2011), вторая межвузовская научная конференция по проблемам информатики (СПИСОК, СПбГУ, 2011), XLI научная и учебно-методическая конференция НИУ ИТМО (2012), 15-th Annual Graduate Students Workshop (part of the «Genetic and Evolutionary Computation Conference» GECCO – 2012, Philadelphia, ACM SIGEVO, 2012), третья российская конференция с международным участием «Технические и программные средства систем управления, контроля и измерения» (Институт проблем управления имени В. А. Трапезникова РАН, 2012) – пленарный доклад.

Публикации. По теме диссертации опубликованы 23 печатные работы, в том числе шесть статей, из которых пять в журналах из перечня ВАК.

Свидетельства о регистрации программ для ЭВМ. В рамках диссертационной работы получены три свидетельства о регистрации программ для ЭВМ: № 2010 614197 от 29.06.2010 г. «Программное средство для построения управляющих конечных автоматов на основе обучающих примеров с использованием генетических алгоритмов», № 2011 615664 от 19.07.2011 г. «Программное средство для генерации конечных автоматов с дискретными и непрерывными выходными воздействиями», № 2011 616977 от 08.09.2011 г. «Виртуальная лаборатория для обучения методам искусственного интеллекта при построении конечных автоматов».

Структура диссертации. Диссертация изложена на 196 страницах и состоит из введения, четырех глав, заключения и двух приложений. Список источников содержит 170 наименований. Работа проиллюстрирована 41 рисунком и 14 таблицами.

В первой главе приводится обзор работ, посвященных автоматному программированию, поисковой инженерии ПО и применению эволюционных алгоритмов для построения автоматов. На основе результатов обзора формулируются задачи, решаемые в настоящей диссертации.

Вторая глава посвящена методам построения автоматов по обучающим примерам на основе эволюционных алгоритмов и выполнения операции скрещивания с учетом поведения автоматов на обучающих примерах, а также построения управляющих автоматов на основе эволюционных алгоритмов по обучающим примерам и темпоральным формулам. Рассматриваются следующие эволюционные алгоритмы: метод спуска на основе случайных мутаций, эволюционная стратегия,

генетический алгоритм. Описывается представление конечных автоматов в эволюционных алгоритмах, алгоритмы выполнения операций мутации и скрещивания, метод вычисления функции приспособленности. Приводятся результаты вычислительных экспериментов по построению автоматов на примерах задач построения автомата управления часами с будильником и автомата управления дверьми лифта, а также для тестов, сгенерированных случайным образом.

В третьей главе описываются разработанные автором технология построения автоматов на основе эволюционных алгоритмов и инструментальное средство для автоматизированного построения автоматов.

Четвертая глава посвящена результатам внедрения предложенных методов на примере построения автомата управления моделью беспилотного самолета и в учебный процесс.

В заключении сформулированы результаты, полученные в диссертации.

ГЛАВА 1. АВТОМАТНОЕ ПРОГРАММИРОВАНИЕ И ПОИСКОВАЯ ИНЖЕНЕРИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В настоящей главе приводятся результаты обзора работ, посвященных автоматному программированию, поисковой инженерии ПО и применению эволюционных алгоритмов для построения конечных автоматов. На основании результатов обзора формулируются задачи, решаемые в диссертации.

1.1. Автоматное программирование

В настоящем разделе приводятся результаты обзора работ, посвященных автоматному программированию.

1.1.1. Сущности со сложным поведением

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

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

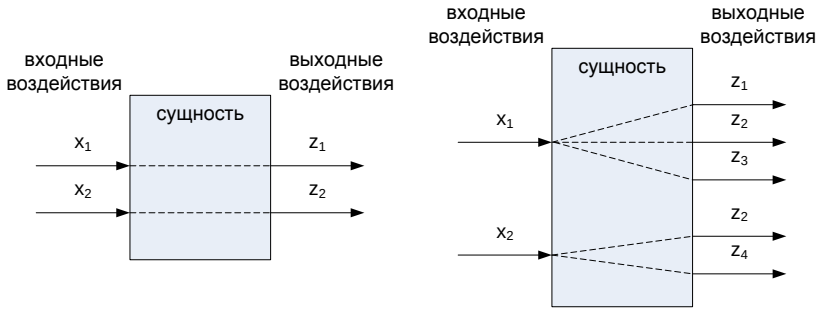


Рис. 1. Сущность с простым поведением и сущность со сложным поведением

При традиционной программной реализации сущностей с таким поведением используются переменные, называемые *флагами*. Их роль – участвовать в конструкциях ветвления, реализующих логику поведения. Флаги неявно задают отдельные компоненты состояний. Использование флагов трудно для понимания, подвержено ошибкам и практически не расширяемо.

Вместо этого в автоматном программировании предлагается [18] описывать системы со сложным поведением, приписывая каждой из них некоторое множество *управляющих состояний* (в дальнейшем, если не оговаривается особо, состояний). В этих состояниях поведение системы является простым и может быть описано явно. Связь управляющих состояний с выходными воздействиями и механизм переходов между состояниями удобно описывать с помощью *конечных автоматов* с выходами [26]. При этом все описание поведения системы оказывается сосредоточенным в автомате или системе взаимодействующих автоматов.

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

Если в системах логического управления [28] в качестве входных воздействий используются опрашиваемые программой двоичные входные переменные и предикаты, соответствующие определенным состояниям автоматов, взаимодействующих с рассматриваемым автоматом, то для «реактивных» систем это понятие расширено. Во-первых, в качестве

входных переменных применяются любые подпрограммы (функции), возвращающие двоичные значения, а, во-вторых, введены события, не только обеспечивающие возможность выполнения переходов в автоматах, но и инициирующие запуск автоматов. События могут также инициировать реализацию выходных воздействий в случае, когда состояние автомата не изменяется.

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

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

1.1.2. Парадигма автоматного программирования

Автоматное программирование, предложенное в работе [22], – это парадигма программирования, которая состоит в представлении сущностей со сложным поведением в виде *автоматизированных объектов управления*.

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

Базовым понятием автоматного программирования является «состояние». Это понятие в том смысле, как оно используется в описываемой парадигме, было введено А. Тьюрингом. Основное свойство

состояния системы в любой момент времени состоит в том, что оно несет в себе всю информацию о прошлом системы, необходимую для определения ее реакции на любое входное воздействие, формируемое в текущий момент времени. Состояние можно рассматривать как особую характеристику, которая в неявной форме объединяет все входные воздействия прошлого, влияющие на реакцию сущности в настоящий момент времени. Реакция системы зависит только от входного воздействия и текущего состояния.

Состояния системы, как отмечалось выше, могут быть разделены на *управляющие* и *вычислительные*. Если говорить неформально, то основные отличия между ними можно сформулировать следующим образом:

- число управляющих состояний не очень велико, а число вычислительных состояний либо очень велико, либо бесконечно;
- управляющие состояния отличаются друг от друга качественно, а управляющие – количественно;
- управляющие состояния определяют совершаемые выходные воздействия, а вычислительные – их результаты.

Понятие «входное воздействие» также является одним из базовых для автоматного программирования. Чаще всего, входное воздействие содержит событие и значения входных переменных. Совокупность конечного множества состояний и конечного множества входных воздействий образует *автомат без выходов*. Такой автомат реагирует на входные воздействия, определенным образом изменяя текущее состояние. Правила, по которым происходит смена состояний, задаются функцией переходов автомата.

То, что в автоматном программировании собственно и называется автоматом (рис. 2), получается, если соединить понятие автомата без выходов с понятием «выходное воздействие». Такой автомат реагирует на входное воздействие не только сменой состояния, но и формированием

определенных значений на выходах. Правила формирования выходных воздействий называют функцией выходных воздействий (выходов) втомата.

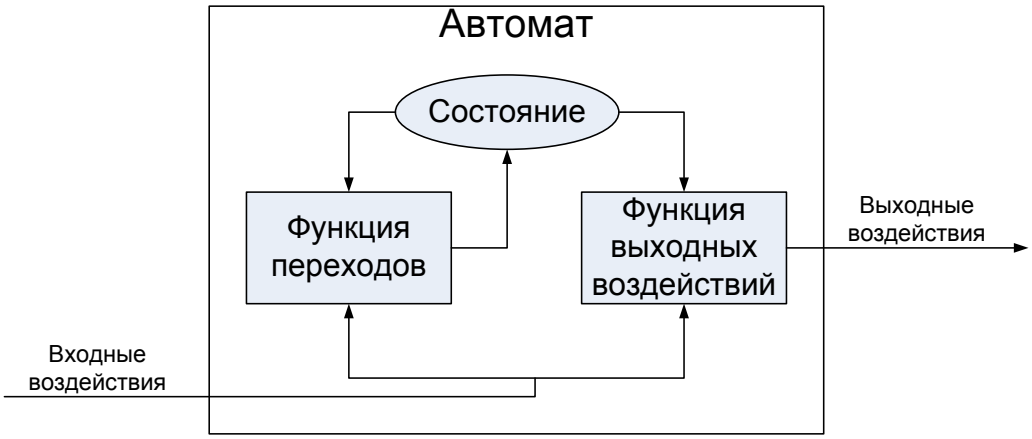


Рис. 2. Автомат

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

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

В соответствии с традицией теории управления, управляемая часть называется *объект управления*, а управляющая часть – *система управления*. Поскольку для реализации управляющей части используются конечные автоматы, то каждый из них называется *управляющий конечный автомат* или просто *автомат*.

После разделения сущности со сложным поведением на объект управления и автомат реализовать ее уже несложно, а главное, ее

реализация становится понятной и удобной для модификации. Вся логика поведения сущности сосредоточена в автоматах. Объект управления обычно обладает простым поведением (а, следовательно, может быть легко реализован традиционными «неавтоматными» методами). Он не обрабатывает непосредственно входные воздействия от внешней среды, а только получает от автоматов команды совершить те или иные действия. При этом каждая команда всегда вызывает одно и то же действие (это и есть определение простого поведения).

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

При построении модели автоматизированного объекта предполагается, что управляющий автомат взаимодействует и с объектом управления, и с внешней средой (рис. 3).

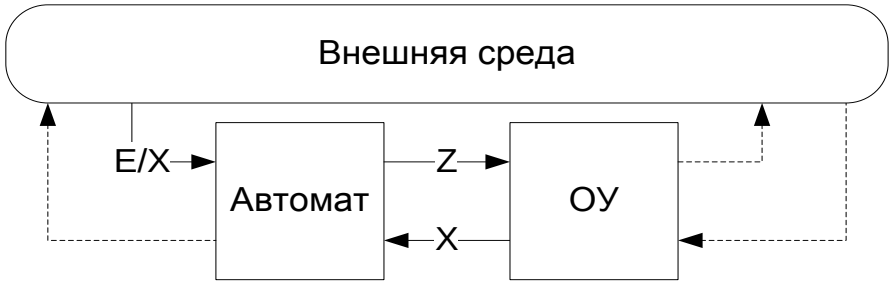


Рис. 3. Взаимодействие компонентов модели автоматизированного объекта

На этом рисунке сплошными стрелками обозначены традиционные и наиболее типичные для программных реализаций виды взаимодействия между автоматом, объектом управления и внешней средой. Автомат получает входные воздействия, как со стороны внешней среды, так и от объекта управления. В событийных системах часть или все компоненты входного воздействия со стороны среды могут быть событиями (множество событий обозначено на рисунке буквой *E*). Входное

воздействие со стороны объекта управления формирует в модели обратную связь (от управляемого объекта к управляющему). Это воздействие может отсутствовать, тогда модель является *разомкнутой* – так в теории управления называются системы управления без обратной связи. В противном случае модель называется *замкнутой*. Автомат, в свою очередь, воздействует на объект управления.

Пунктирными стрелками обозначены менее распространенные, хотя и возможные, варианты взаимодействия. Так, автомат может оказывать выходное воздействие и на внешнюю среду. Однако таких связей обычно можно избежать, включив все управляемые автоматом сущности в состав его объекта управления. Отметим, что в программировании, в общем случае, различие между объектом управления и внешней средой носит скорее концептуальный, а не формальный характер. Создавая модель системы со сложным поведением, разработчик производит ее декомпозицию на *автоматизированные объекты*, определяя тем самым объект управления для каждого автомата. В целях минимизации связей между модулями программной системы целесообразно проводить декомпозицию таким образом, чтобы автомат оказывал выходные воздействия только на собственный объект управления.

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

В программировании на вид входных и выходных воздействий нет ограничений: это могут быть символы, числа, строки, множества, последовательности, произвольные объекты – все зависит от специфики поставленной задачи и инструментов, используемых для ее решения. Кроме того, могут различаться способы передачи входных воздействий автомату и интерпретации выходных воздействий в объекте управления.

Если по назначению сущность близка к традиционной системе управления, то представление входных и выходных воздействий битовыми

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

1.1.3. Управляющий конечный автомат

Управляющий конечный автомат представляет собой набор $< X, E, Y, Z, y_0, \phi, \delta >$, где X – множество входных переменных, E – множество событий, Y – множество состояний, y_0 – начальное состояние, $\phi: E \times 2^X \times Y \to Z^*$ – функция выходных воздействий, $\delta: E \times 2^X \times Y \to Y$ – функция переходов.

Главное отличие управляющих конечных автоматов от других типов конечных автоматов (*конечных преобразователей* и *распознавателей*) состоит в том, что в них в пометки переходов входят не отдельные входные воздействия, а *булевы формулы* из них. Именно такие автоматы и рассматриваются в дальнейшнм в настоящей диссертации.

Приведем пример управляющего конечного автомата – автомата управления часами с будильником [22]. Эти часы имеют три кнопки (рис. 4), которые предназначены для изменения режима их работы и для настройки текущего времени или времени срабатывания будильника.

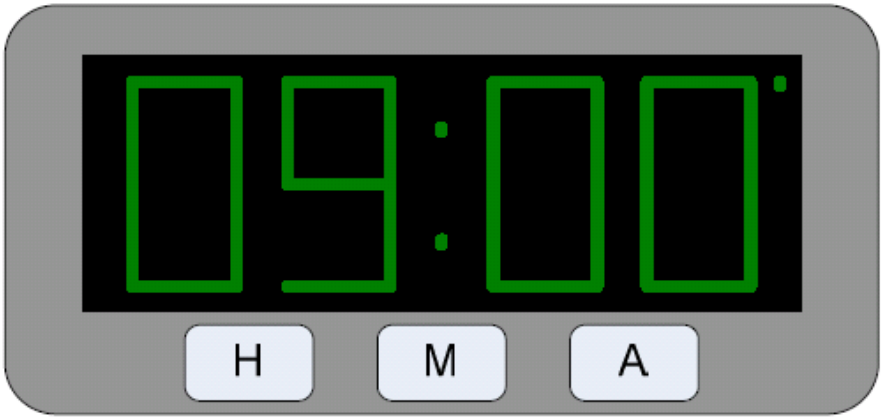


Рис. 4. Внешний вид часов с будильником

Если будильник выключен, то кнопки «H» и «M» предназначены, соответственно, для увеличения на единицу числа часов и минут в текущем времени. Кнопка «A» в этом режиме служит для перехода в режим настройки времени срабатывания будильника. В этом режиме кнопки «H» и «M» позволяют увеличивать на единицу число часов и минут во времени срабатывания будильника. Нажатие кнопки «A» в этом режиме приводит к включению будильника. Он срабатывает, как только время срабатывания совпадает с текущим временем. Звонок автоматически выключается через минуту или может быть выключен нажатием кнопки «A», которая также выключает будильник. Кроме кнопок часы содержат таймер, который срабатывает каждую минуту – при каждом его срабатывании текущее время увеличивается на одну минуту.

Отметим, что рассматриваемые часы 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.

Эта система имеет *четыре события*:

- *H* – нажата кнопка «H» на корпусе часов;
- *M* – нажата кнопка «M» на корпусе часов;
- *A* – нажата кнопка «A» на корпусе часов;

- T – сработал таймер.

Она также содержит *две входные переменные*:

- x_1 – верно ли, что время срабатывания будильника совпадает с текущим временем?
- x_2 – верно ли, что текущее время превышает время срабатывания будильника ровно на одну минуту?

Система имеет *семь выходных воздействий*:

- z_1 – увеличить на единицу число часов текущего времени;
- z_2 – увеличить на единицу число минут текущего времени;
- z_3 – увеличить на единицу число часов времени срабатывания будильника;
- z_4 – увеличить на единицу число минут времени срабатывания будильника;
- z_5 – прибавить минуту к текущему времени;
- z_6 – включить звонок будильника;
- z_7 – выключить звонок будильника.

На рис. 5 приведен граф переходов автомата управления часами с будильником из работы [22].

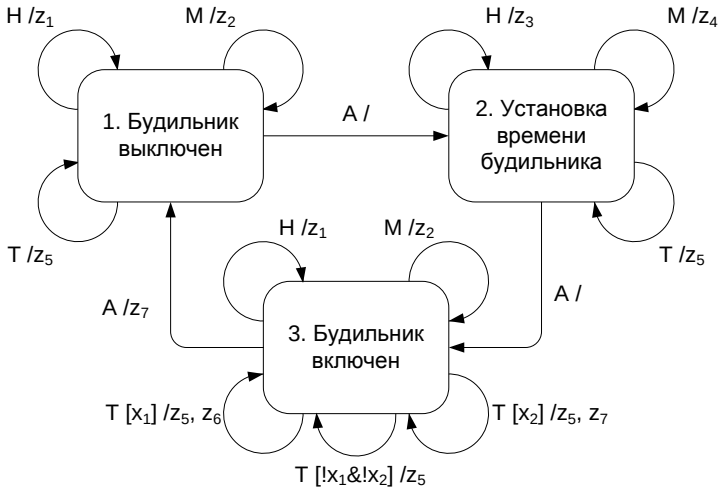


Рис. 5. Граф переходов конечного автомата управления часами с будильником

Начальным состоянием этого автомата является состояние «1. Будильник выключен». На приведенном графе переходов каждый переход *t* помечен событием *e*, логической формулой *f* относительно входных переменных (она выражает условие перехода и приведена в квадратных скобках) и последовательностью *o* выходных воздействий (приведена после косой черты). Переход *t* выполняется в случае поступления события *e*, если при подстановке значений входных переменных в формулу *f* ее значение есть «истина». При выполнении этого перехода генерируются выходные воздействия, составляющие последовательность *o*, а автомат переходит в состояние, в которое ведет переход *t*.

От графа переходов автомата обычно требуют выполнения двух свойств – полноты и непротиворечивости. *Полнота* означает, что в каждом состоянии для любого события и любой комбинации значений входных переменных задан хотя бы один переход. *Непротиворечивость* означает, нет двух переходов из одного состояния, которые бы выполнялись при одном и том же событии и одном и том же наборе значений входных переменных.

Отметим, что приведенный на рис. 5 автомат обладает свойством полноты, но не обладает свойством непротиворечивости, так как в состоянии «3. Будильник включен» при поступлении события Т и одновременной истинности входных переменных x_1 и x_2 задано два перехода.

Если этот граф переходов исправить, как показано на рис. 6, то получится автомат, обладающий свойством непротиворечивости, но не обладающий свойством полноты, так как в этом случае не задано ни одного перехода из состояния «3. Будильник включен» при поступлении события Т и одновременной истинности входных переменных x_1 и x_2.

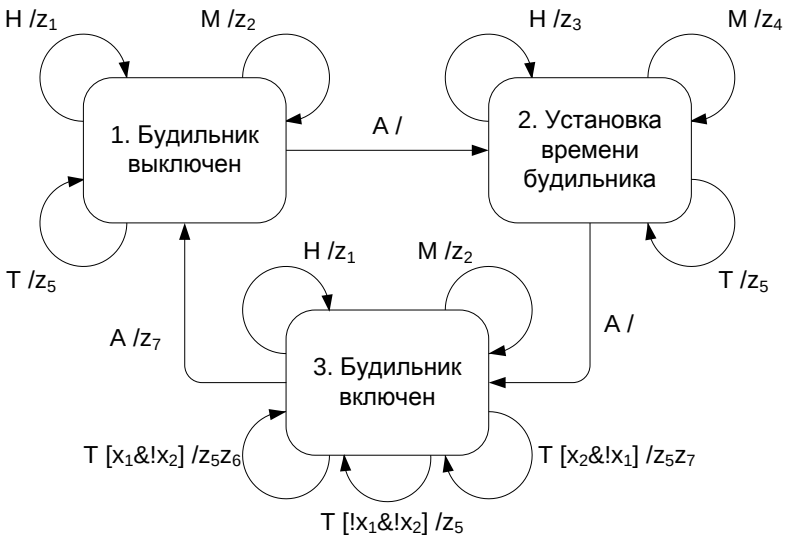


Рис. 6. Исправленный граф переходов конечного автомата управления часами с будильником

При этом отметим, что оба автомата корректно описывают поведение системы управления часами с будильником, так как из семантики входных переменных следует, что x_1 и x_2 не могут быть истинными одновременно. Это означает, что даже если некоторый управляющий конечный автомат формально не обладает непротиворечивостью и полнотой переходов, то он все равно может оказаться приемлемым для управления сущностью со сложным поведением.

В дальнейшем в диссертации будет рассматриваться задача построения автоматов, обладающих свойством непротиворечивости.

1.1.4. Верификация автоматных программ на основе метода *Model Checking*

Метод проверки того, что программная система удовлетворяет определенным требованиям, называется *верификацией*. Наиболее применимым на практике методом верификации в настоящее время является верификация моделей программ (*Model Checking*) [9, 73]. При использовании этого метода процесс верификации состоит из трех этапов.

Первый из них (моделирование программы) состоит в преобразовании программы в модель Крипке [9], содержащую конечное число состояний. Второй этап (спецификация) – это формальная запись утверждений, которые требуется проверить. На третьем этапе выполняется собственно верификация – алгоритмическая проверка выполнения спецификации для модели.

Сложность применения такого подхода для написанных традиционным образом программ (без явного выделения состояний) состоит в том, что после построения модели и ее верификации в случае обнаружения контрпримера необходимо преобразование контрпримера из терминов модели в термины программы. Кроме того, не всегда корректность модели означает соответствие программы спецификации, так как при построении модели выполняется переход на другой уровень абстракции, теряя определенные данные и связи в программе. Данный процесс представлен на рис. 7.

Рис. 7. Процесс верификации программы

При использовании автоматного программирования проблемы построения модели по программе и преобразования контрпримеров из терминов модели в термины программы не возникают, так как конечный автомат может быть автоматически преобразован в модель, пригодную для верификации, а контрпримеры могут быть автоматически преобразованы из терминов модели в термины автомата [4, 125 – 128]. Эта *особенность автоматных программ* упрощает их верификацию по сравнению с программами, написанными традиционным путем.

Для верификации автоматных программ в работах [6, 125 – 128] было разработано несколько инструментальных средств. Некоторые из них использовали существующие верификаторы (например, *SPIN* [76], *SMV* [92], *Bogor* [143]) и осуществляли только преобразование в их входной язык, а инструментальное средство, предложенное в работе [6], использует собственную реализацию алгоритма верификации. Именно оно и использовалось в настоящей работе.

1.1.4.1. Логика линейного времени

В указанном инструментальном средстве для описания утверждений, верификация которых проводится, используется логика линейного времени (*Linear Time Logic*, *LTL*) [9]. Синтаксис *LTL* включает в себя предикаты *P*, булевы связки (! – логическое «НЕ», & – логическое «И», || – логическое «ИЛИ») и темпоральные операторы, применяющиеся для составления утверждений о событиях в будущем.

В этой логике допустимы следующие темпоральные операторы:

- X (neXt) – 《X p》 верно тогда, когда в следующий момент времени в программе будет выполняться предикат p;
- G (Globally) – 《G p》 верно, если во время работы программы всегда выполняется p;

- F (Future) – «F p» верно, если в будущем наступит момент, когда выполнится p;
- U (Until) – «p U q» верно, если в программе в каждый момент времени выполняется p до тех пор, пока не выполнится q. При этом q обязательно должно когда-либо выполниться;
- R (Release) – «q R p» верно, если p выполняется до тех пор, пока не станет выполняться q (включая момент, когда выполнится q), или всегда, если q не выполнится никогда

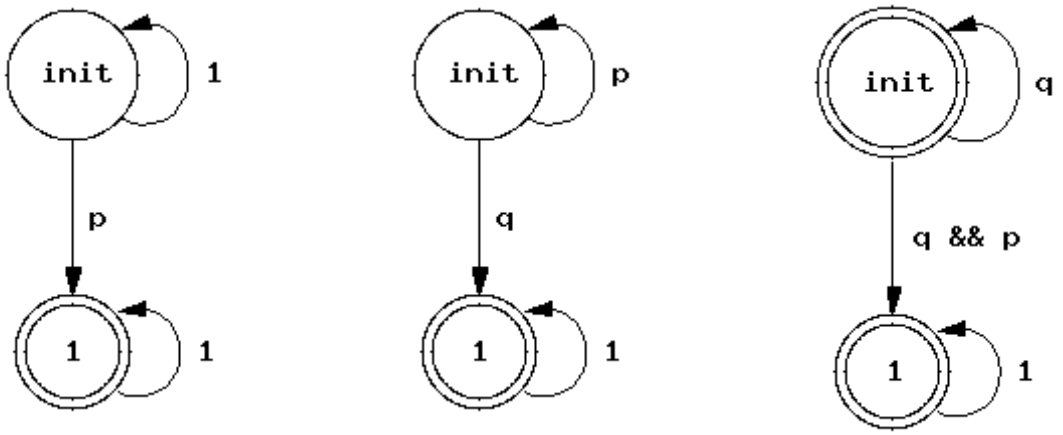
Множество *LTL*-формул таково:

- предикаты *P*;
- *True, False*;
- если φ и ψ – формулы, то
  - $!\varphi$, $\varphi\&\psi$, $\varphi||\psi$ – формулы;
  - $X\varphi$, $F\varphi$, $G\varphi$, $\varphi U\psi$, $\varphi R\psi$ – формулы.

Для проведения верификации *LTL*-формула, описывающая требования к программе, преобразуется в *автомат Бюхи* [9] – конечный автомат над бесконечными словами. Переходы автомата Бюхи помечаются предикатами из исходной *LTL*-формулы. Методы построения автомата Бюхи по *LTL*-формуле описаны в работах [9, 44, 56].

Так как задача верификатора – найти контрпример, если он существует, то автомат Бюхи строится для отрицания исходной *LTL*-формулы. Такой автомат допускает любые последовательности значений предикатов, которые не удовлетворяют требованиям.

Приведем несколько примеров автоматов Бюхи, построенных по *LTL*-формулам. На рис. 8 приведены автоматы, построенные для темпоральных операторов **F**uture, **U**ntil и **R**elease. При этом как «init» обозначены начальные состояния, а состояния, отмеченные двойным кругом, являются допускающими.



а) б) в)

Рис. 8. Автоматы Бюхи для *LTL*-формул Fp (а), pUq (б), pRq (в)

Далее модель Крипке, построенная для исходной программы, также преобразуется в автомат Бюхи, в котором метка на переходе – это выполнимость определенного предиката. Для автоматных программ под предикатом будем понимать утверждение о текущем переходе, например, сгенерированные автоматом выходные воздействия в объекте управления или состояние, в которое перешел автомат.

После этого строится его пересечение с автоматом Бюхи, построенным по отрицанию *LTL*-формулы. Это пересечение также является автоматом Бюхи, и для него запускается алгоритм двойного поиска в глубину [9], который находит допускающую последовательность предикатов. Если эта последовательность существует, то:

- она допускается автоматом Бюхи, построенным по модели Крипке. Следовательно, эта последовательность является историей работы исходной программы;
- последовательность допускается автоматом Бюхи, построенным из отрицания *LTL*-формулы. Следовательно, эта последовательность является историей, нарушающей проверяемые требования.

1.2. Поисковая инженерия программного обеспечения

В настоящем разделе приводятся результаты обзора работ, посвященных поисковой инженерии ПО.

1.2.1. Основные понятия

В рамках поисковой инженерии ПО [45, 69 – 71] для решения задач программной инженерии таких, как анализ требований [37, 121], прогнозирование хода разработки [30], проектирование [110], верификация на основе метода *Model Checking* [31 – 33, 60], тестирование [34, 67, 93], разбиение на модули [91] и рефакторинг [68, 106], предлагается применять алгоритмы поисковой оптимизации.

Отметим, что эта область исследований является новой – на апрель 2012 г. по этой тематике было опубликовано всего 1020 работ (http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/repository.html).

В число методов, которые нашли применение в поисковой инженерии ПО [69], входят эволюционные алгоритмы (генетические алгоритмы [2, 11, 12, 43, 24, 25], генетическое программирование [84] и эволюционные стратегии [40]), а также муравьиные алгоритмы [49], метод роя частиц [82], метод имитации отжига [94], метод спуска [66], алгоритмы оценки распределений [86], поиск с запретами [58], меметические алгоритмы [51], метод рассеянного поиска [59], квадратичное программирование [105], целочисленное программирование [113], искусственные иммунные системы [47]. При этом наибольшее распространение получили эволюционные алгоритмы [71].

В работе [45] приводятся следующие критерии, которые позволяют определить, насколько задача подходит для решения с помощью методов поисковой оптимизации:

- Большой размер пространства поиска. Если пространство поиска достаточно мало, для того чтобы все допустимые решения можно было перебрать за относительно небольшое

время, то нет смысла использовать методы поисковой оптимизации.

- Не существует известных эффективных решений данной задачи. Действительно, не имеет смысла применять новые, заведомо менее эффективные алгоритмы для решения уже решенной задачи. Однако если существующие решения покрывают лишь часть возможных входных данных, применение алгоритмов поисковой оптимизации может помочь для того, чтобы решить задачу для оставшихся вариантов входных данных.
- Существование подходящей функции приспособленности. В частности, при разработке ПО существует множество метрик, применимых для этой цели [72].
- Приемлемое время для генерации новых потенциальных решений, а также для вычисления функции приспособленности.

Общие рекомендации по применению алгоритмов поисковой оптимизации для решения задач инженерии ПО сформулированы в работах [45, 69] следующим образом:

1. Сначала необходимо определить, достаточно ли данная задача сложна, для того чтобы решать ее методами поисковой оптимизации (пространство поиска является чрезмерно большим, для того чтобы использовать на нем традиционные алгоритмы, используемые в задачах такого рода).
2. Далее необходимо разработать метод представления потенциальных решений, позволяющий использовать алгоритмы поисковой оптимизации, и задать функцию приспособленности.

3. Применение методов поисковой оптимизации следует начать с метода спуска. Если он дает достаточно хорошие результаты, то имеет смысл применять и другие алгоритмы поисковой оптимизации.

Последний пункт объясняется тем, что во многих задачах метод спуска часто дает удовлетворительные результаты [66, 83, 96, 97]. При этом существуют как задачи [119], в которых генетический алгоритм работает лучше метода спуска, так и задачи, в которых метод спуска работает лучше генетического алгоритма [100].

В любом случае, результаты применения метода спуска могут служить начальной точкой для анализа задачи. Если применение метода спуска не дает хороших результатов, это может означать, что задача проанализирована недостаточно тщательно или используемое представление является неэффективным. Возможно, это также означает, что данная задача не может быть удовлетворительно решена алгоритмами поисковой оптимизации в целом.

Как отмечается в работе [69], для решения задач проектирования ПО, к которым относится задача построения автоматов, применялись следующие алгоритмы: метод спуска, метод имитации отжига [94], муравьиные алгоритмы, эволюционные алгоритмы. Как отмечалось выше, в литературе для построения автоматов применяются в основном эволюционные алгоритмы, которые улучшаются в настоящей работе и рассматриваются ниже.

1.2.2. Метод спуска

Наиболее простым из алгоритмов решения задач поисковой оптимизации является *метод спуска* (англ. *hill climbing*). Процесс поиска начинается со случайно выбранной точки в пространстве решений задачи. На каждой итерации рассматриваются точки, соседние с текущей. Если некоторая из этих точек представляет собой улучшенное решение по

сравнению с текущей, то она становится текущей на следующей итерации. В противном случае поиск считается оконченным, а текущая точка – оптимальным решением.

Главной проблемой метода спуска является то, что найденное решение является локальным оптимумом, который может оказаться гораздо хуже глобального.

У метода спуска существует множество вариаций. Например, вопрос о том, требуется ли выбирать первую соседнюю точку, оказавшуюся в указанном смысле лучше текущей, или требуется ли рассмотреть все соседние точки и выбрать среди них оптимальный вариант, порождает различные версии алгоритма (*first ascent hill climbing*, *steepest ascent hill climbing*).

Еще одним вариантом метода спуска является метод спуска на основе случайных мутаций (*random mutation hill climbing*), который *единственный из всех вариаций метода спуска* ***относится к эволюционным алгоритмам***. В этом методе к текущему решению на каждом шаге применяется случайное локальное изменение (мутация), и полученное решение сравнивается с текущим. Если оно не хуже его, то оно становится текущим. Если в течение некоторого числа итераций не происходит улучшения значения функции приспособленности, то происходит перезапуск алгоритма (рис. 9).

Этот процесс продолжается до тех пор, пока не будет выполнен критерий остановки алгоритма (достигнуто определенное значение функции приспособленности, с момента запуска алгоритма прошел определенный промежуток времени и т. д.).

Метод спуска является достаточно простым в реализации алгоритмом, и во многих случаях применение этого метода дает достаточно хорошие результаты, например, в задаче разбиения на модули [66, 97] и оценки стоимости проекта [83].

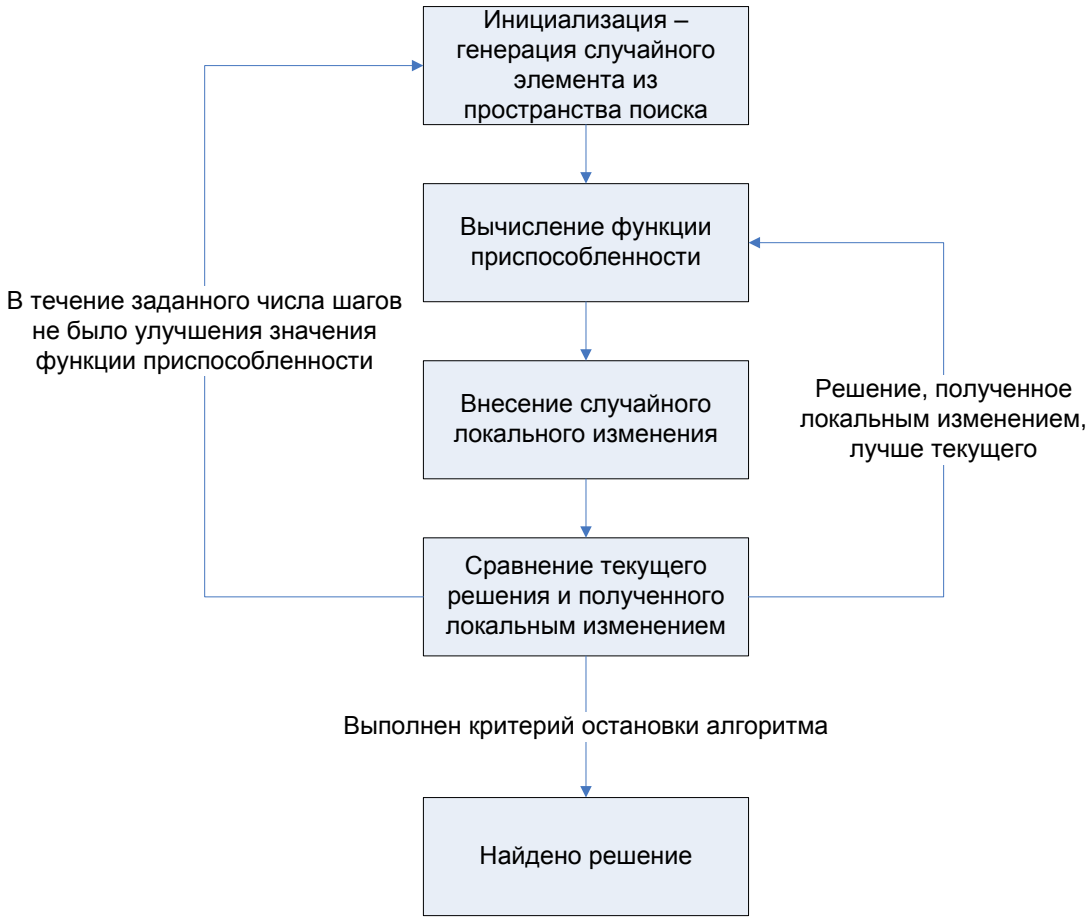


Рис. 9. Схема работы метода спуска на основе случайных мутаций

1.2.3. Эволюционная стратегия

Опишем (1+1)-*эволюционную стратегию* [40]. Процесс поиска начинается со случайно выбранной точки в пространстве решений задачи. На каждой итерации к текущему решению применяется мутация, и полученное решение сравнивается с текущим – если оно не хуже его, то оно становится текущим. При этом в отличие от метода спуска мутация не является локальной – в ее результате может получить любой элемент пространства решений (пусть и с небольшой вероятностью).

Если в течение некоторого числа итераций не происходит улучшения значения функции приспособленности, то аналогично методу спуска происходит перезапуск алгоритма (рис. 10). Этот процесс продолжается до тех пор, пока не будет выполнен критерий остановки алгоритма (достигнуто определенное значение функции приспособленности, с

момента запуска алгоритма прошел определенный промежуток времени и т. д.).

Рис. 10. Схема работы (1+1)-эволюционной стратегии

Кроме (1+1)-эволюционных стратегий известны также (1+λ)-эволюционные стратегии, в которых на каждой итерации с помощью мутаций генерируется λ новых решений, из которых выбирается лучшее. Известны и (μ+λ)-эволюционные стратегии, в которых рассматривается не одно текущее решение, а μ решений, из которых на каждой итерации с помощью мутаций генерируются λ новых решений. Также существуют варианты эволюционных стратегий, использующие операцию скрещивания [40], что делает их во многом похожими на генетические алгоритмы.

1.2.4. Генетические алгоритмы

Генетический алгоритм (*genetic algorithm*) [36, 73] представляет собой метод оптимизации, основная идея которого состоит в использовании принципа *естественного отбора*, составляющего основы теории эволюции живых организмов, предложенной Ч. Дарвином.

Подобные идеи возникали у ряда исследователей. В 1962 г. Л. Фогель занялся изучением интеллектуального поведения индивида и его развития в процессе эволюции [52]. Такое поведение задавалось конечным автоматом. Продолжая данные исследования, Л. Фогель, А. Оуэнс и М. Уолш предложили в 1966 г. схему эволюции конечных автоматов, решающих задачи предсказания [53]. Примерно в это же время (в середине 60-х годов) Дж. Холланд разработал новый метод поиска оптимальных решений – генетические алгоритмы. Результатом работы Дж. Холланда стала книга [73], вышедшая в 1975 г. Эти работы заложили основы эволюционных вычислений.

Кратко сформулировать принцип естественного отбора можно следующим образом: наименее приспособленные особи умирают раньше, а наиболее приспособленные выживают и дают потомство. Потомство выживших особей оказывается в среднем более приспособленным, но среди них опять выделяются более приспособленные особи, и т. д.

В генетических алгоритмах используются те же принципы. В качестве особей выступают элементы пространства возможных решений некоторой задачи (маршруты коммивояжера, графы переходов автомата и т. п.). Задан набор генетических операций, с помощью которых из существующих особей формируются новые. Кроме этого определена так называемая *функция приспособленности* (*fitness function*), которая показывает насколько «хорошим» решением задачи является особь.

Генетические алгоритмы, в отличие от изложенных ранее алгоритмов, ведут поиск оптимальных решений сразу в нескольких точках

пространства поиска. Вначале случайным образом генерируется некоторое число решений (особей, в терминологии генетических алгоритмов), образующих начальное *поколение*. Далее, особи этой популяции *скрещиваются* и *мутируют*, формируя новое поколение. Скрещивание (*кроссовер*, *рекомбинация*) – фундаментальная операция в генетических алгоритмах, позволяющая обмен генетическим материалом между особями. Мутация – не менее важная составляющая, она позволяет получать новый генетический материал, а также предотвращать алгоритм от «застревания» в области локального оптимума.

Процесс работы генетического алгоритма состоит в генерации поколений особей до тех пор, пока не будет выполнено некоторое условие останова (например, достигнуто целевое значение функции приспособленности или сгенерировано заданное число поколений). На рис. 11 приведена общая схема работы генетического алгоритма.

В *классическом генетическом алгоритме особь кодируется строкой* над небольшим алфавитом (как правило, это битовая строка), по аналогии с хромосомой, кодирующей наследственную информацию в живых организмах. По этой причине особь генетического алгоритма нередко также называют хромосомой. Для битовых строк известно несколько стандартных операций скрещивания и мутации, что, однако, не ограничивает возможные варианты выбора этих операций [20].

Генетическое программирование (*genetic programming*), предложенное Дж. Козой в 1992 г. [84], предполагает применение генетических алгоритмов для автоматизированного построения программ.

Основным отличием генетического программирования от традиционных генетических алгоритмов является способ представления особей в виде деревьев разбора программ.

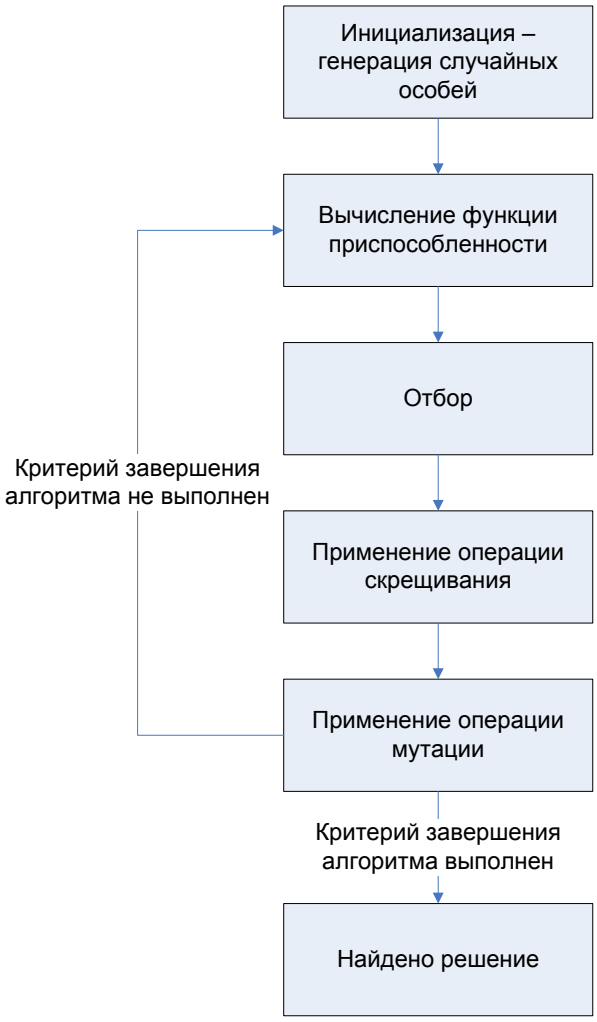


Рис. 11. Общая схема работы генетического алгоритма

1.3. Применение эволюционных алгоритмов для построения конечных автоматов

В настоящем разделе приводится обзор существующих эволюционных алгоритмов построения конечных автоматов. Методы построения конечных автоматов с помощью эволюционных алгоритмов и генетического программирования можно разделить на три типа:

- методы, использующие моделирование для вычисления функции приспособленности;
- методы, использующие обучающие примеры (тесты) для вычисления функции приспособленности;

- методы, использующие верификацию для вычисления функции приспособленности.

1.3.1. Методы, использующие моделирование при вычислении функции приспособленности

В настоящем разделе описываются эволюционные алгоритмы, осуществляющие построение конечных автоматов и использующие моделирование при вычислении функции приспособленности.

1.3.1.1. Эксперименты Фогеля

Один из создателей эволюционного программирования Л.Фогель рассматривал интеллектуальное поведение индивида как способность успешно предсказывать поведение среды, в которой он находится, и сообразно с этим действовать. В 60-х годах прошлого века годах Л. Фогель поставил ряд экспериментов [53] по созданию искусственных систем, способных адаптироваться к первоначально неизвестной им среде.

В проведенных экспериментах Л. Фогель моделировал поведение простейшего живого существа, которое способно предсказывать изменения параметра среды, обладающего *периодичностью*. Это существо моделировалось конечным автоматом с действиями на переходах – автоматом Мили. В качестве среды выступала последовательность символов над двоичным алфавитом, например, $(1111010010)^*$. На вход автомата в каждый момент времени подавалось битовое значение параметра окружающей среды. Автомат формировал значение выходной переменной – возможное значение рассматриваемого параметра в следующий момент времени. На рис. 12 приведен один из возможных автоматов, который построен для распознавания указанной выше последовательности.

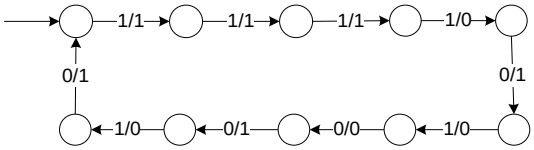


Рис. 12. Граф переходов эвристически построенного автомата

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

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

- добавление состояния;
- удаление состояния (в случае, если число состояний больше единицы);
- замена начального состояния (в случае, если число состояний больше единицы);

- замена перехода;
- замена действия на переходе.

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

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

1.3.1.2. Задача «Умный муравей»

Еще одной задачей, в которой построенные с помощью генетических алгоритмов автоматы обладают подобными свойствами, является задача «Умный муравей» [35, 79]. В этой задаче рассматривается поле, имеющее форму тора размером 32 на 32 клетки (рис. 13). В некоторых клетках поля расположены яблоки – черные клетки на рис. 13. Яблоки расположены вдоль некоторой ломаной линии, но не во всех ее клетках. Клетки ломаной, на которых яблок нет – серые. Белые клетки не принадлежат ломаной и не содержат яблок. Всего на поле 89 яблок.

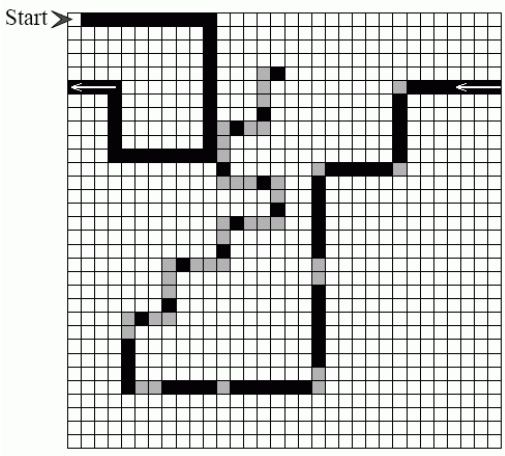


Рис. 13. Поле с яблоками

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

- идет вперед на одну клетку, съедая яблоко, если оно находится в этой клетке;
- поворачивает вправо;
- поворачивает влево;
- стоит на месте.

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

Поведение муравья может быть задано конечным автоматом. Например, конечный автомат, граф переходов которого изображен на рис. 14, содержит пять состояний и описывает стратегию «Вижу яблоко –

иду вперед. Не вижу – поворачиваю. Сделал круг, но яблок нет – иду вперед».

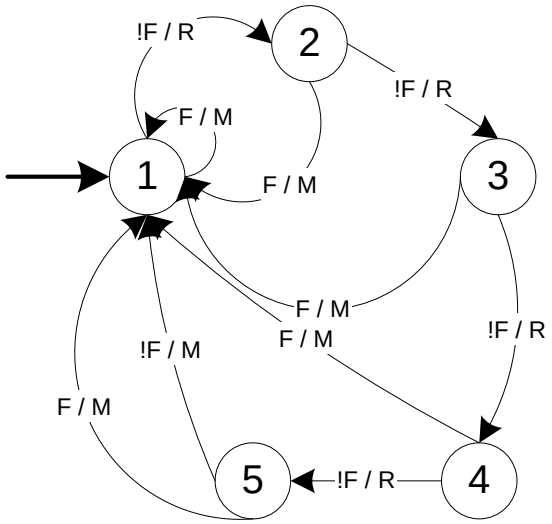


Рис. 14. Конечный автомат, описывающий поведение муравья

В рассматриваемой задаче требуется построить конечный автомат, под управлением которой муравей за 200 ходов съест все 89 яблок. Приведенный на рис. 14 автомат не решает задачу – за 200 ходов муравей съедает только 81 яблоко (на все 89 яблок ему требуется 314 ходов).

В работе [79] с помощью генетических алгоритмов построен автомат из 13 состояний, решающий поставленную задачу. В работе [35] также с помощью генетических алгоритмов построен автомат из 11 состояний, который позволяет муравью съесть все яблоки за 193 хода. В указанных работах автомат представлялся в виде битовой строки, а приспособленность особи определялась как число яблок, съеденное за 200 ходов.

В работах [156 – 159] **при участии автора настоящей диссертации** для решения рассматриваемой задачи было предложено применять генетическое программирование – в отличие от указанных выше работ в генетическом алгоритме автомат представлялся не в виде битовой строки, а в виде графа переходов. Кроме этого, существенное отличие предлагаемого в этой работе алгоритма состоит в использовании двух

операторов скрещивания – традиционного и сохраняющего значимые части автоматов [157].

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

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

Для осуществления скрещивания, сохраняющего значимые части автоматов *P1* и *P2*, найдем переходы, которые эти автоматы выполняют в течение первых сорока ходов по игровому полю. Обозначим множество таких переходов автоматов *P1* и *P2* как *TF(P1)* и *TF(P2)* соответственно. Множество переходов некоторого автомата *A* обозначим *T(A)*. Для автоматов, получающихся в результате скрещивания возможны два равновероятных варианта:

- $T(S1) = TF(P1) \cup (T(P2) \backslash TF(P2))$ и
  $T(S2) = TF(P2) \cup (T(P1) \backslash TF(P1))$;
- $T(S2) = TF(P1) \cup (T(P2) \backslash TF(P2))$ и
  $T(S1) = TF(P2) \cup (T(P1) \backslash TF(P1))$.

За счет применения такого алгоритма удалось построить *автомат, содержащий семь состояний* и позволяющий муравью съесть все 89 яблок за число шагов, не превышающее предельное. Граф переходов этого автомата приведен на рис. 15. При построении этого автомата функция приспособленности была вычислена порядка 130 миллионов раз, в то время как полный перебор в этом случае связан со значительно бо́льшим числом вычислений.

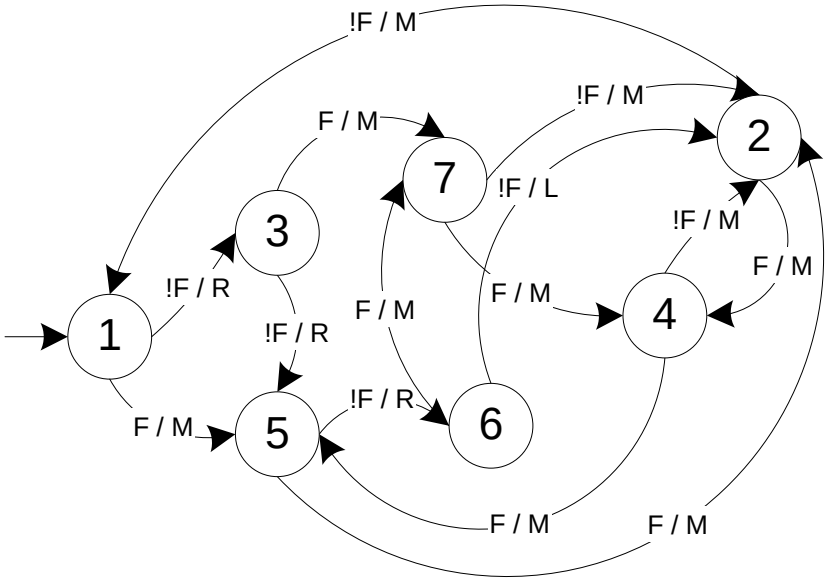


Рис. 15. Граф переходов автомата из семи состояний, решающего задачу «Умный муравей»

Кроме этого, в работе [156] описывается алгоритм перебора, с помощью которого было установлено, что *автоматы с шестью и менее состояниями задачу «Умный муравей» не решают*.

1.3.1.3. Задача «Умный муравей-3»

В рамках исследований по теме «Технология генетического программирования для генерации автоматов управления системами со сложным поведением» на кафедре «Технологии программирования» НИУ ИТМО для сравнения методов генерации конечных автоматов с помощью генетических алгоритмов [129 – 132] была предложена задача «Умный муравей-3». Постановка этой задачи, предложенной в работе [123], содержит несколько существенных отличий от задачи «Умный муравей».

Во-первых, расширена область обзора муравья – вместо одной клетки он видит восемь. Таким образом, множество значений входных переменных содержит $2^8 = 256$ элементов. На рис. 16 изображена область обзора муравья (клетка, в которой находится муравей, обозначена серым цветом).

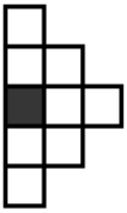


Рис. 16. Область видимости муравья

Во-вторых, расположение яблок на поле не фиксировано, а генерируется случайным образом. При этом вероятность того, что яблоко окажется в некоторой клетке, одинакова для всех клеток поля и равна μ.

В этом случае число яблок, съеденных муравьем за 200 ходов, является случайной величиной ξ (определяемой муравьем) на дискретном множестве элементарных исходов Ω – множестве расположений яблок – битовых матриц 32×32. Для каждого исхода ω, обозначим как apples(ω) соответствующее ему число яблок на поле, и поставим в соответствие ему вероятность $p(\omega) = \mu^{\text{apples}(\omega)}(1-\mu)^{1024-\text{apples}(\omega)}$.

Для вычисления этой величины в общем случае необходимо перебрать все возможные битовые матрицы размером 32×32. Поэтому для оценки эффективности автомата, задающего поведение муравья, вместо точного вычисления этого математического ожидания, оно вычисляется приближенно – с помощью моделирования поведения муравья на 10 000 случайно сгенерированных полях.

В рамках указанных исследований было предложено три метода генерации конечных автоматов с помощью генетических алгоритмов:

- метод сокращенных таблиц переходов;
- метод представления автоматов деревьями решений;
- метод совместного применения конечных автоматов и нейронных сетей.

Метод сокращенных таблиц переходов был предложен в работе [22]. Он позволяет решить проблему экспоненциального роста размера описания автомата, которая возникает при использовании полных таблиц

переходов – число строк в таблице равно 2^r, где r – число входных переменных. Опыт показывает, что в реальных задачах управляющие автоматы, построенные вручную, имеют гораздо меньше переходов, чем $|S| \cdot 2^r$ (здесь как $|S|$ обозначено число состояний автомата). Причина этого состоит в том, что в большинстве задач входные переменные имеют «локальную природу» по отношению к управляющим состояниям. В каждом состоянии *значимым* является лишь определенный, небольшой набор входных переменных, остальные же переменные не влияют на значение функции переходов и функции выходных воздействий. Именно это свойство позволяет существенно сократить размер описания состояний. Кроме того, использование этого свойства в процессе оптимизации позволяет получить результат, более похожий на автомат, построенный вручную, а, следовательно, и более понятный человеку.

Свойство локальности входных переменных можно применять для сокращения описания управляющего состояния разными способами. При разработке метода сокращенных таблиц был выбран один из подходов, при котором число значимых в состоянии входных переменных ограничивается некоторой константой *r'*.

К таблице, задающей сужение функции переходов на данное состояние, в этом случае добавляется битовый вектор, описывающий множество значимых входных переменных. Число строк таблицы в этом случае $2^{r'}$. Ее выбор зависит от сложности задачи. Как показывает опыт, для большинства автоматизированных объектов значение *r'* не превышает пяти.

Второй метод – *метод представления автоматов деревьями решений* [5, 135, 149]. Дерево решений является удобным способом задания дискретной функции, зависящей от конечного числа логических переменных. Оно представляет собой помеченное дерево, метки в котором расставлены по следующему правилу:

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

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

Метод представления автомата с помощью деревьев решений состоит в следующем: функции переходов и выходных воздействий автомата выражаются с помощью деревьев решений. Деревья решений состоят из узлов, среди которых выделяется корень. Каждый узел представляется следующим образом:

- номер переменной, соответствующей метке узла (для внутренних узлов);
- указатель на дочерний узел, соответствующий нулевому значению переменной (для внутренних узлов);
- указатель на дочерний узел, соответствующий единичному значению переменной (для внутренних узлов);
- ассоциированное выходное действие (для листьев);
- номер состояния, в которое ведет переход из данной вершины (для листьев).

Третий метод – *метод совместного применения автоматов и нейронных сетей* был **предложен автором настоящей диссертации** в работах [148, 154]. При использовании этого метода для управления системой со сложным поведением предлагается применять нейронную сеть и автомат, которые совместно строятся генетическим алгоритмом.

При этом нейронная сеть используется для классификации значений вещественных входных переменных и выработки входных логических

переменных для автомата, а автомат – для выработки выходных воздействий на систему со сложным поведением (рис. 17).

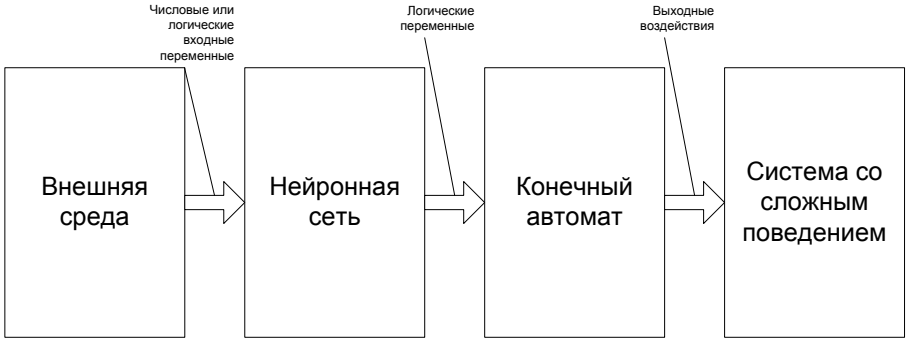


Рис. 17. Структурная схема системы управления при использовании метода совместного применения конечных автоматов и нейронных сетей

Одна из возможных структур нейронной сети и способ ее взаимодействия с конечным автоматом показаны на рис. 18. Отметим, что для решения других задач структура нейронной сети может быть изменена.

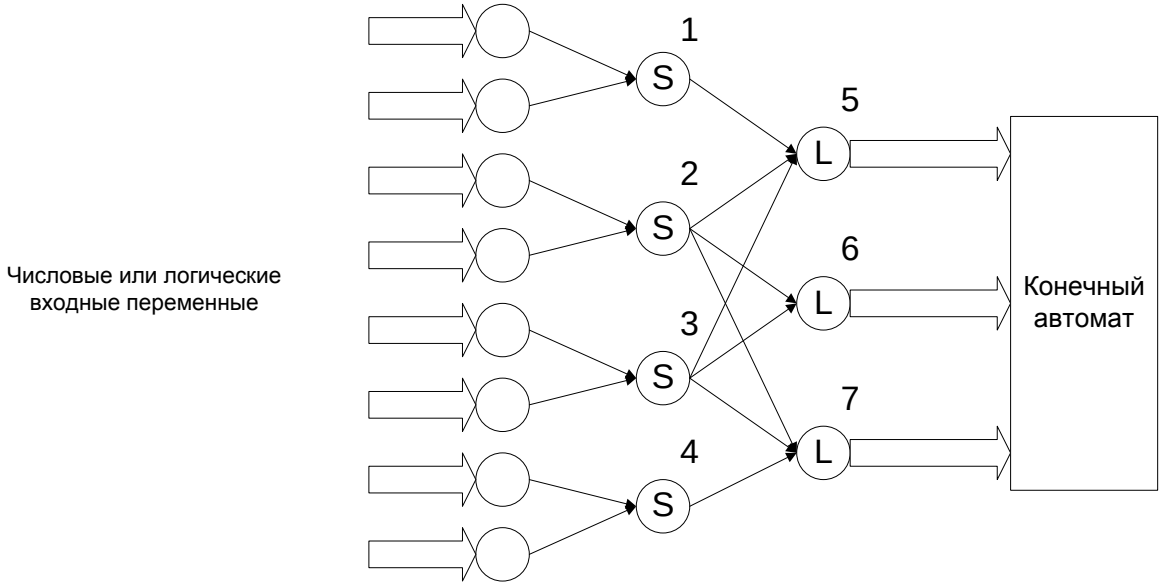


Рис. 18. Возможная структура нейронной сеть и ее взаимодействие с автоматом

Символами **S** на рис. 18 обозначены нейроны с сигмоидальной функцией активации, символом **L** – нейроны с пороговой функцией активации. Рядом с нейронами указаны их номера (они используются при

описании операции скрещивания нейронных сетей). На каждый из трех выходов нейронной сети поступает число равное нулю или единице. Таким образом, существует восемь вариантов комбинаций выходных сигналов нейронной сети (000, 001, 010, 011, 100, 101, 110, 111), подаваемых на вход автомата.

Экспериментальные исследования, проведенные в работе [132], показали, что при построении автомата для задачи «Умный муравей-3» три метода представления автоматов, рассмотренные выше, привели примерно одинаковые результаты.

1.3.1.4. Игра «Соревнование за ресурсы»

В работе [114] рассматривается задача построения автоматов для игры «Соревнование за ресурсы» (*Competition for Resources*). Игровое поле в этой игре имеет форму тора размером 10 на 10 клеток (тор можно представить в виде квадрата, у которого верхняя сторона склеена с нижней, а правая – с левой). Оно моделирует компьютерную сеть, в которой могут распространяться программные агенты.

В игре участвуют два агента. Находясь в некоторой клетке поля, агент получает информацию о четырех соседних клетках (соседи справа, слева, сверху и снизу). Каждая клетка может находиться в одном из трех состояний – никому не принадлежать, принадлежать первому агенту или принадлежать второму агенту. На основании полученной информации агент может переместиться в одну из соседних клеток, при этом перемещение разрешается выполнять либо в никому не принадлежащую клетку, либо в клетку, принадлежащую рассматриваемому агенту. Не совершать перемещение агенту не разрешается.

В начале первый агент находится в левом верхнем углу, а второй – в правом нижнем. Агенты делают ходы по очереди. Игра заканчивается, когда все клетки оказываются принадлежащими либо первому агенту, либо второму. Победителем признается тот агент, которому на момент

окончания игры принадлежит большее число клеток. В случае одинакового числа клеток, принадлежащих агентам, победителем признается второй агент.

В рассматриваемой работе стратегия поведения второго агента была задана, а цель состояла в построении автомата для управления первым агентом, который позволял бы достаточно часто выигрывать игру. Стратегия второго агента в рассматриваемой была такой: на каждом ходе при наличии хотя бы одной никому не принадлежащей соседней клетки, перемещение осуществлялось в случайно выбранную из них, а при отсутствии – случайно выбиралась одна из соседних клеток, принадлежащих этому агенту. Таким образом, стратегия второго агента имеет стохастический характер.

Для решения описанной задачи в работе [114] применяется генетический алгоритм. Автомат представляется в виде таблиц значений функции переходов и функции выходов. В работе рассматривалось построение автоматов, содержащих от одного до десяти состояний. При построении автомата из k состояний размер таблиц значений указанных функции составляет $k\times80$, так как существует 80 комбинаций входных переменных. Это объясняется тем, что имеется четыре переменные, каждая из которых может принимать три возможных значений, при этом ровно одна комбинация (все четыре соседние клетки принадлежат другому агенту) не может встретиться в игре (это объясняется тем, что клетка, из которой агент пришел в текущую, принадлежит ему).

Мутации с некоторой заданной вероятностью подвергалась каждая из ячеек таблицы значений функции переходов и функции выходов. При этом с вероятностью p менялось состояние, в которое ведет переход, а с вероятностью $1–p$ – значение выхода, выполняемое на этом переходе. Кроме этого, использовалась операция однородного скрещивания. Для каждой позиции (i, j) в таблицах значений функций переходов и выходов

производился обмен ячеек между «родителями» с некоторой заданной вероятностью.

Так как поведение второго агента носит вероятностный характер, то оптимальным вариантом функции приспособленности была бы вероятность выигрыша для первого агента, поведение которого задается автоматом, описываемым особью. Однако вычисление этой вероятности сопряжено с перебором всех вариантов поведения первого агента и требует больш*и*х вычислительных ресурсов. Поэтому указанная вероятность вычисляется приближенно – вычисляется не вероятность выигрыша, а доля игр, выигранных агентом.

Вычисление функции приспособленности выполняется в два этапа – на первом из них для всех автоматов, описываемых особями текущего поколения, проводится 500 игр против заданной стратегии поведения второго агента. Если в текущем поколении находится автомат, который по результатам этого вычисления показывает результат, превосходящий наилучший автомат, рассмотренный в процессе работы алгоритма, то для автомата из текущего поколения проводится более точное вычисление функции приспособленности – по итогам 10 000 игр.

В качестве метода отбора, применяемого при генерации нового поколения, использовалась комбинация пропорционального отбора и элитизма. Это означает, что особь, имеющая наибольшее значение функции приспособленности в текущем поколении напрямую переходила в следующее, а для остальных вероятность перехода была пропорциональна значению функции приспособленности.

Вычислительные эксперименты проводились при значениях числа состояний в генерируемых автоматах от одного до десяти. Их результаты показывают, что при решении этой задачи существенное преимущество имеют автоматы, имеющие более двух состояний. Автоматы с числом состояний от трех до десяти показывали примерно одинаковые

результаты – около 90% побед. Кроме этого, были проведены эксперименты, в которых в процессе мутации разрешалось добавлять и удалять состояния – в этом случае было замечено, что в построенных автоматах используется меньше половины состояний. На основании результатов экспериментов был сделан вывод о том, что операции добавления и удаления состояний являются слишком «разрушительными» для их применения в эволюционных алгоритмах построения автоматов.

Анализ поведения агентов, управляемых построенными автоматами, показал, что поведение агента достаточно быстро зацикливается, что приводит к ухудшению его результатов. Для того чтобы устранить этот недостаток, в рассматриваемой работе предлагается метод, основанный на наблюдении за поведением агента во время игры. Этот метод состоит в том, что к автомату добавляется память размером в 20 ячеек, которая позволяет отслеживать циклы в поведении агента. В этой памяти записывались ячейки, посещенные агентом. Если далее обнаруживалось, что действие, генерируемое автоматом, приводит к зацикливанию, то действие выбиралось случайным образом. Применение такого метода позволило построить систему управления агентом, которая добивалась побед в 96% игр. Описанный метод устранения зацикливания является одной из разновидностей динамической верификации [101].

1.3.2. Методы, использующие обучающие примеры при вычислении функции приспособленности

В настоящем разделе приводится обзор методов построения конечных автоматов, использующих обучающие примеры (тесты) при вычислении функции приспособленности.

1.3.2.1. Построение конечных распознавателей

Рассматриваемая ниже модель в литературе обычно называется «детерминированный конечный автомат». В настоящей работе, однако, для

того чтобы отличать автоматы такого типа от управляющих конечных автоматов и конечных преобразователей для этой модели используется термин «конечный распознаватель»,.

Конечным распознавателем A называется пятерка $(Q, \Sigma, \delta, q_0, F)$, где Q – множество состояний автомата, Σ – его входной алфавит, δ – функция переходов, q_0 – начальное состояние, F – множество допускающих состояний. Аргументами функции переходов являются текущее состояние и входной символ, а значением – новое состояние.

Из теории формальных языков известно, что конечные детерминированные автоматы способны распознавать регулярные языки [26]. В связи с этим в ряде работ рассматривалась задача построения по множеству примеров автомата, распознающего некий язык. Для решения этой задачи успешно применялись генетические алгоритмы.

Задача может быть усилена до построения автомата с минимальным числом состояний. В работе [61] показано, что эта задача является NP-полной.

В работе [38] автоматы представлялись с помощью таблицы переходов. Разработанный в этой работе метод позволяет поддерживать в популяции автоматы с разным числом состояний. Функция приспособленности учитывает три компонента – число верно распознанных обучающих примеров, число состояний и переходов автомата, степень общности языка, соответствующего построенному автомату. Это позволяет сузить область поиска, и находить языки, соответствующие определенным критериям. В обучающий набор могут входить примеры слов, которые как принадлежат, так и не принадлежат языку.

Приведем описание генетической операции «репродукция». Число состояний потомка выбирается случайно из диапазона $[k - 2, k + 2]$, где k – число состояний первого родителя. После этого каждый переход

копируется из родителей, при этом вероятность копирования перехода из заданной особи прямо пропорциональна значению ее функции приспособленности. Переходы, представленные только в одном из родителей, копируются из того родителя, в котором присутствуют. Переходы, отсутствующие в обоих родителях, генерируются случайно.

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

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

В работе [88] производится *сравнение алгоритмов построения конечных автоматов-распознавателей*. При этом рассматриваются два типа алгоритмов: эволюционные и основанные на построении префиксного дерева и дальнейшем объединении состояний на основе свидетельств (*Evidence-Driven State Merging*, далее *EDSM*).

Входными данными для алгоритма построения конечного автомата-распознавателя является набор слов, для каждого из которых известно, должно ли оно допускаться автоматом или не должно. Этот набор слов в дальнейшем будет называться множеством обучающих примеров. Выходными данными для рассматриваемого алгоритма является описание построенного конечного автомата, который удовлетворяет входным данным.

Для представления конечного автомата-распознавателя в рассматриваемой работе применяется табличная форма задания его

функции переходов. При этом начальным состоянием всегда считается состояние, имеющее номер «0». В рассматриваемой работе рассматриваются только автоматы, обрабатывающие слова над двухсимвольным алфавитом, поэтому число возможных таблиц переходов для k состояний равно k^{2k}. Если учитывать то, что каждое состояние автомата может быть допускающим или недопускающим, то общий размер пространства поиска равен $2^k \cdot k^{2k}$.

Для сокращения пространства поиска в работе [88] применяется специальный метод определения того, какие состояния автомата являются допускающими, а какие – нет. Для этого предлагается использовать так называемый «алгоритм расстановки пометок» на состояниях. Он состоит в следующем. Для каждой строки t из множества обучающих примеров на единицу увеличивается значение $h[f(t)][\chi]$, где $f(t)$ – номер состояния, в которое приходит рассматриваемый автомат после обработки строки t, а число χ равно единице, если строка t должна допускаться автоматом, и равно нулю, если не должна допускаться автоматом. После этого те состояния s, для которых $h[s][1] > h[s][0]$, помечаются как допускающие, а все остальные – как недопускающие. За счет применения описанного алгоритма размер пространства поиска существенно сокращается – его размер становится равным k^{2k}.

В качестве эволюционного алгоритма в рассматриваемой работе применяется метод спуска на основе случайных мутаций.

Для вычисления функции приспособленности определяется доля строк из обучающего набора, которые правильно классифицируются (допускаются или не допускаются) автоматом.

Сравнение эволюционного алгоритма проводилась на двух наборах тестовых данных. Первый набор тестовых данных был предложен в работе [116]. На этом наборе входных данных сравнение проводилось не только с алгоритмом *EDSM*, но и с алгоритмом генетического программирования,

предложенным в работе [116]. Это сравнение показывает, что разработанный алгоритм эволюционного программирования превосходит и алгоритм *EDSM*, и алгоритм из работы [116], как по точности построенных автоматов, так и по времени работы. Например, среднее время, которое требуется предложенному в рассматриваемой работе алгоритму для построения конечного автомата, составляет 1.6 миллисекунды, а алгоритму *EDSM* требуется 37 миллисекунд в среднем (вычислительные эксперименты проводились на компьютере с процессором *Pentium IV* с тактовой частотой 2.4 ГГц).

Второй набор тестовых данных строился следующим образом:

1. Выбиралось число m.
2. Генерировался случайный ориентированный граф из $5m/4$ вершин, в котором каждая вершина имеет степень два.
3. Случайным образом выбиралась вершина – начальное состояние.
4. Рассматривались все достижимые из начального состояния.
5. Каждое из состояний с равной вероятностью становилось либо допускающим, либо недопускающим.
6. После этого генерировалось множество строк над двухсимвольным алфавитом, для каждой из которых с помощью построенного на шагах 1 – 5 автомата определялось, допускается она или нет.

На этом наборе данных эволюционный алгоритм при $m = 4, 8, 16$ превосходил алгоритм *EDSM*, но при $m = 32$ ему уступал.

На основании проведенных экспериментов был сделан вывод, что разработанный эволюционный алгоритм превосходит алгоритм *EDSM* в том случае, когда необходимо построить автомат с относительно небольшим числом состояний, а множество обучающих примеров содержит небольшую долю строк заданной длины.

В работе [89] также рассматривается задача построения конечных автоматов-распознавателей с помощью эволюционных алгоритмов. Предлагаемый в рассматриваемой работе эволюционный алгоритм является улучшенной версией алгоритма, предложенного в работе [88].

Конечный распознаватель в рассматриваемой работе представляется так же, как и в работе [88]: в хромосому входит таблица переходов, а пометки на состояниях расставляются с помощью специального алгоритма. В качестве эволюционного алгоритма в работе [89] также используется (1+1)-эволюционная стратегия, но операцию мутации предлагается выполнять не так, как в работе [88].

В последней при выполнении мутации случайным образом выбиралась одна из ячеек таблицы переходов, после этого значение в ней заменялось на случайно сгенерированное. Предлагаемый в рассматриваемой работе метод выполнения операции мутации учитывает поведение автомата при обработке обучающего набора.

Для каждого перехода t в автомате вычисляются две величины. Число строк, которые были правильно классифицированы автоматом и при обработке которых использовался переход t, обозначается как $\mathrm{cor}(t)$. Число строк, неправильно классифицированных автоматом и при обработке которых использовался переход t, обозначается как $\mathrm{err}(t)$. При этом вероятность того, что переход t будет выбран при выполнении операции мутации, вычисляется по формуле: $p(t) = \frac{\mathrm{err}(t)}{\mathrm{cor}(t) + \mathrm{err}(t)}$. Здесь $p(t) = 0$, если $\mathrm{err}(t) = 0$. Такой подход к выполнению операции мутации требует выполнения двух проходов для вычисления значения функции приспособленности и указанных вероятностей. На первом из них неизвестно, какие состояния являются допускающими, а какие – нет. Этот проход необходим для алгоритма пометки состояний. На втором проходе

известно, какие состояния являются допускающими, поэтому указанные вероятности могут быть вычислены.

Кроме описанного способа выполнения операции мутации в рассматриваемой работе также предлагается метод: для каждого состояния запоминаются строки, которые оно должно принимать, и строки, которое оно должно не принимать. В соответствии с алгоритмом расстановки пометок, состояние будет допускающим, если строк, которое оно должно допускать больше, чем строк, которое оно не должно допускать, и недопускающим – в противном случае. При выполнении операции мутации случайным образом выбирается строка, которую автомат классифицирует неправильно. После этого случайным образом выбирается один из переходов, которые используется при обработке выбранной строки. Изменению подвергается та ячейка таблицы переходов, которая ему соответствует.

Для сравнения алгоритмов в рассматриваемой работе применялось два набора тестовых данных. Первый строился так же, как второй набор тестовых данных в работе [88] (описание приведено выше). Второй набор данных был зашумленным – для некоторых строк из обучающего набора было неправильно указано, должны они допускаться автоматом или нет.

На первом наборе данных проводилось сравнение четырех алгоритмов:

- не использующего алгоритм расстановки пометок (*plain*);
- метода из работы [88] (*smart labeling*);
- метода, использующего первый из предлагаемых в рассматриваемой работе подходов к выполнению операции мутации (*sampled*);
- метода, использующего второй из предлагаемых в рассматриваемой работе подходов к выполнению операции мутации (*quick-sampled*).

Результаты вычислительных экспериментов показали, что при восьми состояниях автомата, на основе которого генерировались

обучающие наборы, из 50 случаев алгоритм *plain* построил автомат в 26, алгоритм *smart labeling* – в 42, алгоритм *sampled* – в 47, а алгоритм *quick-sampled* – в 43. При использовании автомата из 16 состояний результаты были такими: *plain* – 5 из 50, *smart* – 21 из 50, *sampled* – 26 из 50, *quick-sampled* – 4 из 50. Если же в исходном автомате было 32 состояния, то только два алгоритма справились с задачей: *smart* правильно строил автомат в 6 случаях из 50, а *sampled* – в 12 случаях из 50.

Как отмечалось выше, второй набор данных содержал зашумленные обучающие наборы. На этом наборе данных сравнение проводилось с алгоритмом *EDSM* и с другими алгоритмами, участвовавшими в соревновании в рамках конференции *GECCO* 2004 [144]. Результаты вычислительных экспериментов показали, что предлагаемый в рассматриваемой работе эволюционный алгоритм (*sampled*) превосходит все существующие алгоритмы построения автоматов на этом обучающем наборе по точности их построения. При этом отметим, что предложенный алгоритм эффективен только для построения автоматов из относительно небольшого числа состояний (порядка нескольких десятков), в то время как с помощью алгоритма *EDSM* могут быть успешно построены автоматы из сотен состояний.

В работе [115] также рассматривается задача построения конечных автоматов-распознавателей. В качестве метода представления конечных автоматов выбраны двоичные строки, в качестве операций мутации и скрещивания используется однородное скрещивание и мутация.

В качестве исходных данных для генетического алгоритма выступают пары, состоящие из входных слов и последовательностей пометок состояний, которые используются при обработке соответствующих слов.

Рассматривается три варианта функции приспособленности. Первый из них основан на строгом сравнении строк, второй – на вычислении

функции приспособленности, третий – на вычислении длины общего префикса строк. В качестве метода генерации следующего поколения используется метод рулетки.

Алгоритмы, предлагаемые в рассматриваемой работе, реализованы в инструментальном средстве *GeSM*. В этом инструментальном средстве, кроме самого генетического алгоритма, также реализован алгоритм удаления недостижимых состояний автоматов.

Экспериментальное исследование разработанных генетических алгоритмов проводилось на задачах построения автоматов для проверки четности, элемента задержки на два такта, распознавателя паттернов и счетчика.

1.3.2.2. Построение конечных преобразователей

Конечным преобразователем называется шестерка $(Q, \Sigma, O, \delta, \gamma, q_0)$, где Q – множество состояний автомата, Σ – его входной алфавит, O – его выходной алфавит, δ – функция переходов, γ – функция выходов, q_0 – начальное состояние. Аргументами функции переходов являются текущее состояние и входной символ, а значением – новое состояние. Аргументами функции выходов также являются текущее состояние и входной символ, а значением – символ из выходного алфавита или пустая строка.

В работе [90] для построения конечных преобразователей применяется (1 + 1)-эволюционная стратегия. Конечный преобразователь представляется в виде набора двух таблиц – значений функции переходов и значений функции выходов. При этом предполагалось, что на каждом из переходов конечный преобразователь может вывести не более одного символа.

Входными данными для построения конечного автомата-распознавателя является набор обучающих примеров – пар слов, одно из которых является входным, а второе – соответствующим ему выходным.

Опишем алгоритм выполнения операции мутации в алгоритме, предложенном в рассматриваемой работе. Выполняется мутация одного из трех типов:

- добавление состояния – выполняется с вероятностью 0.1, если автомат содержит меньше заданного максимального числа состояний. При этом переходы из нового состояния генерируются случайным образом;
- удаление состояния;
- случайное изменение одного перехода.

Последние две мутации из перечисленных выбираются равновероятно.

В рассматриваемой работе рассматриваются три варианта функции приспособленности:

- на основе строгого сравнения;
- на основе вычисления расстояния Хэмминга [64];
- на основе вычисления редакционного расстояний (расстояния Левенштейна) [13].

Опишем указанные функции приспособленности подробнее. Пусть конечный автомат-преобразователь, задаваемый рассматриваемой особью эволюционного алгоритма, на обучающем примере, который содержит входную строку *s* и эталонную выходную строку *s'*, выдал строку *s''*. При использовании строгого сравнения значение функции приспособленности *f* будет вычисляться по формуле: $f = \begin{cases} 0, s' = s'' \\ 1, s' \neq s'' \end{cases}$.

При использовании расстояния Хэмминга функция приспособленности вычисляется по формуле: $f = \frac{\sum_{i=1}^{\min(|s'|,|s''|)} \Delta(s'_i, s''_i)}{\max(|s'|,|s''|)}$, где как $|s'|$ и $|s''|$ обозначены длины строк *s'* и *s''* соответственно. Как Δ в формуле

обозначена функция, значение которой равно нулю, если значения ее аргументов совпадают, и единице – в противном случае.

При использовании редакционного расстояния функция приспособленности вычисляется по формуле: $f = \frac{\text{edits}(s', s'')}{\max(|s''|, |s''|)}$. Здесь как edits(*s’, s’’*) обозначено редакционное расстояние между строками *s’* и *s’’*, которое равно минимальному числу операций вставки, замены или удаления символа, которые необходимо выполнить, для того чтобы преобразовать строку *s’* в строку *s’’*. Отметим, что значения всех трех указанных функций приспособленности находятся в отрезке от 0 до 1, при этом меньшее значение соответствует большему соответствию выходной строки эталону.

Экспериментальное исследование алгоритмов проводилось на задаче построения конечного преобразователя для преобразования представления двумерного изображения из цепного кода для четырех направлений в цепной код для восьми направлений. Применялись два набора данных. В «простом» наборе было достаточно много коротких строк, а в «сложном» – их доля была намного меньше.

Результаты экспериментов показали, что наилучшие результаты на «сложном» наборе данных достигаются при использовании функции приспособленности, основанной на редакционном расстоянии. Второй по эффективности является функция, основанная на расстоянии Хэмминга, а третьей – основанная на строгом сравнении. На «простом» наборе ситуация примерно такая же, только функции строгого сравнения и вычисления расстояния Хэмминга меняются местами.

1.3.3. Методы, использующие верификацию при вычислении функции приспособленности

Известна только одна работа [80], посвященная построению взаимодействующих конечных автоматов с использованием генетического

программирования с функцией приспособленности, при вычислении которой применяется верификация моделей [9]. Верификация моделей – это набор методов и алгоритмов, которые позволяют определять, соответствует ли программа некоторому множеству требований. При этом требования выражаются на языке темпоральной логики – записываются утверждения о поведении программы во времени.

В рассматриваемой работе применяется темпоральная логика *CTL* (*Computation Tree Logic* – логика деревьев вычислений), а для верификации используется система *SMV* (http://www.cs.cmu.edu/~modelcheck/smv.html).

В рассматриваемой работе решается задача построения системы взаимодействующих конечных автоматов. Это означает, что автоматы, входящие в систему, имеют общие глобальные переменные и каналы связи, по которым могут передаваться сообщения.

В качестве входных данных для вычисления функции приспособленности используется описание системы автоматов, а также набор темпоральных формул, которые требуется проверить. Результатом вычисления функции приспособленности является число верных для системы автоматов формул.

В качестве эволюционного алгоритма в рассматриваемой работе применяется (1+λ)-эволюционная стратегия. Возможные варианты выполнения операции мутации таковы:

- с вероятностью 0.4 добавляется новое состояние;
- всегда (с вероятностью 1.0) добавляется новое состояние со случайно выбранной пометкой;
- с вероятностью 0.3 удаляется случайно выбранный переход;
- с вероятностью 0.1 изменяется пометка одного из состояний;

Экспериментальное исследование разработанного алгоритма проводилось на задаче построения автомата управления кофе-машиной. Применялось два набора входных данных. Первый из них содержал восемь

темпоральных формул, а второй – 17. Второй набор описывал кофе-машину, которая обладает большей функциональностью, чем описываемая первым набором. Задача состояла в нахождении системы из двух автоматов, один из которых описывал поведение пользователя, а второй – кофе-машины.

Вычислительные эксперименты проводились следующим образом: выполнялось 30 запусков эволюционного алгоритма, в каждом из которых проводилось 20 итераций эволюционного алгоритма. На первом наборе данных в 25 из 30 запусков была найдена система автоматов, удовлетворяющая всем восьми формулам, а в пяти оставшихся – удовлетворяющая семи.

На втором наборе данных в каждом запуске выполнялось 30 итераций эволюционного алгоритма. Ни в одном из запусков не была найдена система автоматов, удовлетворяющая всем темпоральным формулам. При этом в семи запусках была найдена система автоматов, для которой выполняются 16 из 17 формул.

1.3.4. Анализ эволюционных алгоритмов построения автоматов

Обобщим материал разделов 1.3.1 – 1.3.3, проведя сравнение рассмотренных эволюционных алгоритмов построения автоматов (табл. 1). Сравнение алгоритмов проводится по следующим параметрам:

- Тип конечных автоматов.
- Тип эволюционного алгоритма.
- Метод представления особей в эволюционном алгоритме.
- Метод вычисления функции приспособленности.

Таблица 1. Сравнительная характеристика эволюционных алгоритмов построения автоматов

Название работы	Тип конечных автоматов	Тип эволюционного алгоритма	Метод представления особей в эволюционном алгоритме	Метод вычисления функции приспособленности
Artificial Intelligence through Simulated Evolution [53]	Конечный автомат-преобразователь	(μ+λ)- эволюционная стратегия	Граф переходов	На основе моделирования
The Genesys System: Evolution as a Theme in Artificial Life [79]	Управляющий конечный автомат	Генетический алгоритм	Битовая строка	На основе моделирования
Evolutionary Module Acquisition [35]	Управляющий конечный автомат	Генетический алгоритм	Битовая строка	На основе моделирования
Применение генетических алгоритмов для построения автоматов с минимальным числом состояний для задачи «Умный муравей» [157]	Управляющий конечный автомат	Генетическое программирование	Граф переходов	На основе моделирования
Применение	Управляющие	Генетическое	Функции	На основе

генетического программирования для генерации автоматов с большим числом входных переменных [23]	конечные автоматы	программирование	переходов и действий в виде сокращенных таблицы	моделирования
Метод представления автоматов деревьями решений для использования в генетическом программировании [5]	Управляющие конечные автоматы	Генетическое программирование	Функции переходов и действий в виде деревьев решений	На основе моделирования
Совместное применение генетического программирования, конечных автоматов и искусственных нейронных сетей для построения системы управления беспилотным летательным аппаратом [148]	Управляющие конечные автоматы	Генетическое программирование	Таблица значений функции переходов и таблица значений функции выходов, нейронная сеть для обработки входных переменных	На основе моделирования
Evolving Finite-State	Управляющие конечные	Генетическое программирование	Таблица значений	На основе моделирования

Machine Strategies for Protecting Resources [114]	автоматы		функции переходов и таблица значений функции выходов	
Learning DFA: Evolution versus Evidence Drive State Merging [88]	Конечные автоматы-распознаватели	Метод спуска на основе случайных мутаций	Таблица значений функции переходов	На основе обучающих примеров
Learning Deterministic Finite Automata with a Smart State Labeling Algorithm [89]	Конечные автоматы-распознаватели	Метод спуска на основе случайных мутаций	Таблица значений функции переходов	На основе обучающих примеров
A Genetic Algorithm for Finite State Automata Induction with Application to Phonotactics [38]	Конечные автоматы-распознаватели	Генетическое программирование	Таблица значений функции переходов	На основе обучающих примеров
Genetic Inference of Finite State Machines [115]	Конечные автоматы-распознаватели	Генетический алгоритм	Битовые строки	На основе обучающих примеров
Evolving Finite-State Transducers: Some Initial Explorations [90]	Конечные автоматы-преобразователи	Метод спуска на основе случайных мутаций	Таблица значений функции переходов и таблица значений функции выходов	На основе обучающих примеров

Genetic Programming with Fitness based on Model Checking [80]	Система двух взаимодействующих конечных автоматов	(1+λ)- эволюционная стратегия	Графы переходов	На основе верификации

На основании проведенного обзора можно сделать следующие выводы:

- главное отличие управляющих конечных автоматов от других типов конечных автоматов (конечных преобразователей и распознавателей) состоят в том, что в пометки переходов входят булевы формулы, зависящие от входных переменных;
- для построения управляющих конечных автоматов ранее применялись только методы, использующие вычисление функции приспособленности на основе моделирования;
- в работах, использующих эволюционную стратегию и метод спуска (в частности, в работах [80, 89]) выбор этих методов обоснован тем, что для конечных автоматов трудно предложить осмысленный способ выполнения операции скрещивания;
- практически нет работ, в которых проводится сравнение различных методов построения конечных автоматов, основанных на алгоритмах поисковой оптимизации. Это подтверждается работой [69], в которой упоминается, что в области поисковой инженерии ПО наблюдается такая же ситуация;
- только одна работа посвящена построению конечных автоматов с вычислением функции приспособленности на основе верификации.

1.4. Задачи, решаемые в диссертационной работе

Из выполненного обзора следует, что существующие методы построения *управляющих* конечных автоматов используют вычисление функции приспособленности *на основе моделирования* и обладают следующими недостатками:

- для решения новой задачи необходимо заново писать программу моделирования для вычисления функции приспособленности, что может быть весьма трудоемким;
- каждое вычисление функции приспособленности может требовать значительного времени, так как связано с моделированием поведения автомата в некоторой внешней среде.

На основании проведенного обзора сформулируем задачи, решаемые в диссертационной работе:

1. Разработать метод построения управляющих конечных автоматов по обучающим примерам на основе эволюционных алгоритмов.
2. Разработать метод выполнения операции скрещивания для генетических алгоритмов, учитывающий поведение автоматов на обучающих примерах.
3. Разработать метод построения управляющих конечных автоматов по обучающим примерам и темпоральным формулам на основе эволюционных алгоритмов и верификации.
4. Разработать технологию построения управляющих конечных автоматов по обучающим примерам и темпоральным формулам.
5. Разработать инструментальное средство для автоматизации построения управляющих конечных автоматов.

6. Внедрить разработанные методы при построении автомата управления моделью беспилотного самолета и в учебный процесс.

Выводы по главе 1

1. Приведены результаты обзора работ, посвященных автоматному программированию и поисковой инженерии программного обеспечения.
2. Приведен обзор методов построения конечных автоматов на основе эволюционных алгоритмов.
3. Сформулированы основные недостатки существующих методов построения управляющих конечных автоматов на основе эволюционных алгоритмов.
4. Сформулированы задачи, решаемые в диссертационной работе.

ГЛАВА 2. МЕТОДЫ ПОСТРОЕНИЯ УПРАВЛЯЮЩИХ КОНЕЧНЫХ АВТОМАТОВ НА ОСНОВЕ ЭВОЛЮЦИОННЫХ АЛГОРИТМОВ ПО ОБУЧАЮЩИМ ПРИМЕРАМ И ТЕМПОРАЛЬНЫМ ФОРМУЛАМ

Настоящая глава посвящена методам построения управляющих конечных автоматов на основе эволюционных алгоритмов:

- методу построения управляющих конечных автоматов по обучающим примерам на основе эволюционных алгоритмов;
- методу выполнения операции скрещивания с учетом поведения автоматов на обучающих примерах;
- методу построения управляющих конечных автоматов по обучающим примерам и темпоральным формулам на основе эволюционных алгоритмов и верификации.

2.1. Метод построения управляющих конечных автоматов по обучающим примерам на основе эволюционных алгоритмов

В настоящем разделе описывается разработанный метод построения управляющих конечных автоматов по обучающим примерам (*тестам*) на основе эволюционных алгоритмов [133, 150, 160, 161].

2.1.1. Входные данные

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

- множество событий E;
- множество входных переменных X;
- множество выходных воздействий Z;
- множество тестов Tests (число тестов в дальнейшем будет обозначаться как n);
- максимальное число состояний в искомом автомате k.

Опишем структуру тестов: i-й из них состоит из двух последовательностей – входной Input[i] и выходной Answer[i]. *Элементами входной последовательности* являются пары (e, f), где e – некоторое событие из множества E, а f – логическая формула от входных переменных, задающая условие перехода.

2.1.1.1. Предварительная обработка входных данных

Перед выполнением эволюционного алгоритма производится предварительная обработка входных данных. Создается список всех условий, входящих во входные последовательности тестов: f_1, f_2, …, f_l. Далее строятся две таблицы, содержащие логические значения:

- `areEquivalent[i, j]` – верно ли, что f_i и f_j эквивалентны;
- `haveCommon[i, j]` – верно ли, что f_i и f_j имеют общую выполняющую подстановку.

Для построения этих таблиц перебираются все пары охранных условий (f_i, f_j), $i \neq j$, и для каждой из них строится таблица истинности для формул ($f_i \leftrightarrow f_j$) и !(f_i & f_j). Если первая из них является тавтологией, то в ячейку таблицы `areEquivalent[i, j]` записывается значение «истина», если вторая формула всегда ложна, то в ячейку таблицы `haveCommon[i, j]` записывается значение «истина».

В дальнейшем информация, содержащаяся в этих таблицах, будет использоваться в алгоритме расстановки выходных воздействий и в алгоритме удаления дублированных и неоднозначных переходов.

Время работы алгоритма построения этих таблиц – $O(2^{2r'} \cdot l^2)$, где, как и раньше, r' – максимальное число входных переменных, от которых зависит некоторое условие перехода. На практике, как уже отмечалось выше, значение r' не превышает пяти, что позволяет говорить о том, что время, требуемое на предварительную обработку данных пропорционально квадрату суммарного размера входных последовательностей.

2.1.2. Выходные данные

Выходные данные должны представлять собой описание автомата, содержащего не более *k* состояний, удовлетворяющего каждому тесту из множества Tests и обладающего свойством непротиворечивости.

Управляющий конечный автомат *A удовлетворяет тесту* Test = (Input, Answer), если результат обработки входной последовательности Input при старте из начального состояния автомата совпадает с последовательностью Answer.

Результат обработки входной последовательности Input при начале из состояния *s* определяется рекурсивно:

- если последовательность Input пуста, то результат ее обработки также является пустой последовательностью;
- если последовательность Input не пуста, а ее первый элемент имеет вид (*e*, *f*), то результат обработки последовательности Input конкатенация двух последовательностей – результата обработки элемента входной последовательности (*e*, *f*) в состоянии *s* и результата обработки оставшейся части последовательности Input при начале из состояния, в которое ведет переход из *s* по событию *e* и условию, эквивалентному *f*. Если результат обработки оставшейся части последовательности Input не определен, то и результат обработки последовательности также не определен.

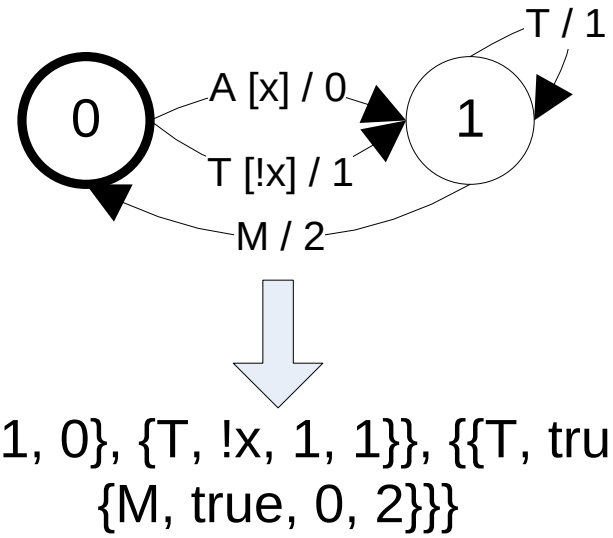
Результатом обработки элемента входной последовательности (*e*, *f*) в состоянии *s* является последовательность выходных воздействий, соответствующая переходу, который помечен событием *e* и условием, эквивалентным *f*. Если же такого перехода из состояния *s* нет или таких переходов несколько, то результат обработки указанного элемента не определен.

Например, для автомата управления часами с будильником, граф переходов которого приведен на рис. 6, результат обработки элемента входной последовательности (*H, true*) в состоянии «1. Будильник выключен» является последовательность, содержащее одно выходное воздействие – z_1, результат обработки элемента входной последовательности (*A, true*) в состоянии «2. Установка времени будильника» – пустая последовательность, результат обработки элемента входной последовательности (*T, !x₂&x₁*) в состоянии «3. Будильник включен» – последовательность из двух элементов z_5, z_6, а результат обработки элемента (*T, x₂&x₁*) в состоянии «3. Будильник включен» не определен.

Результат обработки входной последовательности (*A, true*), (*M, true*) при начале из состояния «1. Будильник выключен» – последовательность из одного элемента z_4, а результат обработки входной последовательности (*A, true*), (*A, true*), (*T, x₂&x₁*) при начале из того же состояния не определен.

2.1.3. Представление управляющего конечного автомата в виде особи в эволюционных алгоритмах

Конечный автомат в эволюционных алгоритмах предлагается представлять в виде объекта, который содержит описания переходов для каждого из состояний. Во всех автоматах, рассматриваемых в процессе работы эволюционных алгоритмов, номер начального состояния равен нулю. Для каждого из состояний хранится список переходов. Каждый переход описывается событием, при поступлении которого этот переход выполняется, условием перехода и *числом* выходных воздействий, которые должны быть сгенерированы при выборе этого перехода. При этом в качестве условий переходов используются булевы формулы, встречающиеся во входных последовательностях тестов. На рис. 19 приведен пример автомата и его представления в виде особи эволюционного алгоритма.



{2, {{A, x, 1, 0}, {T, !x, 1, 1}}, {{T, true, 1, 1}, {M, true, 0, 2}}}

Рис. 19. Представление конечного автомата в виде особи эволюционного алгоритма

Для построения автоматов в настоящей работе *существующие эволюционные алгоритмы предлагается модифицировать*, добавив перед вычислением функции приспособленности в каждый из них дополнительный шаг – расстановку выходных воздействий. Он необходим, так как особь в эволюционном алгоритме представляет собой «каркас» автомата. При этом расстановка выходных воздействий должна выполняться так, чтобы значение функции приспособленности получившегося в результате автомата было максимальным или близким к нему.

2.1.4. Алгоритм расстановки выходных воздействий

Опишем алгоритм расстановки выходных воздействий для дискретных воздействий. Как было сказано выше, для каждого перехода в особи эволюционного алгоритма записано, сколько выходных воздействий должно вырабатываться при его выборе.

Подадим на вход конечного автомата входную последовательность, соответствующую одному из тестов, и будем наблюдать, за тем, какие переходы выполняет автомат (для определения этих переходов

необходимо использовать значения из таблицы `areEquivalent`). 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.

Например, пусть входной последовательности событий *A, T, T, M, H, T, T* соответствует выходная последовательность z_5, z_5, z_4, z_3, z_5, z_5, и при обработке входной последовательности выполняются следующие переходы: 1 → 2 → 2 → 2 → 3 → 4 → 2 → 2. Пусть при этом на первом из переходов не должно выполняться ни одно выходное воздействие, а на каждом из остальных – должно выполняться одно (рис. 20).

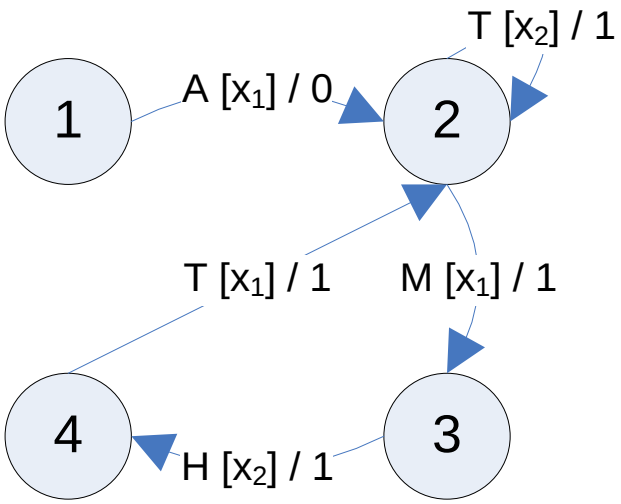


Рис. 20. Некоторые переходы «каркаса» автомата

На рис. 20 для каждого из переходов, кроме события, при возникновении которого он выполняется, указано число выходных воздействий, которые ему соответствуют.

Тогда на основании этого теста можно сделать вывод о том, что на переходе по событию *T* из второго состояния в себя должно вырабатываться выходное воздействие z_5, на переходе по событию *M* из второго состояния в третье должно вырабатываться выходное воздействие

z_4, на переходе по событию *H* из третьего состояния в четверное должно вырабатываться z_5, а на переходе по событию *T* из четвертого состояния во второе – z_3 (рис. 21).

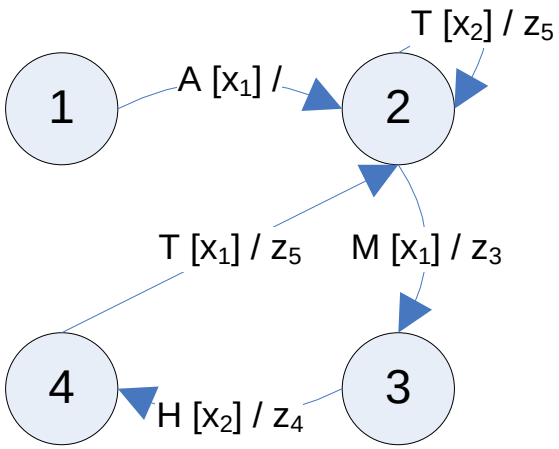


Рис. 21. Помеченные на основании теста переходы автомата

Таким образом, на основании одного теста можно расставить выходные воздействия на части переходов автомата. Если тестов больше одного, то возможны различные ситуации:

- может оказаться, что в разных тестах используется один и тот же переход, но на этих тестах при выборе этого перехода должны генерироваться одни и те же выходные воздействия;
- может оказаться, что в разных тестах используется один и тот же переход, но при его выборе должны генерироваться разные выходные воздействия (имеется противоречие).

В первом случае ситуация аналогична ситуации с одним тестом. Во втором случае этот переход помечается тем выходным воздействием, которое чаще всего вырабатывается на нем в тестах.

Этот же принцип можно распространить на случай, когда на переходе должно вырабатываться более одного выходного воздействия. 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.

2.1.5. Вычисление функции приспособленности

Функция приспособленности в этом случае основывается на редакционном расстоянии. Для ее вычисления выполняются следующие действия: на вход автомату подается каждая из последовательностей Input[i]. Обозначим как Output[i] последовательность выходных воздействий, которую сгенерировал автомат при входе Input[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 ED*(A, B)* – редакционное расстояние между последовательностями *A* и *B*. Отметим, что значение этой величины лежит в пределах от 0 до 1. При этом, чем «лучше» автомат соответствует тестам, тем больше ее значение.

Функция приспособленности зависит не только от того, насколько «хорошо» автомат работает на тестах, но и от числа переходов, которые он содержит. Она вычисляется по формуле: $FF_2 = \begin{cases} 10 \cdot FF_1 + \frac{1}{M} \cdot (M - cnt), FF_1 < 1 \\ 20 + \frac{1}{M} \cdot (M - cnt), FF_1 = 1 \end{cases}$,

где M – число, заведомо большее числа переходов в автомате (в экспериментах M выбиралось равным 100), а cnt – число переходов в автомате, кодируемом рассматриваемой особью.

Учет числа переходов в функции приспособленности необходим по двум причинам. Во-первых, минимизация числа переходов приводит к тому, что в результирующем автомате отсутствуют не используемые в тестах переходы. Во-вторых, чем меньше в автомате переходов, тем более «общее» поведение он задает. Таким образом, решается проблема переобучения (overfitting), состоящая в том, что автомат демонстрирует

правильное поведение только на входных последовательностях, соответствующих набору тестов.

Оценим время вычисления функции приспособленности. Для вычисления редакционного расстояния применяется алгоритм динамического программирования [10], время работы которого пропорционально произведению длин последовательностей, для которых оно вычисляется. На вход этому алгоритму подаются две последовательности $A = A_1 \ldots A_{|A|}$ и $B = B_1 \ldots B_{|B|}$. Он основан на заполнении таблицы ED, которая определяется следующим образом: элемент ED[i, j] равен редакционному расстоянию между последовательностями $A_1 \ldots A_i$ и $B=B_1 \ldots B_j$.

Таким образом, время вычисления функции приспособленности составляет $O(\sum_{i=1}^{n} |\mathrm{Output}[i]| \cdot |\mathrm{Answer}[i]|)$. Заметим также, что добавление в набор тестов их префиксов не увеличивает времени вычисления функции приспособленности, так как достаточно вычислить редакционное расстояние только для «самых больших» тестов, а для префиксов редакционное расстояние взять из таблицы, вычисляемой алгоритмом динамического программирования.

2.1.6. Операция мутации, использующаяся в методе спуска на основе случайных мутаций и в генетическом алгоритме

При выполнении операции мутации с заданной вероятностью (по умолчанию, она равна 0.05) выполняется одно из действий: изменение описания каждого из переходов и удаление или добавление перехода для каждого состояния.

При изменении описания перехода с равной вероятностью выполняется одно из следующих действий:

- Изменение состояния, в которое ведет переход (изменяется на случайно выбранное).

- Изменение события, по которому выполняется переход (изменяется на случайно выбранное).
- Изменение числа выходных воздействий, вырабатываемых на этом переходе (с равной вероятностью это число либо уменьшается на единицу, либо увеличивается на единицу, но при этом не может стать отрицательным или превзойти некоторое заданное ограничение).
- Изменение условия (изменяется на случайно выбранное из числа тех, которые встречаются в тестах вместе с событием, которым помечен переход).

2.1.7. Операция удаления дублированных и противоречивых переходов

Обозначим L_{in} список переходов, поступающих на вход операции удаления дублированных и противоречивых переходов, а список переходов, получающийся в результате этой операции, – L_{out}. Для его формирования выполняются следующие операции:

- по очереди рассматриваются переходы, входящие в список L_{in};
- очередной переход t добавляется в L_{out}, если в нем еще нет перехода, помеченного тем же событием, что и t, и охранным условием, имеющим общую выполняющую подстановку с охранным условием перехода t.

2.1.8. Операция мутации, использующаяся в эволюционной стратегии

Так как в эволюционной стратегии операции мутации должна быть устроена таким образом, чтобы в ее результате мог получиться любой элемент пространства поиска, то она выполняется следующим образом.

Выбирается случайным образом число x из геометрического распределения с вероятностью успеха, равной 0.5. После этого к

поступившей на вход особи (x + 1) раз применяется мутация, использующаяся в методе спуска и эволюционной стратегии.

2.1.9. Генетический алгоритм

В настоящем разделе описывается генетический алгоритм, предложенный в настоящей работе для построения управляющих конечных автоматов по тестам.

2.1.9.1. Генерация начального поколения

Начальное поколение заполняется автоматами, сгенерированными случайным образом. При этом все автоматы содержат одинаковое число состояний, из каждого их которых число переходов определяется случайным образом в диапазоне от нуля до числа событий, которые присутствуют в рассматриваемой системе со сложным поведением.

2.1.9.2. Формирование следующего поколения

В качестве основной стратегии формирования следующего поколения используется элитизм [2]. При обработке текущего поколения отбрасываются все особи, кроме нескольких наиболее приспособленных. Доля выживающих особей постоянна для каждого поколения и является одним из параметров алгоритма.

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

Кроме этого, если на протяжении достаточно большого числа поколений не происходит увеличения приспособленности, то применяются «малая» и «большая» мутации поколения. При «малой» мутации

`поколения ко всем особям, кроме 10% лучших, применяется оператор мутации. При «большой» мутации каждая особь либо мутирует, либо заменяется на случайно сгенерированную.

Число поколений до «малой» и «большой» мутации постоянно во время работы алгоритма, но может быть различным для разных его запусков.

2.1.10. Операция скрещивания

Скрещивание особей производится следующим образом. Обозначим как P1 и P2 «родительские» особи, а как S1 и S2 – особи-«потомки». Начальные состояния S1.is и S2.is имеют номер ноль. Как было отмечено выше, у всех особей нулевое состояние является начальным.

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].

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.

К получившимся в результате скрещивания автоматам S1 и S2 применяется операция удаления дублированных и противоречивых переходов.

2.1.11. Совместное использование генетического алгоритма, эволюционной стратегии и метода спуска на основе случайных мутаций

В процессе проведения вычислительных экспериментов с описанными выше алгоритмами была замечена следующая особенность эволюционной стратегии и метода спуска на основе случайных мутаций – в ряде запусков целевое значение функции приспособленности достигалось очень быстро – в процессе работы алгоритм рассматривал не более 10 000 особей. Как правило, это происходило тогда, когда сгенерированное случайным образом начальное решение обладало достаточно высоким значением функции приспособленности.

В то же время алгоритм генетического программирования в каждом из запусков использовал более 10 000 вычислений функции приспособленности. Это объясняется тем, что даже если у лучшей особи в поколении значение функции приспособленности близко к целевому, то в процессе ее «доведения» до оптимума будет проведено достаточно много вычислений функции приспособленности, так как она будет вычисляться не только для этой особи, но и для всех других особей, входящих в поколение.

Поэтому в число рассматриваемых алгоритмов были добавлены гибридные алгоритмы, в которых до достижения лучшей особью значения функции приспособленности в 95% от целевого используется генетический алгоритм, а далее полученная лучшая особь оптимизируется с помощью эволюционной стратегии или метода спуска на основе случайных мутаций.

2.2. Метод выполнения операции скрещивания с учетом поведения автоматов на обучающих примерах

Как и ранее, обозначим как P1 и P2 «родительские» особи, а как S1 и S2 – особи-«потомки». Начальные состояния S1.is и S2.is имеют номер ноль – как было отмечено выше, у всех особей нулевое состояние является начальным.

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.$T[i]$.

При использовании предлагаемого в настоящей диссертации *метода скрещивания с учетом поведения автоматов на обучающих примерах* [170] 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 нормированное редакционное расстояние между «правильным ответом» Answer и последовательностью Output выходных воздействий, генерируемой автоматом, $\frac{ED(Output[i], Answer[i])}{\max(|Output[i]|, |Answer[i]|)}$ минимально.
2. Список переходов S1.$T[i]$ формируется следующим образом: сначала в него копируются помеченные переходы из P1.$T[i]$, затем помеченные переходы из P2.$T[i]$, а затем непомеченные переходы из P1.$T[i]$.
3. Список переходов S2.$T[i]$ формируется следующим образом: сначала в него копируются помеченные переходы из P2.$T[i]$, затем помеченные переходы из P1.$T[i]$, а затем непомеченные переходы из P2.$T[i]$.

К получившимся в результате скрещивания автоматам S1 и S2 применяется операция удаления дублированных и противоречивых переходов.

2.3. Экспериментальное исследование методов построения управляющих автоматов по обучающим примерам на основе эволюционных алгоритмов

В настоящем разделе описываются результаты экспериментального исследования эволюционных алгоритмов для построения управляющих конечных автоматов [170]. Рассматриваются шесть алгоритмов:

- генетический алгоритм, в котором используется только традиционный метод выполнения операции скрещивания (далее обозначается ГА-1);
- метод спуска на основе случайных мутаций (МС);
- (1+1)-эволюционная стратегия (ЭС);
- генетический алгоритм, в котором используется только операция скрещивания на основе тестов (ГА-2);
- алгоритм, использующий совместно ГА-2 и метод спуска на основе случайных мутаций (ГА-2 + МС);
- алгоритм, использующий совместно ГА-2 и метод спуска на основе случайных мутаций (ГА-2 + ЭС).

2.3.1. Построение автомата управления часами с будильником

В настоящем разделе описывается результаты экспериментального исследования указанных эволюционных алгоритмов на задаче построения автомата управления часами с будильником, описанной в разд. 1.1.3.

2.3.1.1. Набор тестов

В систему тестов для построения автомата управления часами с будильником включены тесты, описывающие его работу во всех трех

режимах. Тесты, описывающие поведение часов с будильником в режиме работы «Будильник выключен», приведены в табл. 2. В этой таблице, если во входной последовательности для некоторого события не указано условие перехода, считается, что условием является тождественно истинная формула.

Таблица 2. Тесты для режима работы «Будильник выключен»

Тест	Комментарий
Input: T, T, T, T Answer: z_5, z_5, z_5, z_5	Описывает обработку часами события «Сработал таймер». При возникновении этого события текущее время должно быть увеличено на минуту.
Input: H, H, H, H Answer: z_1, z_1, z_1, z_1	Описывает обработку часами нажатия кнопки «H». При нажатии на эту кнопку число часов в текущем времени должно быть увеличено на единицу.
Input: M, M, M, M Answer: z_2, z_2, z_2, z_2	Описывает обработку часами нажатия кнопки «M». При нажатии на эту кнопку число минут в текущем времени должно быть увеличено на единицу.
Input: T, M, H, T, T, T, M, T, H, H, T, M Answer: z_5, z_2, z_1, z_5, z_5, z_5, z_2, z_5, z_1, z_1, z_5, z_2	Описывает обработку событий H, M и T в режиме «Будильник выключен».
Input: A, A, A, T, T, T, T Answer: z_7, z_5, z_5, z_5, z_5	После трех нажатий кнопки «A» часы должны находиться в режиме «Будильник выключен».
Input: A, A, A, H, H, H, H Answer: z_7, z_1, z_1, z_1, z_1	После трех нажатий кнопки «A» часы должны находиться в режиме «Будильник выключен».
Input: A, A, A, M, M, M, M	После трех нажатий кнопки «A» часы должны

Answer: z_7, z_2, z_2, z_2, z_2	находиться в режиме «Будильник выключен».

Тесты, описывающие поведение часов с будильником в режиме работы «Настройка будильника», приведены в табл. 3.

Таблица 3. Тесты для режима работы «Настройка будильника»

Тест	Комментарий
Input: A, T, T, T, T Answer: z_5, z_5, z_5, z_5	Описывает обработку часами события «Сработал таймер». При возникновении этого события текущее время должно быть увеличено на минуту.
Input: A, H, H, H, H Answer: z_3, z_3, z_3, z_3	Описывает обработку часами нажатия кнопки «H». При нажатии на эту кнопку число часов во времени срабатывания будильника должно быть увеличено на единицу.
Input: A, M, M, M, M Answer: z_4, z_4, z_4, z_4	Описывает обработку часами нажатия кнопки «M». При нажатии на эту кнопку число минут во времени срабатывания будильника должно быть увеличено на единицу.
Input: A, T, M, H, T, T, T, M, T, H, H, T, M Answer: z_5, z_4, z_3, z_5, z_5, z_5, z_4, z_5, z_3, z_3, z_5, z_4	Описывает обработку событий H, M и T в режиме «Настройка будильника».
Input: A, A, A, A, T Answer: z_7, z_5	После четырех нажатий кнопки «A» часы должны находиться в режиме «Настройка будильника». Описывается обработка срабатывания таймера.

Input: A, A, A, A, T, T, T, T Answer: z_7, z_5, z_5, z_5, z_5	После четырех нажатий кнопки «A» часы должны находиться в режиме «Настройка будильника». Описывается обработка нескольких срабатываний таймера.
Input: A, A, A, A, H Answer: z_7, z_3	После четырех нажатий кнопки «A» часы должны находиться в режиме «Настройка будильника». Описывается обработка нажатия кнопки «H».
Input: A, A, A, A, H, H, H, H Answer: z_7, z_3, z_3, z_3, z_3	После четырех нажатий кнопки «A» часы должны находиться в режиме «Настройка будильника». Описывается обработка нескольких нажатий кнопки «H».
Input: A, A, A, A, M Answer: z_7, z_4	После четырех нажатий кнопки «A» часы должны находиться в режиме «Настройка будильника». Описывается обработка нажатия кнопки «M».
Input: A, A, A, A, M, M, M, M Answer: z_7, z_4, z_4, z_4, z_4	После четырех нажатий кнопки «A» часы должны находиться в режиме «Настройка будильника». Описывается обработка нескольких нажатий кнопки «M».

Тесты, описывающие поведение часов с будильником в режиме работы «Будильник включен», приведены в табл. 4.

Таблица 4. Тесты для режима работы «Будильник включен»

Тест	Комментарий
Input: A, A, H, H, H, H Answer: z_1, z_1, z_1, z_1	Описывает обработку часами нажатия кнопки «H». При нажатии на эту кнопку число часов в текущем времени должно быть увеличено

	на единицу.
Input: A, A, M, M, M, M Answer: z_2, z_2, z_2, z_2	Описывает обработку часами нажатия кнопки «M». При нажатии на эту кнопку число минут в текущем времени должно быть увеличено на единицу.
Input: A, A, T [!x_1 & !x_2] Answer: z_5	Описывает обработку часами события «Сработал таймер» при условии, что текущее время не совпадает со временем срабатывания будильника или со временем срабатывания будильника, увеличенными на минуту. Текущее время должно быть увеличено на минуту.
Input: A, A, T [!x_1 & !x_2], T [!x_1 & !x_2], T [!x_1 & !x_2] Answer: z_5, z_5, z_5	Расширение предыдущего теста.
Input: A, A, T [!x_1 & !x_2], T [x_1 & !x_2], T [x_2 & !x_1] Answer: z_5, z_5, z_6, z_5, z_7	Описывает обработку события «Сработал таймер» при различных значениях входных переменных.
Input: A, A, T [!x_1 & !x_2], T [x_1 & !x_2] Answer: z_5, z_5, z_6	Префикс предыдущего теста.
Input: A, A, T [!x_1 & !x_2], T [x_1 & !x_2], T [x_2 & !x_1], H Answer: z_5, z_5, z_6, z_5, z_7, z_1	Описывает обработку события «Сработал таймер» при различных значениях входных переменных, а также обработку нажатие кнопки «H».
Input: A, A, T [!x_1 & !x_2], T [x_1 & !x_2], T [x_2 & !x_1], M Answer: z_5, z_5, z_6, z_5, z_7, z_2	Описывает обработку события «Сработал таймер» при различных значениях входных переменных, а также обработку нажатие кнопки «M».

Input: A, A, T [!x_1 & !x_2], T [x_1 & !x_2], T [x_2 & !x_1], T [!x_1 & !x_2] Answer: z_5, z_5, z_6, z_5, z_7, z_5	Описывает обработку события «Сработал таймер» при различных значениях входных переменных.
Input: A, A, T [x_1 & !x_2] Answer: z_5, z_6	Описывает обработку события «Сработал таймер» при условии, что текущее время совпадает со временем срабатывания будильника.
Input: A, A, T [x_2 & !x_1] Answer: z_5, z_7	Описывает обработку события «Сработал таймер» при условии, что текущее время превышает на минуту время срабатывания будильника.
Input: A, A, T [x_1 & !x_2], T [x_2 & !x_1] Answer: z_5, z_6, z_5, z_7	Объединяет два предыдущих теста.
Input: A, A, T [!x_1 & !x_2], M, H, T [!x_1 & !x_2], T [!x_1 & !x_2], T [!x_1 & !x_2], M, T [!x_1 & !x_2], H, H, T [!x_1 & !x_2], M Answer: z_5, z_2, z_1, z_5, z_5, z_5, z_2, z_5, z_1, z_1, z_1, z_2	Описывает обработку событий H, M и T в режиме «Будильник включен».

Кроме тестов, описывающих поведение автомата в каждом из трех режимов, в набор тестов включены тесты, описывающие поведение автомата сразу в нескольких режимах. Эти тесты приведены в табл. 5.

Таблица 5. Тесты, описывающие поведение автомата в нескольких режимах

Тест	Комментарий
Input: A, A, T [!x_1 & !x_2], T [x_1 & !x_2], T [x_2 & !x_1], A, H Answer: z_5, z_5, z_6, z_5, z_7, z_7, z_1	Описывает поведение часов с будильником в двух режимах: «Будильник включен» и «Будильник выключен».
Input: A, A, T [!x_1 & !x_2], T [x_1 & !x_2], T [x_2 & !x_1], A, M Answer: z_5, z_5, z_6, z_5, z_7, z_7, z_2	Описывает поведение часов с будильником в двух режимах: «Будильник включен» и «Будильник выключен».
Input: A, A, T [!x_1 & !x_2], T [x_1 & !x_2], T [x_2 & !x_1], A, T Answer: z_5, z_5, z_6, z_5, z_7, z_7, z_5	Описывает поведение часов с будильником в двух режимах: «Будильник включен» и «Будильник выключен».
Input: A, A, T [!x_1 & !x_2], T [x_1 & !x_2], T [x_2 & !x_1], A, A, H Answer: z_5, z_5, z_6, z_5, z_7, z_7, z_3	Описывает поведение часов с будильником в двух режимах: «Будильник включен» и «Настройка будильника».
Input: A, A, T [!x_1 & !x_2], T [x_1 & !x_2], T [x_2 & !x_1], A, A, M Answer: z_5, z_5, z_6, z_5, z_7, z_7, z_4	Описывает поведение часов с будильником в двух режимах: «Будильник включен» и «Настройка будильника».
Input: A, A, T [!x_1 & !x_2], T [x_1 & !x_2], T [x_2 & !x_1], A, A, T Answer: z_5, z_5, z_6, z_5, z_7, z_7, z_5	Описывает поведение часов с будильником в двух режимах: «Будильник включен» и «Настройка будильника».
Input: A, A, T [!x_1 & !x_2],	Описывает поведение часов с

T [x_1 & !x_2], A, A, T Answer: z_5, z_5, z_6, z_7, z_5	будильником в двух режимах: «Будильник включен» и «Настройка будильника».

2.3.1.2. Описание вычислительных экспериментов

Для каждого алгоритма было проведено 1000 запусков со следующими параметрами алгоритмов:

- генетический алгоритм:
  - 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;
- эволюционная стратегия и метод спуска – перезапуск производился, если в течение 300 000 итераций не было улучшения значения функции приспособленности.

Цель в каждом из запусков каждого из алгоритмов состояла в том, чтобы построить автомат, содержащий 14 переходов и соответствующий всем тестам (значение функции приспособленности – 20.86). При этом в процессе работы каждого из алгоритмов рассматривались автоматы из четырех состояний.

Для каждого алгоритма было проведено 1000 запусков. Для каждого запуска каждого из алгоритмов записывалось число вычислений функции приспособленности.

2.3.1.3. Результаты вычислительных экспериментов

Приведем распределения времени работы алгоритмов, полученные в результате вычислительных экспериментов. В эволюционных алгоритмах

время работы измеряется числом запусков функции приспособленности, так как, как правило, ее вычисление является более трудоемким, чем выполнение операций мутации и скрещивания. Для анализа результатов вычислительных экспериментов использовался язык программирования *R* [147].

В результате каждого запуска каждого из алгоритмов был построен автомат, граф переходов изображен на рис. 22. Отметим, что этот граф переходов изоморфен приведенному в разд. 1.1.3.

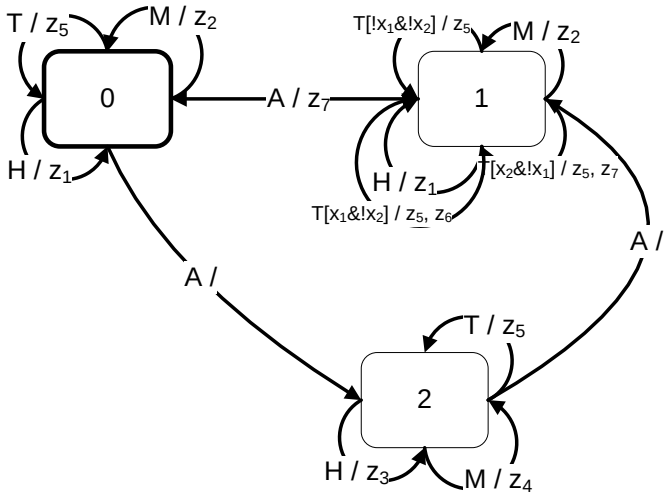


Рис. 22. Граф переходов автомата управления часами с будильником, построенного с помощью эволюционных алгоритмов

На рис. 23 приведена гистограмма распределения времени работы алгоритма ГА-1. Она была построена следующим образом: полученные в результаты вычислительных экспериментов значения были распределены по 30 корзинам одинакового размера. В каждой из них на гистограмме соответствует один столбец, высота которого пропорциональна числу элементов, попавших в эту корзину. Гистограммы, приведенные далее, построены по такому же принципу.

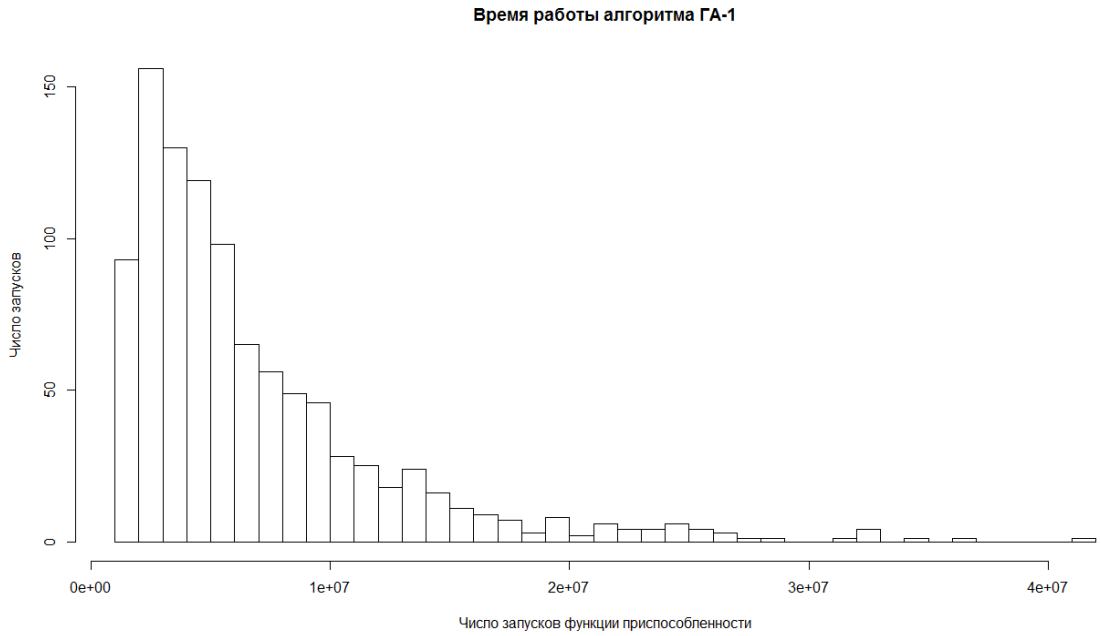


Рис. 23. Распределение времени работы алгоритма ГА-1

На рис. 24 приведена гистограмма распределения времен работы алгоритма МС.

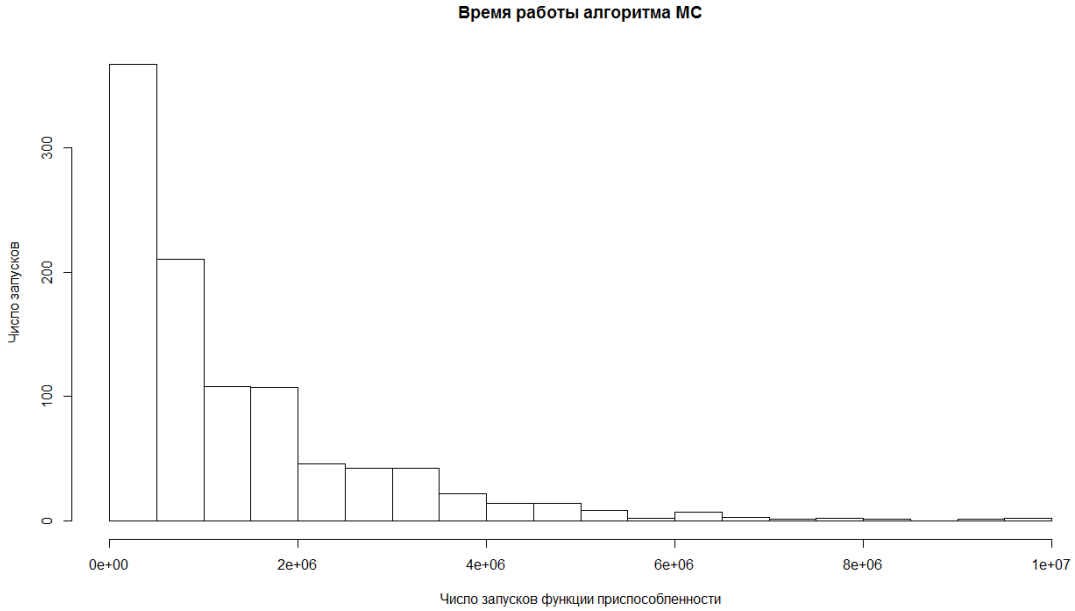


Рис. 24. Распределение времени работы алгоритма МС

На рис. 25 приведена гистограмма распределения времен работы алгоритма ЭС.

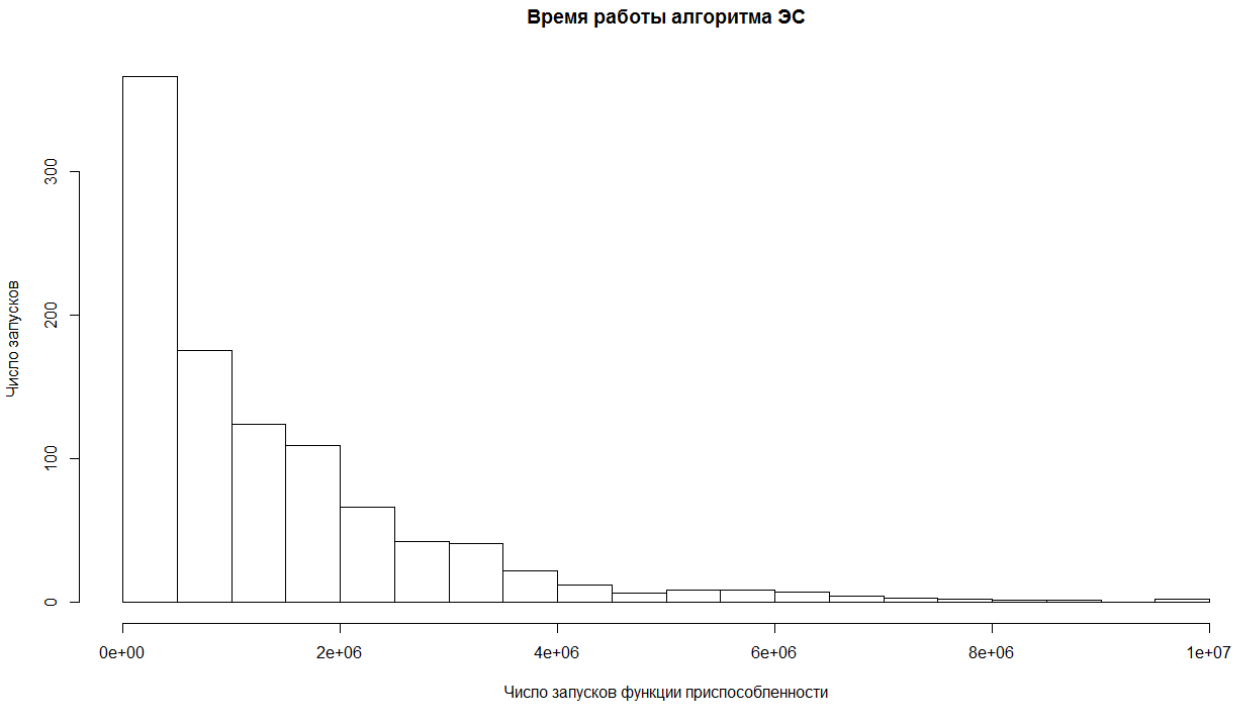


Рис. 25. Распределение времени работы алгоритма ЭС

На рис. 26 приведена гистограмма распределения времен работы алгоритма ГА-2.

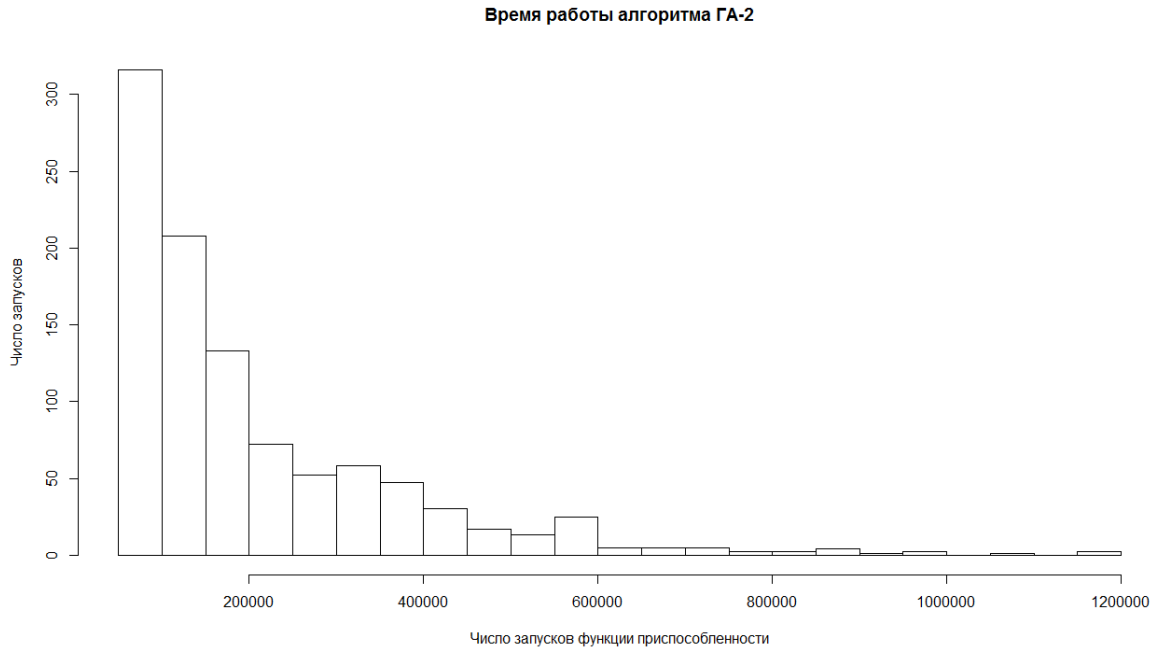


Рис. 26. Распределение времени работы алгоритма ГА-2

На рис. 27 приведена гистограмма распределения времен работы алгоритма ГА-2 + МС.

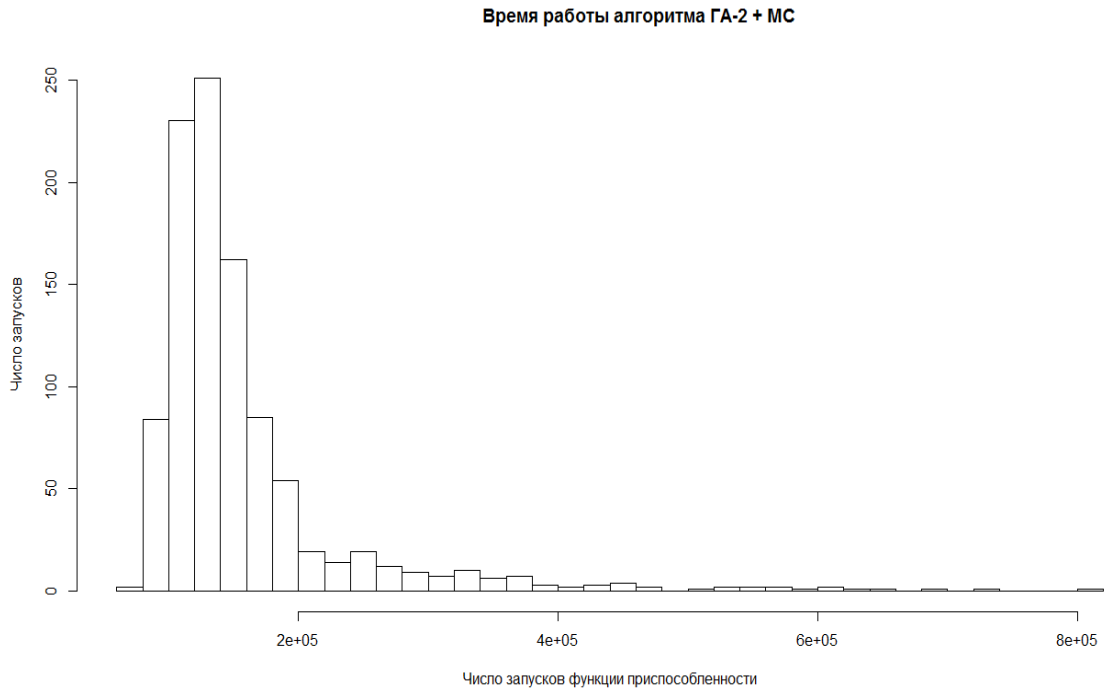


Рис. 27. Распределение времени работы алгоритма ГА-2 + МС

На рис. 28 приведена гистограмма распределения времен работы алгоритма ГА-2 + ЭС.

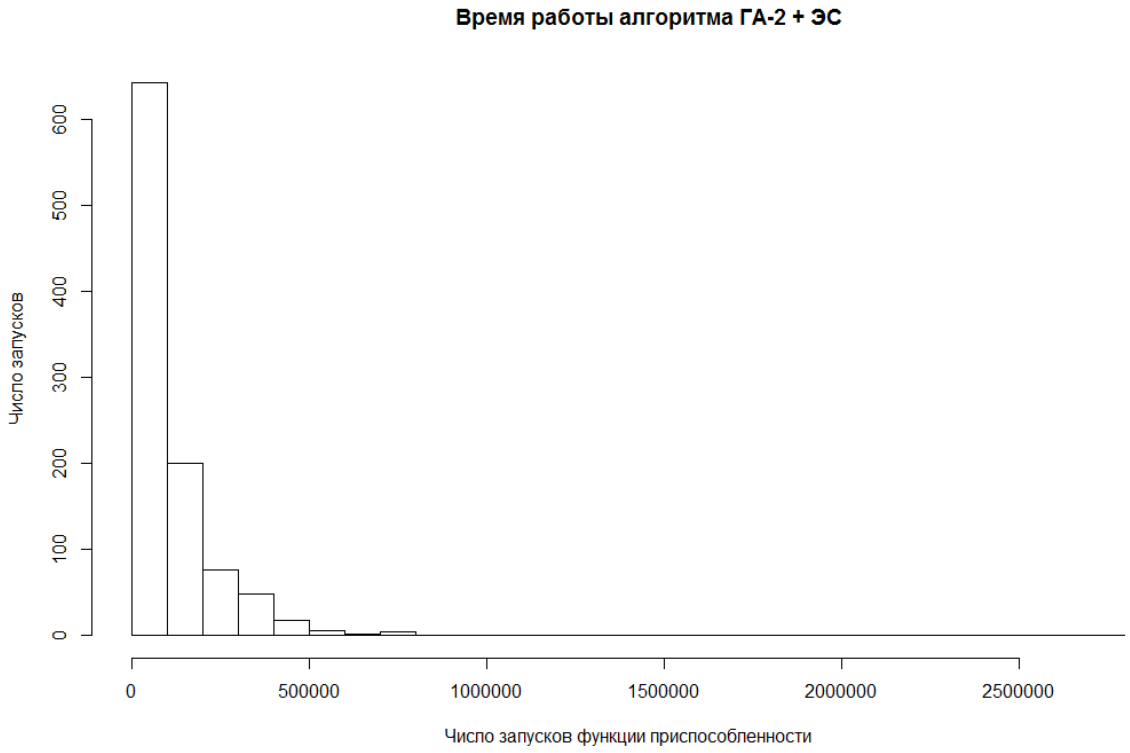


Рис. 28. Распределение времени работы алгоритма ГА-2 + ЭС

В табл. 6 приведены минимальные, максимальные, средние и медианные значения числа вычислений функции приспособленности для каждого из алгоритмов.

Таблица 6. Результаты вычислительных экспериментов

Алгоритм	Минимум	Максимум	Среднее	Медиана
ГА-1	1093938	41794531	6783215	5014202
МС	1387	9710090	1275439	792481
ЭС	1325	9915947	1317674	901615
ГА-2	51977	1196233	205451.6	142013
ГА-2 + МС	46311	780469	127712.2	103904
ГА-2 + ЭС	38458	2714324	129330.4	84778

Для каждой пары алгоритмов был проведен статистический тест ANOVA [142]. Результаты этого теста приведены в табл. 7. В ней для каждой пары алгоритмов указано *p*-значение (вычисленная с использованием указанного теста вероятность того, что среднее число вычислений функции приспособленности одинаково для соответствующей пары алгоритмов). Если эта вероятность меньше 0.05, то можно сделать вывод о том, что время работы соответствующих алгоритмов на рассматриваемой задаче существенно различается.

Таблица 7. Результаты статистического теста

	ГА-1	**МС**	**ЭС**	**ГА-2**	**ГА-2 + МС**	**ГА-2 + ЭС**
ГА-1	-	$<3\cdot10^{-16}$	$<3\cdot10^{-16}$	$<3\cdot10^{-16}$	$<3\cdot10^{-16}$	$<3\cdot10^{-16}$
МС	-	-	0.5178	$<3\cdot10^{-16}$	$<3\cdot10^{-16}$	$<3\cdot10^{-16}$
ЭС	-	-	-	$<3\cdot10^{-16}$	$<3\cdot10^{-16}$	$<3\cdot10^{-16}$
ГА-2	-	-	-	-	$<3\cdot10^{-16}$	$<3\cdot10^{-16}$
ГА-2 + МС	-	-	-	-	-	0.7446
ГА-2 + ЭС	-	-	-	-	-	-

По результатам статистического теста можно сказать, что алгоритмы можно разбить на четыре группы:

- в первую входят ГА-2 + ЭС и ГА-2 + МС;
- во вторую – ГА-2;
- в третью – МС и ЭС;
- в четвертую – ГА-1.

Указанное разбиение на группы обладает тем свойством, что алгоритмы внутри одной группы имеют статистически неразличимое время работы, а для алгоритмов из разных групп оно существенно различается.

На основании проведенных вычислительных экспериментов можно сделать вывод о том, что применение метода скрещивания с учетом поведения автоматов на обучающих примерах позволяет существенно повысить скорость работы генетического алгоритма. Этот вывод обосновывается тем, что генетический алгоритм, не использующий этот метод выполнения операции скрещивания, выполняет построение автоматов существенно медленнее по сравнению с эволюционной стратегией и методом спуска на основе случайных мутаций, а генетический алгоритм, использующий этот метод выполнения операции скрещивания, работает существенно быстрее этих двух алгоритмов.

2.3.2. Тесты, сгенерированные случайным образом

Для проведения экспериментов по построению управляющих автоматов по тестам, сгенерированным случайным образом, были сгенерированы автоматы, содержащие 4, 5, …, 10 состояний. Далее по каждому из автоматов был сгенерирован случайным образом набор тестов (при этом суммарная длина входных последовательностей в этих тестах составляла $150k$ для автомата из k состояний).

Число событий в генерируемых автоматах было равно двум, число выходных воздействий – двум, наибольшее число воздействий на одном

переходе также равно двум, охранные условия зависели от единственной входной переменной. В каждом генерируемом автомате присутствовала половина возможных ребер. По каждому сгенерированному автомату генерировалось несколько случайных путей суммарной длиной в $150k$ для автомата.

Входные последовательности тестов формировались на основе последовательностей событий, соответствующих переходам в этих путях, а выходные последовательности – на основе последовательностей выходных воздействий.

Для каждого набора тестов проводилось 100 запусков каждого из алгоритмов, для каждого запуска записывалось число вычислений функции приспособленности, проведенных в рамках этого запуска.

В табл. 8 приведены средние значения времени работы алгоритмов при генерации автоматов по тестам, сгенерированным случайным образом.

Таблица 8. Среднее время работы алгоритмов на тестах, сгенерированных случайным образом

Число состояний	ГА-1	МС	ЭС	ГА-2	ГА-2 + МС	ГА-2 + ЭС
4	25514587	5432560	6017983	875514	629232	460564
5	62124022	10711175	13519507	2032536	1545070	1241844
6	161261630	29291065	36216142	4537179	3721738	2981650
7	376970651	80844608	66850953	10928249	8799660	6815845
8	881025771	144343577	179974240	27500471	20036385	13877897
9	2149301686	350141493	409724265	56380237	41708779	32884793
10	4496075865	830139535	924940477	131551924	108635507	85621418

В табл. 9 приведены минимальные значения времени работы алгоритмов при генерации автоматов по тестам, сгенерированным случайным образом.

Таблица 9. Минимальное время работы алгоритмов на тестах, сгенерированных случайным образом

Число состояний	ГА-1	МС	ЭС	ГА-2	ГА-2 + МС	ГА-2 + ЭС
4	4450666	22728	8702	226277	305653	153667
5	11125313	38674	23419	561985	761696	503085
6	29526605	42520	50715	1458957	2098487	1186526
7	70124676	154112	103496	3274814	4883189	2682360
8	157506999	376283	686229	8004707	11072585	5552896
9	425132279	677943	559753	16821591	24104822	12016433
10	881931482	1353674	1292637	33051714	55850889	31173813

В табл. 10 приведены максимальные значения времени работы алгоритмов при генерации автоматов по тестам, сгенерированным случайным образом.

Таблица 10. Максимальное время работы алгоритмов на тестах, сгенерированных случайным образом

Число состояний	ГА-1	МС	ЭС	ГА-2	ГА-2 + МС	ГА-2 + ЭС
4	128084670	25531049	33402011	3827994	2267119	1893812
5	327022223	63827050	72560219	11539775	4408312	7748436
6	581028240	190639172	176047012	13932578	12708223	17840922
7	2027627902	359547552	320105572	45892343	31137911	44186639
8	3279139457	816989660	986060034	116893304	94516506	61305365
9	12036694052	1305740341	2855806080	284066504	132115726	150696001
10	20575772375	5384346929	6346239622	574414101	362471791	505240679

В табл. 11 приведены максимальные значения времени работы алгоритмов при генерации автоматов по тестам, сгенерированным случайным образом.

Таблица 11. Медианное время работы алгоритмов на тестах, сгенерированных случайным образом

Число состояний	ГА-1	МС	ЭС	ГА-2	ГА-2 + МС	ГА-2 + ЭС
4	20036223	2635943	3909706	616509	543407	337541
5	46596249	6586319	8197656	1406818	1338371	871440
6	126837100	19305364	21307240	3417363	3207257	2118335
7	249924503	57797982	44591716	7615641	7845902	4985004
8	646687247	82266747	102373914	16662240	16838018	10380041
9	1455756119	249606313	234509937	36787191	37917625	24212496
10	3205917742	511224031	473184839	86252583	87076432	57412402

Результаты вычислительных экспериментов на тестах, сгенерированных случайным образом, подтверждают результаты экспериментов по построению автомата управления часами с будильником.

Таким образом, результаты вычислительных экспериментов показывают, что *применение метода скрещивания с учетом поведения автоматов на обучающих примерах позволяет существенно повысить скорость работы генетического алгоритма*.

2.4. Метод построения автоматов по обучающим примерам и темпоральным формулам на основе эволюционных алгоритмов и верификации

Настоящий раздел посвящен методу построения автоматов по обучающим примерам (тестам) и темпоральным формулам на основе эволюционных алгоритмов и верификации [163, 165 – 167].

Этот метод расширяет метод, описанный в разд. 2.1, за счет того, что использует верификацию на стадии вычисления функции

приспособленности. Для верификации используется инструментальное средство, предложенное в работе [6], однако при необходимости вместо него может применяться любое другое инструментальное средство, предназначенное для верификации автоматных программ.

2.4.1. Входные данные

Входные данные для метода построения автоматов по тестам и темпоральным формулам на основе эволюционных алгоритмов и верификации в дополнение к данным, описанным в разд. 2.1.1, содержат набор темпоральных формул на языке логики *LTL*.

2.4.2. Выходные данные

Выходные данные представляют собой описание автомата. Этот автомат должен содержать не более k состояний, удовлетворять каждому заданному тесту и каждой заданной темпоральной формуле, а также обладать свойством непротиворечивости. Если такой автомат не найден, то выдается соответствующее сообщение.

2.4.3. Представление конечного автомата в виде хромосомы эволюционного алгоритма

В методе построения автоматов по тестам и темпоральным формулам на основе эволюционных алгоритмов и верификации используется такое же представление автомата в виде хромосомы эволюционного алгоритма, как и в методе построения автоматов по тестам (разд. 2.1).

2.4.4. Вычисление функции приспособленности

Для вычисления функции приспособленности автомат, задаваемый рассматриваемой особью, запускается на всех тестах и проверяется на соответствие всем темпоральным формулам.

Функция приспособленности вычисляется по формуле $FF = FF_2 \cdot (1 + \frac{nf_1}{nf_2})$, где величина FF_2 определена в разд. 2.1.5. Здесь nf_2 – общее число темпоральных формул в спецификации, а nf_1 – число формул, которые выполняются для рассматриваемого автомата. При вычислении функции приспособленности считается, что в случае $nf_1 = nf_2 = 0$, величина $\frac{nf_1}{nf_2}$ также равна нулю. Таким образом, если число темпоральных формул равно нулю, то функция приспособленности совпадает с определенной в разд. 2.1.5.

Оценим, насколько увеличилось время вычисления функции приспособленности по сравнению с описанной ранее. Для выполнения верификации автомата, как было сказано выше, каждая *LTL*-формула преобразуется в автомат Бюхи.

Оценить время проверки автомата на соответствие одной *LTL*-формуле можно следующим образом. Если L – длина формулы, то соответствующий ей автомат Бюхи в худшем случае будет содержать $L \times 2^L$ состояний [55]. Следовательно, в худшем случае автомат-пересечение, который строится по автомату Бюхи и модели Крипке, соответствующей верифицируемому автомату, будет содержать $O(L \times 2^L \times k)$ состояний, где k – число состояний в верифицируемом автомате.

Для построения автоматов Бюхи по *LTL*-формулам применяется библиотека *LTL2BA* [146], которая использует ряд эвристик для уменьшения размера получаемого автомата Бюхи. Кроме этого, как правило, темпоральные формулы, задающие требования имеют не сложную структуру и относительно небольшую длину. Поэтому на практике размер автомата Бюхи редко превышает 100 состояний.

Преобразование темпоральных формул в автоматы Бюхи производится один раз – перед запуском эволюционного алгоритма.

Отметим также, что при верификации не всегда требуется целиком обходить автомат-пересечение. Если модель не удовлетворяет темпоральной формуле, то контрпример может быть найден задолго до обхода всего автомата. Кроме этого, часть состояний автомата-пересечения могут быть недостижима.

Из изложенного следует, что на практике время вычисления функции приспособленности с учетом результатов верификации несущественно увеличивается по сравнению с функцией приспособленности, учитывающей только тесты.

2.4.5. Операции мутации и скрещивания

Операция мутации выполняется так же, как описано в разд. 2.1.6, а операция скрещивания – как описано в разд. 2.2.

2.5. Экспериментальное исследование метода построения автоматов по обучающим примерам и темпоральным формулам

Настоящий раздел посвящен экспериментальному исследованию метода построения автоматов по обучающим примерам (тестам) и темпоральным формулам на основе эволюционных алгоритмов и верификации. Исследование проводилось на задаче построения автомата управления дверьми лифта.

Требования к этой системе управления можно неформально описать следующим образом. Двери лифта могут открываться и закрываться. Если при закрытии дверей, они наткнулись на препятствие, то требуется остановить закрытие и открыть двери. Кроме этого, лифт может сломаться в процессе открытия или закрытия дверей, и тогда необходим звонок в аварийную службу.

Система управления дверьми лифта содержит пять входных событий:

- e_{11} – нажата кнопка «Открыть двери»;
- e_{12} – нажата кнопка «Закрыть двери»;
- e_2 – двери успешно открыты или закрыты;
- e_3 – препятствие мешает закрытию дверей;
- e_4 – двери лифта сломались.

Эта система имеет три выходных воздействия:

- z_1 – начать открытик дверей;
- z_2 – начать закрытие дверей
- z_3 – звонок в аварийную службу.

Поведение дверей лифта может быть описано конечным автоматом (рис. 29), построенным вручную в работе [125].

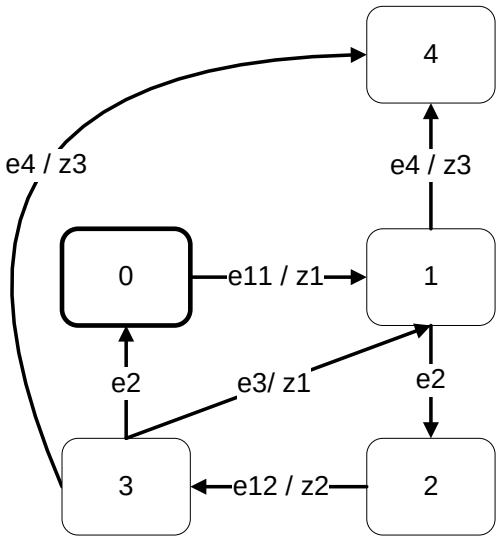


Рис. 29. Граф переходов автомата управления дверьми лифта, построенного на основе тестов и *LTL*-формул

Начальное состояние имеет номер «0» и выделено на рисунке жирной линией.

2.5.1.1. Набор тестов и темпоральных формул

Число тестов для построения автомата управления дверьми лифта равно девяти. Они описывают различные варианты поведения дверей лифта и представлены в табл. 12.

Таблица 12. Тесты для автомата управления дверьми лифта

Тест	Комментарий
Input: e_{11}, e_2, e_{12}, e_2 Answer: z_1, z_2	Описывает ситуацию открытия и закрытия дверей лифта.
Input: e_{11}, e_2, e_{12}, e_2, e_{11}, e_2, e_{12}, e_2 Answer: z_1, z_2, z_1, z_2	Описывает процесс открытия и закрытия дверей лифта дважды.
Input: e_{11}, e_2, e_{12}, e_3, e_2, e_{12}, e_2 Answer: z_1, z_2, z_1, z_2	Описывает открытие дверей и появления препятствия при закрытии. Дверь закрывается со второго раза.
Input: e_{11}, e_2, e_{12}, e_2, e_{11}, e_2, e_{12}, e_3, e_2, e_{12}, e_2 Answer: z_1, z_2, z_1, z_2, z_1, z_2	Открытие и закрытие дверей лифта, закрытие, снова открытие, и при закрытии появление препятствия.
Input: e_{11}, e_2, e_{12}, e_3, e_2, e_{12}, e_3, e_2, e_{12}, e_2 Answer: z_1, z_2, z_1, z_2, z_1, z_2	Процесс открытия дверей, при закрытии появление препятствия, вторая попытка закрытия и опять препятствие мешает закрыть дверь.
Input: e_{11}, e_4 Answer: z_1, z_3	Описывает процесс возникновения поломки в процессе открытия дверей.
Input: e_{11}, e_2, e_{12}, e_4 Answer: z_1, z_2, z_3	Описывает процесс возникновения поломки в процессе закрытия дверей.
Input: e_{11}, e_2, e_{12}, e_2, e_{11}, e_4 Answer: z_1, z_2, z_1, z_3	Описывает процесс возникновения поломки во время второго открытия дверей лифта.
Input: e_{11}, e_2, e_{12}, e_3, e_4 Answer: z_1, z_2, z_1, z_3	Описывает процесс возникновения поломки в момент открытия дверей из-за препятствия.

Кроме тестов, использовались 11 темпоральных формул. Их описание приведено в табл. 13.

В темпоральных формулах используются следующие предикаты:

- wasEvent(e) – на вход автомату поступило событие e;
- wasAction(z) – автомат выработал выходное воздействие z.

Таблица 13. Темпоральные формулы для автомата управления дверьми лифта

Формула	Комментарий
G(wasEvent(e_{11}) => wasAction(z_1))	Если на вход автомату поступило событие e_{11}, то автомат выработает выходное воздействие z_1. В терминах системы управления дверьми лифта утверждение можно записать следующим образом: при обработке события «Нажата кнопка «Открыть двери» обязательно будет начато открытие дверей.
G(wasEvent(e_{12}) <=> wasAction(z_2))	Выходное воздействие z_2 вырабатывается тогда и только тогда, когда на вход автомату поступает событие e_{12}. В терминах системы управления дверьми лифта утверждение можно записать следующим образом: при нажатии кнопки «Закрыть двери» начато их закрытие, и закрытие дверей может начаться только при нажатии кнопки «Закрыть двери».

G(wasEvent(e_4) <=> wasAction(z_3))	Выходное воздействие z_3 генерируется тогда и только тогда, когда на вход автомату поступает событие e_4. В терминах системы управления дверьми лифта утверждение можно записать следующим образом: звонок в аварийную службу будет произведен тогда и только тогда, когда лифт сломается.
G(wasEvent(e_3) => wasAction(z_1))	Если на вход автомату поступило событие e_3, то будет выработано выходное воздействие z_1. В терминах системы управления дверьми лифта утверждение можно записать следующим образом: если препятствие мешает закрыть двери, то они начнут открываться.
G(wasEvent(e_2) => X[wasEvent(e_{11}) || wasEvent(e_{12})])	После события e_2 следующим обработанным событием будет e_{11} или e_{12}. В терминах системы управления дверьми лифта утверждение можно записать следующим образом: если дверь успешно открылась или закрылась, то следующее обработанное событие может быть либо «Нажата кнопка «Открыть двери» или «Нажата кнопка «Закрыть двери».

G(wasEvent(e_{11}) => X[wasEvent(e_4) || wasEvent(e_2)])	Если на вход автомату поступило событие e_{11}, то следующим обработанным событием будет e_4 или e_2. В терминах системы управления дверьми лифта утверждение можно записать следующим образом: если была нажата кнопка «Открыть двери», то двери лифта либо успешно закроются, либо сломаются.
G(wasAction(z_1) => X[wasEvent(e_2) || wasEvent(e_4)])	Если автоматом было выработано выходное воздействие z_1, то следующим обработанным событием будет e_2 или e_4. В терминах системы управления дверьми лифта утверждение можно записать следующим образом: если дверь начала закрываться, то либо она успешно откроется, либо сломается.
G(wasEvent(e_{12}) => X[wasEvent(e_2) || wasEvent(e_3) || wasEvent(e_4)])	Если на вход автомату поступило событие e_{12}, то следующим обработанным событием будет или e_2, или e_3, или e_4. В терминах системы управления дверьми лифта утверждение можно записать следующим образом: если была нажата кнопка «Закрыть двери», то произойдет одно из трех: либо

	двери закроются, либо препятствие помешает закрыть двери, либо лифт сломается.
G(wasAction(z_1) => X[U(!wasAction(z_1), wasAction(z_2))])	Если автомат выработал выходное воздействие z_1, то оно не будет больше выработано, пока автомат не выработает z_2. В терминах системы управления дверьми лифта утверждение можно записать следующим образом: если было начато открытие дверей, то в следующий раз открытие может быть начато только после того, как они закроются.
G(wasAction(z_2) => X[U(!wasAction(z_2), wasAction(z_1)])	Если автомат выработал выходное воздействие z_2, то оно не будет больше выработано, пока автомат не выработает z_1. В терминах системы управления дверьми лифта утверждение можно записать следующим образом: если было начато закрытие дверей, то в следующий раз закрытие может быть начато только после того, как они откроются.
!F(wasEvent(e_4) & X(F(wasEvent(e_{11}) || wasEvent(e_{12}) || wasEvent(e_2) || wasEvent(e_3))))	После поступления e_4 на вход автомату не будут поступать события e_{11}, e_{12}, e_2 или e_3. В терминах системы управления

	дверьми лифта утверждение можно записать следующим образом: не верно, что после поломки лифта будут поступать события «Нажата кнопка «Открыть двери», «Нажата кнопка «Закрыть двери», «Двери успешно открыты или закрыты», «Препятствие мешает закрытию дверей».

2.5.1.2. Описание вычислительных экспериментов

В вычислительных алгоритмах использовался генетический алгоритм, использующий операцию скрещивания с учетом поведения автоматов на тестах. Рассматривались два варианта входных данных – первый содержал только тесты, а второй – и тесты, и темпоральные формулы.

Для каждого набора входных данных было проведено 1000 запусков со следующими параметрами алгоритма:

- 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.

Для каждого запуска записывалось число вычислений функции приспособленности.

2.5.1.3. Результаты экспериментов

В результате экспериментов, использующих только тесты в качестве входных данных, в 993 случаях из 1000 получался неправильный конечный автомат. Один из таких автоматов представлен на рис. 30, жирной линией выделено начальное состояние.

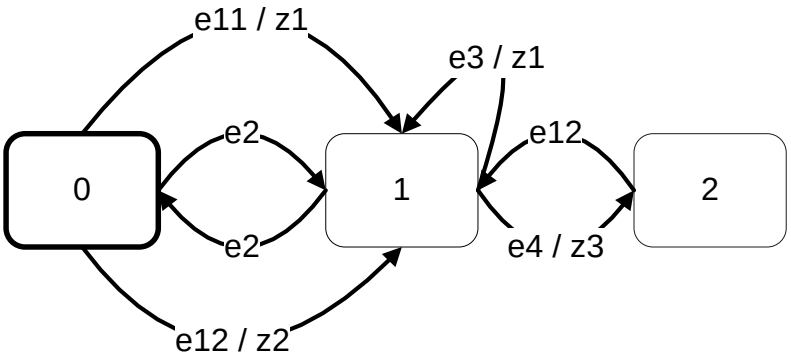


Рис. 30. Граф переходов автомата управления дверьми лифта, построенного только на основе тестов

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

Среднее значение числа вычислений функции приспособленности оказалось равным 7.479×10^4, минимальное число вычислений – 2.184×10^4, максимальное число – 2.999×10^5.

При использовании в качестве входных данных тестов и темпоральных формул в результате каждого из 1000 запусков алгоритма был построен автомат, изоморфный автомату, изображенному на рис. 29.

Среднее значение числа вычислений функции приспособленности оказалось равным 7.246×10^5, минимальное число вычислений – 7.054×10^4, максимальное число – 5.492×10^6.

Таким образом, число вычислений функции приспособленности для метода построения автоматов по тестам и темпоральным формулам на

основе эволюционных алгоритмов и верификации на задаче построения автомата управления дверьми лифта оказалось примерно в 10 раз больше, чем для метода построения автоматов по тестам на основе эволюционных алгоритмов. Однако применение верификации оправдывает себя, так как при ее использовании при каждом из 1000 запусков был построен автомат, удовлетворяющий не только тестам, но и всем темпоральным формулам.

Выводы по главе 2

1. Разработан метод построения управляющих автоматов по обучающим примерам на основе эволюционных алгоритмов. Его основное отличие от известных состоит в том, что в предлагаемые алгоритмы *добавлен новый шаг* «Расстановка выходных воздействий», который выполняется перед вычислением функции приспособленности.
2. Разработан *метод выполнения операции скрещивания* для генетических алгоритмов, учитывающий поведение автоматов на обучающих примерах. Показано, что генетический алгоритм, использующий разработанный метод выполнения операции скрещивания, осуществляет построение управляющих автоматов по обучающим примерам быстрее, чем генетический алгоритм, использующий традиционный метод выполнения операции скрещивания, эволюционная стратегия или метод спуска на основе случайных мутаций.
3. Разработан *метод построения автоматов по обучающим примерам и темпоральным формулам* на основе эволюционных алгоритмов и верификации. Его основное отличие от известных состоит в том, что для вычисления функции приспособленности совместно применяются обучающие примеры и метод *Model Checking*.

4. Проведено экспериментальное исследование, показывающее эффективность разработанных методов.

ГЛАВА 3. ТЕХНОЛОГИЯ И ИНСТРУМЕНТАЛЬНОЕ СРЕДСТВО ПОСТРОЕНИЯ УПРАВЛЯЮЩИХ КОНЕЧНЫХ АВТОМАТОВ НА ОСНОВЕ ЭВОЛЮЦИОННЫХ АЛГОРИТМОВ И ВЕРИФИКАЦИИ

Настоящая глава посвящена описанию предлагаемых в работе технологии построения управляющих конечных автоматов на основе эволюционных алгоритмов и верификации и инструментального средства для автоматизированного построения автоматов, поддерживающего эту технологию. Исходный код этого программного средства размещен в открытом доступе в сети Интернет по адресу: http://code.google.com/p/gabp/.

3.1. Технология построения управляющих конечных автоматов на основе эволюционных алгоритмов и верификации

Исходными данными для построения управляющих автоматов на основе эволюционных алгоритмов являются описания поставщиков событий и объектов управления (в общем случае их может быть несколько), а также спецификация программы. На основе описаний формируются списки входных переменных, выходных воздействий и событий. Далее по спецификации программы строится набор тестов, а, при необходимости, и темпоральных формул.

Каждый тест содержит входную последовательности Input[i], содержащую входные события и условия переходов из входных переменных, и эталонную последовательность выходных воздействий Answer[i].

К набору тестов предъявляются следующие требования. Во-первых, если система со сложным поведением, для управления которой строится автомат, имеет несколько режимов работы, то система тестов должна

содержать тесты для каждого из этих режимов и для переходов между ними. Во-вторых, тесты не должны противоречить друг другу – если входная последовательность в одном из тестов является префиксом входной последовательности в другом тесте, то и выходная последовательность из первого теста должна быть префиксом выходной последовательности из второго теста. В-третьих, набор тестов должен содержать все существующие в системе входные события, входные переменные и выходные воздействия.

При записи темпоральных формул необходимо учитывать, что, так как на момент их формирования известны только входные воздействия, входные переменные и выходные воздействия, то они не могут содержать утверждения о состояниях автомата.

На основе спецификации необходимо также задать максимальное число состояний в автомате и требуемое число переходов в нем. По числу переходов необходимо определить целевое значение функции приспособленности.

Тесты совместно с темпоральными формулами подаются на вход эволюционному алгоритму. Отметим, что эволюционный алгоритм одинаков для всех задач – для новой задачи необходим только новый набор тестов, входных и выходных воздействий и, возможно, параметров алгоритма – размера поколения, вероятности выполнения операции мутации и т. д. (рис. 31).

В качестве эволюционного алгоритма, как показывает приведенное выше экспериментальное исследование, рекомендуется использовать генетический алгоритм, использующий метод скрещивания с учетом поведения автомата на обучающих примерах.

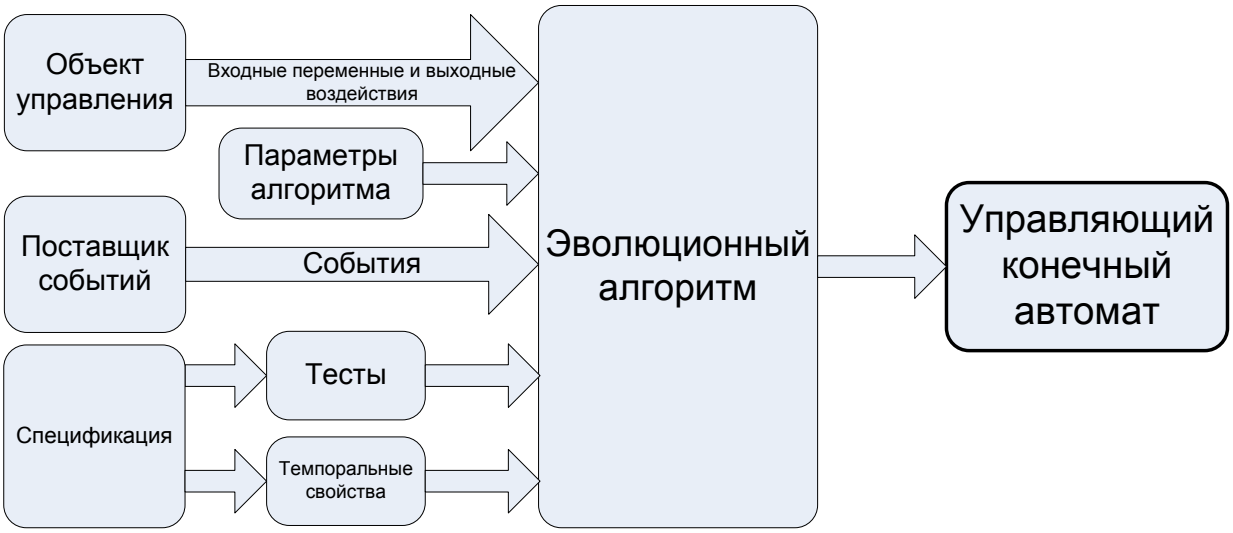


Рис. 31. Технология построения управляющих конечных автоматов на основе эволюционных алгоритмов и верификации

Если в результате работы эволюционного алгоритма был построен автомат, значение функции приспособленности для которого равно целевому или превышает его, то этот автомат соответствует спецификации. При любом изменении спецификации необходимо сначала проверить, соответствует ли ей уже построенный автомат. В случае несоответствия необходимо модифицировать подаваемые на вход эволюционному алгоритму тесты и темпоральные формулы (при их использовании) и выполнить построение автомата заново.

Если же в результате работы эволюционного алгоритма не удается построить автомат, значение функции приспособленности для которого равно целевому или превышает его, то в качестве результата работы алгоритма рассматривается сгенерированный алгоритмом автомат с наибольшим значением функции приспособленности. Значение функции приспособленности этого автомата может быть меньше целевого по трем причинам:

1. Он удовлетворяет всем тестам и темпоральным формулам, но имеет слишком большое число переходов.
2. Он удовлетворяет не всем тестам.
3. Он соответствует не всем темпоральным формулам.

В первом случае необходимо провести экспертную оценку построенного автомата на соответствие не формализуемым требованиям спецификации.

Во втором и третьем случаях требуется проверить корректность тестов и темпоральных формул, которым не удовлетворяет построенный автомат. Если в результате проверки будет определено, что в них содержатся ошибки, то необходимо их исправить и выполнить построение автомата заново. Если же ошибки в них найдены не будут, то необходимо заново выполнить построение автомата, увеличив максимальное число состояний.

3.2. Инструментальное средство для автоматизированного построения управляющих конечных автоматов

На основе разработанной технологии было создано инструментальное средство *GABP* (Genetic Automata-Based Programming), исходный код которого размещен в открытом доступе в сети Интернет по адресу: http://code.google.com/p/gabp/. На разработанное инструментальное средство было получено свидетельство о регистрации программы для ЭВМ (Первое свидетельство в Приложении 1).

3.2.1. Формат входных данных

Входные данные для этого инструментального средства задаются файлом в формате *XML*. Это файл описывает параметры алгоритма, наборы событий, входных переменных, выходных воздействий, а также тесты и темпоральные формулы. Приведем фрагмент входного файла для задачи построения автомата управления часами с будильником, рассмотренной в главе 2.

```
<program xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation = "resources/program.xsd">

   <parameters>
```

```
            <populationSize>2000</populationSize>
            <desiredFitness>20.86</desiredFitness>
            <stateNumber>4</stateNumber>
            <partStay>0.1</partStay>
            <timeSmallMutation>70</timeSmallMutation>
            <timeBigMutation>100</timeBigMutation>
            <mutationProbability>0.01</mutationProbability>
        </parameters>

        <inputSet>A, T, H, M, T [!x1 &amp; !x2], T [x1], T
[x2]</inputSet>
        <outputSet>z1, z2, z3, z4, z5, z6, z7</outputSet>

        <group>
            <tests>
                <test>
                    <input>T, T, T, T</input>
                    <output>z5, z5, z5, z5</output>
                </test>
                <test>
                    <input>A, T, T, T, T</input>
                    <output>z5, z5, z5, z5</output>
                </test>
                <test>
                    <input>A, A, A, T, T, T, T</input>
                    <output>z7, z5, z5, z5, z5</output>
                </test>
                <test>
                    <input>H, H, H, H</input>
                    <output>z1, z1, z1, z1</output>
                </test>
            ...
                <test>
                    <input>A, A, T [!x1 &amp; !x2]</input>
                    <output>z5</output>
                </test>
            </tests>
        </group>
    </program>
```

3.2.2. Формат выходных данных

Результат работы программного средства представляет собой описание конечного автомата в *XML*-формате инструментального средства *UniMod*.

Приведем пример выходного файла, описывающего автомат управления часами с будильником.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><!DOCTYPE model PUBLIC "-//eVelopers Corp.//DTD State machine model V1.0//EN" "http://www.evelopers.com/dtd/unimod/statemachine.dtd">
<model name="Model1">
  <controlledObject class="ru.ifmo.ControlledObjectStub" name="o1"/>
  <eventProvider class="ru.ifmo.EventPrividerStub" name="p1">
    <association clientRole="p1" targetRef="A1"/>
  </eventProvider>
  <rootStateMachine>
    <stateMachineRef name="A1"/>
  </rootStateMachine>
  <stateMachine name="A1">
    <configStore class="com.evelopers.unimod.runtime.config.DistinguishConfigManager"/>
    <association clientRole="A1" supplierRole="o1" targetRef="o1"/>
    <state name="Top" type="NORMAL">
      <state name="s0" type="NORMAL"/>
      <state name="s1" type="INITIAL"/>
      <state name="s2" type="NORMAL"/>
      <state name="s3" type="NORMAL"/>
    </state>
    <transition event="A" sourceRef="s0" targetRef="s2">
    </transition>
    <transition event="M" sourceRef="s0" targetRef="s0">
      <outputAction ident="o1.z4"/>
    </transition>
    <transition event="H" sourceRef="s0" targetRef="s0">
      <outputAction ident="o1.z3"/>
    </transition>
    <transition event="T" sourceRef="s0" targetRef="s0">
      <outputAction ident="o1.z5"/>
    </transition>
    <transition event="A" sourceRef="s1" targetRef="s0">
```

```
          </transition>
          <transition event="H" sourceRef="s1" targetRef="s1">
            <outputAction ident="o1.z1"/>
          </transition>
          <transition event="T" sourceRef="s1" targetRef="s1">
            <outputAction ident="o1.z5"/>
          </transition>
          <transition event="M" sourceRef="s1" targetRef="s1">
            <outputAction ident="o1.z2"/>
          </transition>
          <transition  event="T"  guard="!x1  &amp;  !x2"  sourceRef="s2"
targetRef="s2">
            <outputAction ident="o1.z5"/>
          </transition>
          <transition event="A" sourceRef="s2" targetRef="s1">
            <outputAction ident="o1.z7"/>
          </transition>
          <transition event="H" sourceRef="s2" targetRef="s2">
            <outputAction ident="o1.z1"/>
          </transition>
          <transition event="T" guard="x2" sourceRef="s2" targetRef="s2">
            <outputAction ident="o1.z5"/>
            <outputAction ident="o1.z7"/>
          </transition>
          <transition event="M" sourceRef="s2" targetRef="s2">
            <outputAction ident="o1.z2"/>
          </transition>
          <transition event="T" guard="x1" sourceRef="s2" targetRef="s2">
            <outputAction ident="o1.z5"/>
            <outputAction ident="o1.z6"/>
          </transition>
        </stateMachine>
      </model>
```

3.2.3. Структура программной реализации

Программная реализация инструментального средства выполнена на языке программирования *Java*. Основные классы, входящие в эту программную реализацию, приведены на рис. 32.

Класс `Starter` является точкой входа в программу, он использует класс `InputReader` для чтения входных данных, а также запускает эволюционный алгоритм (в текущей версии инструментального средства используется генетический алгоритм, описанный в разд. 2.1.9), который реализован в классе `GeneticAlgorithm`.

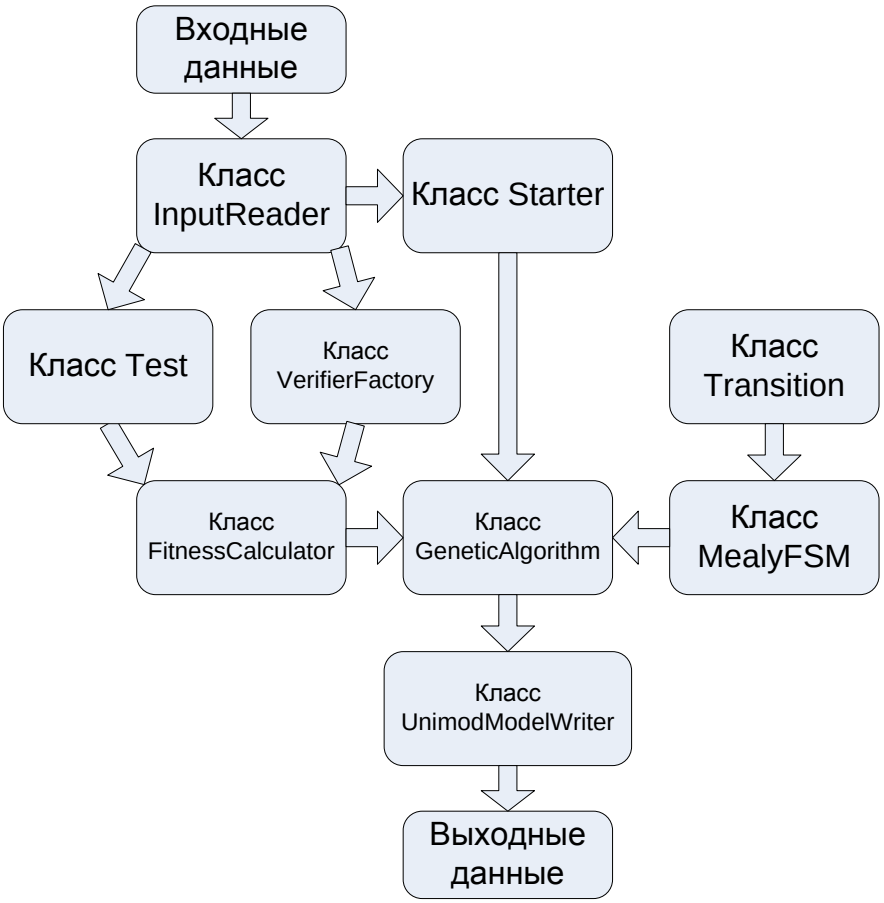


Рис. 32. Структура инструментального средства

Класс `InputReader` 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 формул, разбитых по группам.

Класс `VerifierFactory` предоставляет методы для преобразования темпоральных формул в автоматы Бюхи, для представления особи в виде автомата, принимаемого используемым

верификатором, и для верификации. Как было сказано выше, для преобразования в автоматы Бюхи используется библиотека *LTL2BA* [146].

Преобразование темпоральных формул в автоматы Бюхи происходит только один раз – при запуске инструментального средства, так как темпоральные формулы в процессе работы эволюционного алгоритма не изменяются. Этот класс, а также используемые им вспомогательные классы `ModifiableAutomataContext` и `AutomataTransition` основаны на исходном коде инструментального средства для верификации автоматных программ, описанного в работе [6].

В процессе работы генетический алгоритм использует особь (класс `MealyFSM`) и вычислитель функции приспособленности (класс `FitnessCalculator`). Особь, в свою очередь, представлена набором состояний (класс `Transition`), а при вычислении функции приспособленности используется набор тестов, каждый из которых представлен экземпляром класса `Test`.

Класс `MealyFSM` соответствует описанному выше представлению конечного автомата с действиями на переходах в виде особи эволюционного алгоритма, он хранит число состояний, набор переходов для каждого состояния, набор входных событий и набор выходных воздействий. В этом классе также реализованы операции мутации, скрещивания и алгоритм расстановки выходных воздействий. Этот класс реализует интерфейс `Individual`, соответствующий особи эволюционного алгоритма, таким образом, при необходимости в дальнейшем инструментальное средство может быть расширено за счет добавления других типов особей.

Класс `Transition` хранит событие, по которому выполняется данный переход, условие перехода, число выходных воздействий, которые должны на нем вырабатываться, а также номер состояния, в которое ведет этот переход. Кроме этого, он содержит массив, в который будут записаны

выходные воздействия в результате работы алгоритма расстановки выходных воздействий.

Класс `Test` хранит последовательность входных событий, представленную в виде набора строк, а также соответствующую ей последовательность выходных воздействий.

Класс `FitnessCalculator` реализует описанный выше метод вычисления функции приспособленности на основе тестов и темпоральных формул.

Класс `UnimodModelWriter` сохраняет особь в файл в *XML*-формате программного средства *UniMod*. В результате построенный с помощью разработанного инструментального средства автомат можно использовать в программах, написанных с помощью инструментального средства *UniMod*, при условии, что выполнена программная реализация поставщиков событий и объектов управления.

Выводы по главе 3

1. Разработана технология построения управляющих конечных автоматов по тестам и темпоральным формулам.
2. Разработано инструментальное средство для автоматизации построения управляющих конечных автоматов, поддерживающее предложенную технологию. На это программное средство получено свидетельство о регистрации программы для ЭВМ.

ГЛАВА 4. ВНЕДРЕНИЕ РЕЗУЛЬТАТОВ РАБОТЫ

Настоящая глава посвящена внедрению результатов работы.

4.1. Внедрение разработанных методов на примере построения автомата управления моделью беспилотного самолета

Настоящий раздел посвящен внедрению разработанных методов на примере построения автомата управления моделью беспилотного самолета [152, 162, 164]. В рамках этого внедрения было получено свидетельство о регистрации программы для ЭВМ (второе свидетельство в Приложении 1).

Для моделирования беспилотного самолета используется свободный кроссплатформенный симулятор *FlightGear* (http://www.flightgear.org). Его выбор обоснован тем, что он позволяет осуществлять как ручное, так и автоматное управление моделью беспилотного самолета. На рис. 33 представлен снимок экрана симулятора *FlightGear*.

Рис. 33. Снимок экрана симулятора *FlightGear*

Симулятор позволяет осуществлять сохранение параметров полета (скорость, направление полета и т. д.) и параметров самолета (положение руля, элеронов, состояние стартера и т. п.). Параметры полета являются

входными параметрами для системы управления, а параметры самолета – управляющими, так как за счет их изменений выполняется управление самолетом. Значения управляющих параметров формируются автоматом (рис. 34).

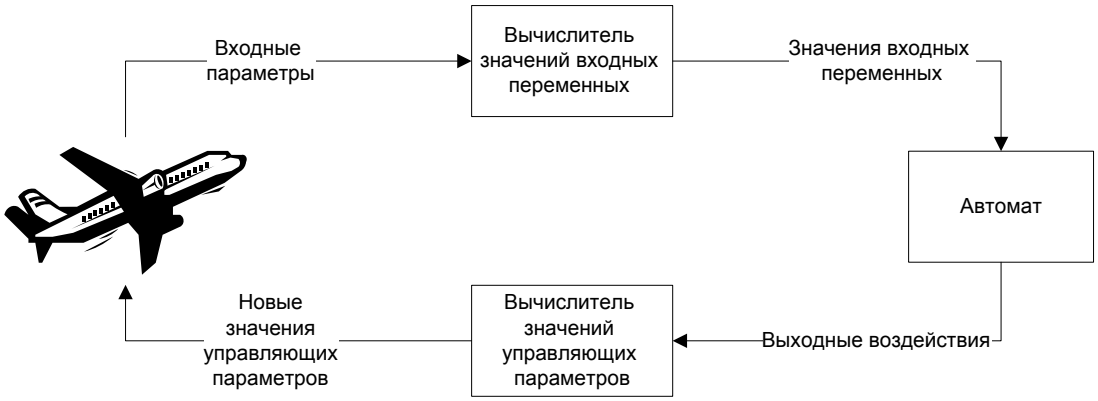


Рис. 34. Управление моделью беспилотного самолета с помощью автомата

4.1.1. Описание объекта управления

Объектом управления является модель беспилотного самолета. Объект управления характеризуется набором органов управления, с помощью воздействия на которые объектом можно управлять. Параметры, соответствующие органам управления, будем называть управляющими. Параметры некоторых органов управления могут принимать лишь конечное множество значений – *такие органы называются дискретными*. Параметры других органов управления характеризуются вещественными значениями – *такие органы называются непрерывными*. Будем также называть управляющие воздействия на непрерывные органы управления непрерывными, а управляющие воздействия на дискретные органы – дискретными.

Управляющими органами являются: магнето, стартер, дроссель, элероны, руль высоты и руль направления. Первые два органа являются дискретными, а остальные – непрерывными. При этом, например,

воздействие «включить стартер» является дискретным, а «повернуть руль на 0.5 град вправо» – непрерывным.

Непрерывное воздействие изменяет параметр органа управления на некоторую вещественную величину, а дискретное – устанавливает соответствующий орган управления в конкретное значение. Заметим, что последовательное выполнение действий с одним органом управления эквивалентно сумме воздействий на него в случае непрерывного воздействия и последнему воздействию – в случае дискретного.

Например, одним из непрерывных управляющих параметров является угол поворота руля самолета. Непрерывным воздействием на этот орган управления является *изменение* угла его поворота на некоторое значение. Тогда последовательность поворотов руля на x и y градусов эквивалентна повороту руля на $x + y$ градусов.

В свою очередь, примером дискретного исполнительного органа является стартер. Дискретное воздействие на него – включение или выключение. Тогда последовательность включений и выключений стартера эквивалентна последнему совершенному над ним действию.

4.1.2. Входные переменные и события

В качестве поставщика событий выступает модель беспилотного самолета. Будем считать, что генерируемый автомат является синхронным – все такты его работы одинаковы.

На каждом такте система управления получает значения всех входных параметров. После этого вычисляются логические значения всех условий объекта управления в порядке увеличения их номеров. После вычисления соответствующего значения условия генерируется соответствующее событие. Автомат в течение одного такта может совершить несколько переходов, и результирующее воздействие автомата в каждый момент времени t может быть составлено из нескольких последовательно выполненных воздействий на отдельных переходах.

Напомним, что каждый параметр результирующего воздействия – сумма воздействий в случае непрерывного параметра, и последнее воздействие – в случае дискретного.

Приведем список входных переменных:

- x_0 – двигатель включен;
- x_1 – ускорение изменения направления движения самолета больше нуля;
- x_2 – скорость изменения направления движения самолета больше нуля;
- x_3 – величина отклонения от начального направления меньше одного градуса;
- x_4 – величина отклонения от начального направления больше нуля (отклонение влево);
- x_5 – ускорение изменения крена (угла наклона) больше нуля;
- x_6 – скорость изменения крена больше нуля;
- x_7 – крен самолета меньше одного градуса;
- x_8 – крен больше нуля (самолет завалился на правый бок);
- x_9 – ускорение изменения вертикальной скорости самолета больше нуля;
- x_{10} – скорость изменения вертикальной скорости больше нуля;
- x_{11} – вертикальная скорость маленькая (меньше 0.1 м/с);
- x_{12} – вертикальная скорость больше нуля (самолет поднимается).

Набор событий содержит тринадцать событий e_0, …, e_{12}, при этом поступление события e_i означает, что закончено вычисление значения входной переменной x_i.

4.1.3. Набор обучающих примеров

Структура обучающего примера приведена на рис. 35. Обучающий пример T состоит из двух частей: $T.in$ и $T.ans$. Каждая из них является последовательностью длины $T.len$ – первая из которых состоит из значений входных параметров, а вторая – из соответствующих эталонных значений

управляющих параметров, которые *записываются в ходе экспериментов, проводимых человеком*.

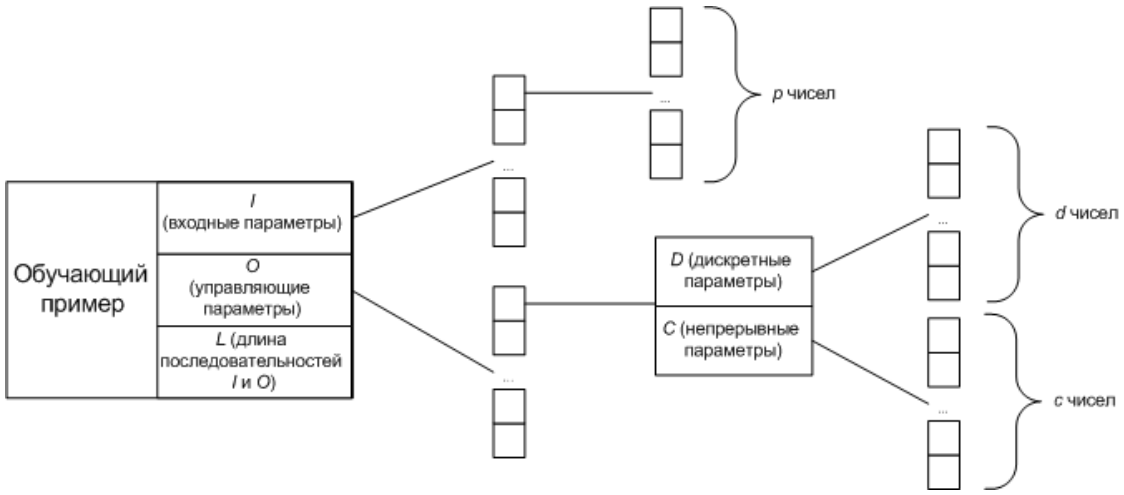


Рис. 35. Структура обучающего примера

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*.

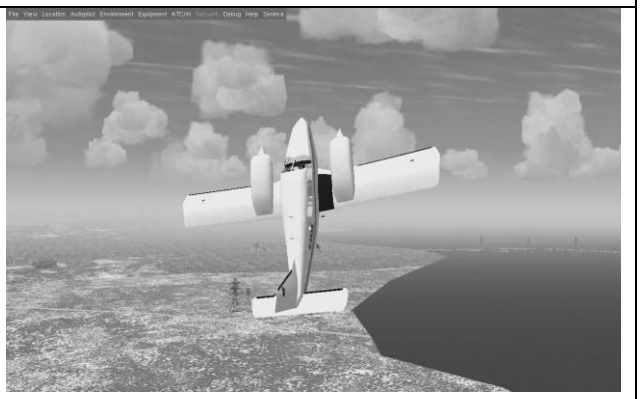
Было подготовлено 10 обучающих примеров, каждый из которых представляет собой запись выполнения «мертвой петли» под управлением пилота-человека.

На рис. 36, а – м приведены кадры видеозаписи одного из обучающих примеров.

а)

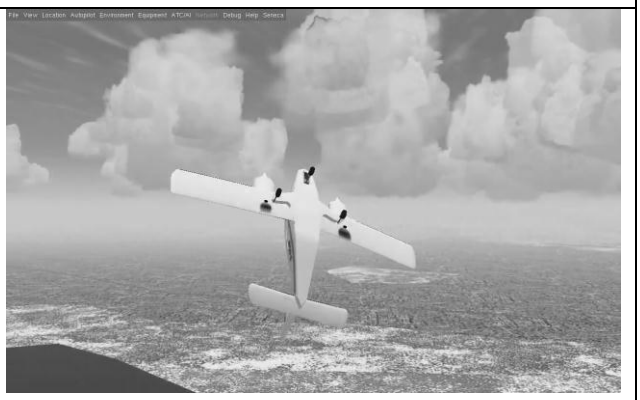
б)

в)



г)

д)



е)

ж)

з)

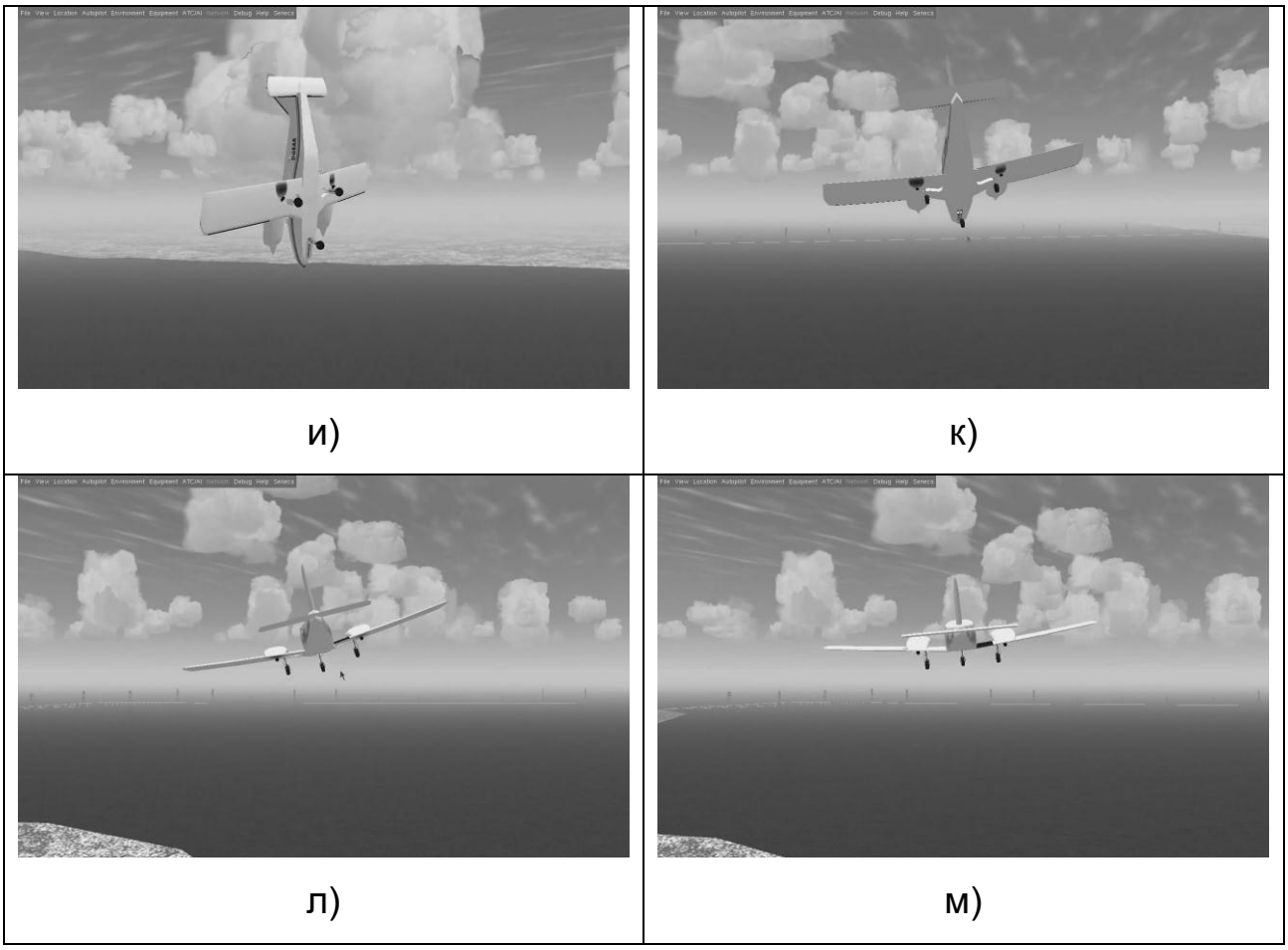


Рис. 36. Кадры видеозаписи одного из обучающих примеров

4.1.4. Вычисление функции приспособленности

Для вычисления значения функции приспособленности на вход автомата подается каждая из *n* последовательностей I_i – входной набор *i*-го обучающего примера и определяется последовательность значений управляющих параметров $\tilde{O}_i$, которую генерирует автомат. После этого вычисляется значение функции.

Так как в рассматриваемой задаче выходные воздействия могут быть не только дискретными, но и непрерывными, то был выбран следующий вид функции приспособленности

$$Fitness = 1 - \sqrt{\frac{1}{N}\sum_{i=1}^{n}\frac{\rho^2(O_i,\tilde{O}_i)}{\rho^2(O_i,0)}}.$$

Для упрощения дальнейших формул обозначим через A_i множитель $\frac{1}{\rho^2(O_i,0)}$, тогда предыдущее равенство примет вид

$$Fitness = 1 - \sqrt{\frac{1}{N}\sum_{i=1}^{n} A_i \cdot \rho^2(O_i, \tilde{O}_i)} \quad (1)$$

Здесь под $\rho(O_i,\tilde{O}_i)$ понимается «расстояние» между выходной и эталонной последовательностями управляющих параметров, которое вычисляется по формуле:

$$\rho(O_i,\tilde{O}_i) = \sqrt{\sum_{t=1}^{L_i}\left(\sum_{k=1}^{d}\left[D_{i,t,k} \neq \tilde{D}_{i,t,k}\right] + \sum_{k=1}^{c}\left(C_{i,t,k} - \tilde{C}_{i,t,k}\right)^2\right)}.$$

Как предлагалось выше, перед вычислением функции приспособленности к «каркасу» автомата применяется алгоритм расстановки выходных воздействий. Для построения автомата управления моделью беспилотного самолета этот алгоритм был модифицирован для учета непрерывных выходных воздействий.

4.1.5. Модифицировнный алгоритм расстановки выходных воздействий

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

При использовании этого алгоритма на переходах «скелета» автомата на его вход подается входная последовательность из каждого обучающего примера. При этом запоминаются переходы, совершаемые автоматом.

4.1.5.1. Случай одного обучающего примера

Для упрощения понимания предлагаемого подхода рассмотрим случай, когда обучающий набор состоит из одного обучающего примера T (n = 1). Тогда функция приспособленности приобретает вид

$$Fitness = 1 - \sqrt{A \cdot \rho^2(O, \tilde{O})}.$$

Здесь и дальше в этом пункте будем опускать первый нижний индекс со значением *1* для упрощения формул (вместо A_1 будем писать A, вместо O_1 – O и т.д.).

Функция приспособленности достигает максимума, когда квадрат расстояния $\rho^2(O,\tilde{O})$ минимален. После подстановки и изменения порядка суммирования получим:

$$\sum_{k=1}^{d}\sum_{t=1}^{L}\left[D_{t,k} \neq \tilde{D}_{t,k}\right] + \sum_{k=1}^{c}\sum_{t=1}^{L}\left(C_{t,k} - \tilde{C}_{t,k}\right)^2.$$

Заметим, что сумму для каждого параметра можно минимизировать независимо. Следовательно, достаточно минимизировать сумму:

$$\sum_{t=1}^{L}\left[D_{t,k} \neq \tilde{D}_{t,k}\right] \quad (2)$$

для каждого k от 1 до d и сумму:

$$\sum_{t=1}^{L}\left(C_{t,m} - \tilde{C}_{t,m}\right)^2 \quad (3)$$

для каждого m от 1 до c. Далее будем считать индексы k и m фиксированными.

Расстановка дискретных выходных воздействий. Пусть $\tilde{D}_{t,k}$ – это еще не определенное значение k-го дискретного параметра воздействия на последнем выполненном переходе в момент времени t. Поэтому выражение (2) можно переписать следующим образом:

$$\sum_{i=1}^{\mathrm{cnt}}\sum_{t\in\Psi_i}\left[D_{t,k} \neq \tilde{D}_{t,k}\right] \quad (4)$$

Здесь Ψ_i – множество значений моментов времени t, в которых номер последнего выполненного перехода был равен i, а cnt – число переходов в автомате. Для минимизации суммы (4) достаточно минимизировать сумму вида:

$$\sum_{t\in\Psi_i}\left[D_{t,k}\neq\tilde{D}_{t,k}\right]$$

для каждого i от 1 до n. Зафиксируем i. При $t\in\Psi_i$ значение $\tilde{D}_{t,k}$ равно значению k-го дискретного параметра на переходе i, которое не зависит от t. Обозначим его через u ($u=\tilde{D}_{t,k}$ при $t\in\Psi_i$). После введения этого обозначения, получим:

$$\sum_{t\in\Psi_i}\left[D_{t,k}\neq u\right].$$

Пусть $\tilde{\Psi}_{i,h}$ – это множество тех моментов времени t из Ψ_i, в которые $D_{t,k}$ равно v_h. Здесь v_h – h-е возможное значение k-го дискретного параметра. Заметим, что $\tilde{\Psi}_{i,h}$ – непересекающиеся подмножества, и при этом выполняется соотношение

$$\bigcup_{h=1}^{G}\tilde{\Psi}_{i,h}=\Psi_i.$$

Учитывая сказанное и обозначив как G число возможных значений k-го дискретного параметра, получим:

$$\sum_{h=1}^{G}\sum_{t\in\tilde{\Psi}_{i,h}}\left[u\neq D_{t,k}\right]=\sum_{h=1}^{G}\sum_{t\in\tilde{\Psi}_{i,h}}\left[u\neq v_h\right]=\sum_{h=1}^{G}\left[u\neq v_h\right]\sum_{t\in\tilde{\Psi}_{i,h}}1=\sum_{h=1}^{G}\left[u\neq v_h\right]\cdot\left|\tilde{\Psi}_{i,h}\right| \quad (5)$$

Выбрав в качестве значения дискретного параметра на i-м переходе $u=v_g$, получим значение суммы (5), равное $|\Psi_i|-|\tilde{\Psi}_{i,g}|$, так как все коэффициенты $[u\neq v_h]$ равны единице тогда и только тогда, когда $g\neq h$, а $\sum_{h=1}^{G}|\tilde{\Psi}_{i,h}|=|\Psi_i|$.

Поэтому для минимизации суммы (5) необходимо выбрать то значение v_g, при котором $\left|\tilde{\Psi}_{i,g}\right|$ максимально, – наиболее часто встречающееся значение среди $D_{t,k}$ при $t \in \Psi_i$. Если таких значений несколько, то можно выбрать любое из них.

Расстановка непрерывных выходных воздействий. Как упоминалось выше, величина изменения m-го непрерывного параметра равна сумме изменений этого параметра на всех выполненных переходах. При этом на каждом из этих переходов рассматриваемый параметр изменяется на определенную (постоянную для этого перехода), но неизвестную величину. Обозначим u_i величину изменения параметра на i-м переходе. Тогда окончательное значение параметра в момент времени t равно сумме его начального значения (равного нулю) и всех его изменений на каждом выполненном переходе:

$$\tilde{C}_{t,k} = \sum_{i=1}^{\mathrm{cnt}} \alpha_i[t] u_i \ .$$

Здесь $\alpha_i[t]$ – число выполнений i-го перехода к моменту времени t. Таким образом, сумму (3) можно переписать в виде:

$$S = \sum_{t=1}^{L} \left(C_{t,m} - \sum_{i=1}^{\mathrm{cnt}} \alpha_i[t] u_i \right)^2 .$$

Для того чтобы найти точку, в которой S принимает минимальное значение, необходимо найти точку, в которой частные производные S по u_j для всех j были равны нулю. Эти производные имеют вид

$$\frac{\partial S}{\partial u_j} = -\sum_{t=1}^{L} 2\alpha_j[t] \left(C_{t,m} - \sum_{i=1}^{\mathrm{cnt}} \alpha_i[t] u_i \right) = \sum_{t=1}^{L} 2\alpha_j[t] \left(\sum_{i=1}^{\mathrm{cnt}} \alpha_i[t] u_i - C_{t,m} \right).$$

После преобразования получим систему линейных уравнений:

$$\begin{cases} \sum_{i=1}^{\text{cnt}} \left(\sum_{t=1}^{L} \alpha_1[t]\alpha_i[t] \right) u_i = \sum_{t=1}^{L} C_{t,m}\alpha_1[t]; \\ \sum_{i=1}^{\text{cnt}} \left(\sum_{t=1}^{L} \alpha_2[t]\alpha_i[t] \right) u_i = \sum_{t=1}^{L} C_{t,m}\alpha_2[t]; \\ \ldots \\ \sum_{i=1}^{\text{cnt}} \left(\sum_{t=1}^{L} \alpha_n[t]\alpha_i[t] \right) u_i = \sum_{t=1}^{L} C_{t,m}\alpha_n[t]. \end{cases}$$

В данной работе эти *c* систем (для *m* от 1 до *c*) решаются методом Гаусса с целью получения искомых величин u_j изменения *m*-го параметра на переходе *j*.

Обобщим метод на случай нескольких обучающих примеров. При этом функция (1) достигает максимума, когда минимальна сумма

$$\sum_{i=1}^{n} A_i \cdot \rho^2(O_i, \tilde{O}_i).$$

Как и ранее, суммы по каждому параметру можно минимизировать независимо друг от друга. Получим, что необходимо минимизировать сумму

$$\sum_{i=1}^{n} A_i \sum_{t=1}^{L_i} \left[D_{i,t,k} \neq \tilde{D}_{i,t,k} \right] \quad (6)$$

для каждого *k* от 1 до *d* и сумму

$$\sum_{i=1}^{n} A_i \sum_{t=1}^{L_i} \left(C_{i,t,m} - \tilde{C}_{i,t,m} \right)^2 \quad (7)$$

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

Расстановка дискретных выходных воздействий. Выделим из суммы (6) слагаемые, соответствующие каждому переходу:

$$\sum_{i=1}^{n} A_i \sum_{j=1}^{\text{cnt}} \sum_{t \in \Psi_{i,j}} \left[D_{i,t,k} \neq \tilde{D}_{i,t,k} \right] = \sum_{j=1}^{\text{cnt}} \sum_{i=1}^{n} A_i \sum_{t \in \Psi_{i,j}} \left[D_{i,t,k} \neq \tilde{D}_{i,t,k} \right].$$

Здесь $\Psi_{i,j}$ – множество таких моментов времени *t*, при которых номер последнего выполненного перехода автомата, запущенного на обучающем

примере i, равен j, а n – число переходов в автомате. Таким образом, для каждого j от 1 до n необходимо минимизировать выражение вида

$$\sum_{i=1}^{n} A_i \sum_{t \in \Psi_{i,j}} \left[D_{i,t,k} \neq \tilde{D}_{i,t,k} \right].$$

Зафиксируем j. Как и в случае, когда обучающий пример состоит из одного обучающего примера, $\tilde{D}_{i,t,k}$ при $t \in \Psi_{i,j}$ равно значению k-го дискретного параметра на переходе j, которое не зависит от t. Снова обозначим его через u.

Пусть $\tilde{\Psi}_{i,j,h}$ – это множество тех моментов времени t из $\Psi_{i,j}$, при которых $D_{i,t,k}$ равно v_h. Как и раньше, v_h – h-е возможное значение k-го дискретного параметра, подмножества $\tilde{\Psi}_{i,j,h}$ являются непересекающимися, и выполняется соотношение

$$\bigcup_{h=1}^{G} \tilde{\Psi}_{i,j,h} = \Psi_{i,j}.$$

Обозначая как G число возможных значений этого параметра, получим:

$$\sum_{i=1}^{n} A_i \sum_{t \in \Psi_{i,j}} \left[D_{i,t,k} \neq u \right] = \sum_{i=1}^{n} A_i \sum_{h=1}^{G} \sum_{t \in \tilde{\Psi}_{i,j,h}} \left[D_{i,t,k} \neq u \right] =$$

$$= \sum_{i=1}^{n} A_i \sum_{h=1}^{G} \left[v_h \neq u \right] \sum_{t \in \tilde{\Psi}_{i,j,h}} 1 = \sum_{i=1}^{n} A_i \sum_{h=1}^{G} \left[u \neq v_h \right] \cdot \left| \tilde{\Psi}_{i,j,h} \right|.$$

Выбрав v_g в качестве значения дискретного параметра на j-м переходе (u=v_g) и подставив в предыдущую сумму, получим:

$$\sum_{i=1}^{n} A_i \sum_{h=1}^{G} \left[v_g \neq v_h \right] \cdot \left| \tilde{\Psi}_{i,j,h} \right| = \sum_{i=1}^{n} A_i \sum_{h=1}^{G} \left[g \neq h \right] \cdot \left| \tilde{\Psi}_{i,j,h} \right| = \sum_{i=1}^{n} A_i \left(\sum_{h=1}^{G} \left| \tilde{\Psi}_{i,j,h} \right| - \left| \tilde{\Psi}_{i,j,g} \right| \right) =$$

$$= \sum_{i=1}^{n} A_i \left(\left| \Psi_{i,j} \right| - \left| \tilde{\Psi}_{i,j,g} \right| \right) = \sum_{i=1}^{n} A_i \left| \Psi_{i,j} \right| - \sum_{i=1}^{n} A_i \left| \tilde{\Psi}_{i,j,g} \right| \quad (8)$$

Для минимизации суммы (8) необходимо выбрать то значение v_g, при котором

$$\sum_{i=1}^{n} A_i \left| \tilde{\Psi}_{i,j,g} \right|$$

максимально. Таким образом, на *j*-м переходе в качестве значения *k*-го дискретного параметра следует выбрать v_g, где

$$g = \arg\max_{g} \sum_{i=1}^{n} A_i \left| \tilde{\Psi}_{i,j,g} \right| .$$

Расстановка непрерывных выходных воздействий. Ввиду того, что величина изменения *m*-го непрерывного параметра в момент времени *t* ($\tilde{C}_{i,t,m}$) равна сумме изменений этого параметра на всех выполненных переходах к этому моменту, из формулы (7) получим

$$S = \sum_{i=1}^{n} A_i \sum_{t=1}^{L_i} \left(C_{i,t,m} - \sum_{j=1}^{\mathrm{cnt}} \alpha_{i,j}[t] u_j \right)^2 .$$

Здесь, как и раньше, u_j – неизвестная величина изменения *m*-го непрерывного параметра на *j*-м переходе, а $\alpha_{i,j}[t]$ – число выполнений *j*-го перехода к моменту времени *t* автомата, запущенного на обучающем примере *i*. Производная *S* по u_h имеет вид

$$\frac{\partial S}{\partial u_h} = \sum_{i=1}^{n} A_i \sum_{t=1}^{L_i} 2\alpha_{i,h}[t] \left(\sum_{j=1}^{\mathrm{cnt}} \alpha_{i,j}[t] u_j - C_{i,t,m} \right).$$

Приравняв все производные к нулю, для каждого непрерывного *m*-го параметра получим систему из *n* уравнений:

$$\begin{cases} \sum_{j=1}^{\text{cnt}} \left(\sum_{i=1}^{n} A_i \sum_{t=1}^{L_i} \alpha_{i,1}[t]\alpha_{i,j}[t] \right) u_j = \sum_{i=1}^{n} A_i \sum_{t=1}^{L_i} C_{i,t,m}\alpha_{i,1}[t]; \\ \sum_{j=1}^{\text{cnt}} \left(\sum_{i=1}^{n} A_i \sum_{t=1}^{L_i} \alpha_{i,2}[t]\alpha_{i,j}[t] \right) u_j = \sum_{i=1}^{N} A_i \sum_{t=1}^{L_i} C_{i,t,m}\alpha_{i,2}[t]; \\ \dots \\ \sum_{j=1}^{\text{cnt}} \left(\sum_{i=1}^{n} A_i \sum_{t=1}^{L_i} \alpha_{i,n}[t]\alpha_{i,j}[t] \right) u_j = \sum_{i=1}^{N} A_i \sum_{t=1}^{L_i} C_{i,t,m}\alpha_{i,n}[t]. \end{cases}$$

Каждая из этих систем решается методом Гаусса. Полученные u_j – искомые величины изменений рассматриваемого непрерывного m-го параметра на переходе j.

4.1.6. Результаты построения автомата

Было проведено 50 запусков генетического алгоритма, в каждом из которых выбирался автомат с наибольшим значением функции приспособленности. Полеты моделей самолетов, управляемых выбранными автоматами, были просмотрены экспертом.

После этого автомат, используемый в полете, больше других похожем на «идеальную мертвую петлю», был назван лучшим. В табл. 14 приведено описание переходов лучшего из сгенерированных автоматов для задачи выполнения «мертвой петли» (число состояний – 4, начальное состояние – 0, число дуг – 68).

Таблица 14. Описание лучшего из построенных автоматов

Дуга	Событие и условие	Дискретные выходные воздействия		Непрерывные выходные воздействия			
		Магнето	Стартер	Дроссель	Элероны	Руль высоты	Руль направления
0 -> 0	e_0 [not x_0]	0	0	1.00000	0.05239	0.01043	0.00000
0 -> 0	e_2 [not x_2]	0	0	0.00000	-0.01315	0.02117	0.00000
0 -> 0	e_4 [not x_4]	0	0	0.00000	0.00199	-0.09535	0.00000
0 -> 0	e_6 [not x_6]	0	0	0.00000	0.01336	-0.07833	0.00000

0 -> 0	e_8 [not x_8]	0	0	0.00000	0.00178	-0.10299	0.00000
0 -> 0	e_{11} [not x_{11}]	3	0	0.00000	-0.00178	0.01466	0.00000
0 -> 1	e_5 [not x_5]	0	0	0.00000	-0.01119	0.09089	0.00000
0 -> 1	e_9 [x_9]	0	0	0.00000	-0.01622	0.20543	0.00000
0 -> 1	e_9 [not x_9]	0	0	0.00000	0.00000	0.00000	0.00000
0 -> 2	e_1 [x_1]	0	0	0.00000	0.00217	0.01965	0.00000
0 -> 2	e_3 [x_3]	0	0	0.00000	0.02432	0.01225	0.00000
0 -> 2	e_5 [x_5]	0	0	0.00000	0.00000	0.00000	0.00000
0 -> 2	e_6 [x_6]	0	0	0.00000	0.00000	0.00000	0.00000
0 -> 2	e_{12} [not x_{12}]	3	0	0.00000	0.00018	-0.00087	0.00000
0 -> 3	e_4 [x_4]	0	0	0.00000	-0.01352	-0.08765	0.00000
0 -> 3	e_7 [not x_7]	0	0	0.00000	-0.00394	0.06189	0.00000
0 -> 3	e_{10} [not x_{10}]	0	0	0.00000	-0.00688	0.05064	0.00000
1 -> 0	e_0 [not x_0]	0	0	1.00000	0.05239	0.01043	0.00000
1 -> 0	e_2 [x_2]	0	0	0.00000	-0.02552	-0.01522	0.00000
1 -> 0	e_4 [x_4]	0	0	0.00000	0.00000	0.00000	0.00000
1 -> 0	e_{11} [x_{11}]	3	0	0.00000	0.00000	0.00000	0.00000
1 -> 1	e_1 [not x_1]	0	0	0.00000	-0.00132	0.00866	0.00000
1 -> 1	e_2 [not x_2]	0	0	0.00000	0.00000	0.00000	0.00000
1 -> 1	e_5 [x_5]	0	0	0.00000	-0.02809	-0.01434	0.00000
1 -> 1	e_7 [x_7]	0	0	0.00000	-0.00054	-0.00506	0.00000
1 -> 1	e_{12} [not x_{12}]	3	0	0.00000	-0.01224	-0.03468	0.00000
1 -> 2	e_0 [x_0]	0	0	0.00000	-0.00161	-0.00478	0.00000
1 -> 2	e_3 [x_3]	0	0	0.00000	0.00000	0.00000	0.00000
1 -> 2	e_3 [not x_3]	0	0	0.00000	-0.00274	-0.00202	0.00000
1 -> 2	e_6 [x_6]	0	0	0.00000	-0.01154	-0.11022	0.00000
1 -> 2	e_8 [x_8]	0	0	0.00000	-0.01082	-0.06436	0.00000
1 -> 2	e_9 [not x_9]	0	0	0.00000	-0.00993	-0.06966	0.00000
1 -> 3	e_5 [not x_5]	0	0	0.00000	-0.03879	-0.09682	0.00000
1 -> 3	e_{10} [not x_{10}]	0	0	0.00000	0.00172	0.03226	0.00000
1 -> 3	e_{12} [x_{12}]	3	0	0.00000	0.00497	-0.01437	0.00000
2 -> 0	e_1 [not x_1]	0	0	0.00000	0.03000	0.06766	0.00000
2 -> 0	e_3 [not x_3]	0	0	0.00000	0.00000	0.00000	0.00000
2 -> 0	e_9 [x_9]	3	0	0.00000	0.01939	0.04064	0.00000
2 -> 1	e_0 [x_0]	0	0	0.00000	0.02731	0.02049	0.00000

2 -> 1	e_0 [not x_0]	0	0	0.00000	0.02047	-0.02413	0.00000
2 -> 1	e_4 [x_4]	0	0	0.00000	0.00110	-0.00006	0.00000
2 -> 1	e_4 [not x_4]	0	0	0.00000	0.00000	0.00000	0.00000
2 -> 1	e_8 [x_8]	0	0	0.00000	0.01159	0.06208	0.00000
2 -> 2	e_2 [x_2]	0	0	0.00000	0.01880	0.00322	0.00000
2 -> 2	e_6 [x_6]	0	0	0.00000	-0.02062	-0.01551	0.00000
2 -> 2	e_7 [not x_7]	0	0	0.00000	0.00309	0.05025	0.00000
2 -> 2	e_{10} [not x_{10}]	0	0	0.00000	-0.00054	-0.01195	0.00000
2 -> 2	e_{11} [not x_{11}]	3	0	0.00000	0.01573	0.06745	0.00000
2 -> 2	e_{12} [not x_{12}]	3	0	0.00000	-0.00140	-0.00006	0.00000
2 -> 3	e_3 [not x_3]	0	0	0.00000	-0.00889	-0.04245	0.00000
2 -> 3	e_5 [x_5]	0	0	0.00000	0.00000	0.00000	0.00000
2 -> 3	e_7 [x_7]	0	0	0.00000	0.00239	0.03443	0.00000
2 -> 3	e_{11} [x_{11}]	0	0	0.00000	0.00000	0.00000	0.00000
3 -> 0	e_1 [x_1]	0	0	0.00000	0.00661	-0.00695	0.00000
3 -> 0	e_{10} [not x_{10}]	0	0	0.00000	0.01817	0.04933	0.00000
3 -> 1	e_3 [x_3]	0	0	0.00000	0.02187	0.03925	0.00000
3 -> 1	e_5 [x_5]	0	0	0.00000	0.00905	0.09217	0.00000
3 -> 1	e_6 [not x_6]	0	0	0.00000	0.00664	0.08602	0.00000
3 -> 1	e_{12} [not x_{12}]	3	0	0.00000	0.00556	-0.07512	0.00000
3 -> 2	e_4 [x_4]	0	0	0.00000	-0.01550	-0.09491	0.00000
3 -> 2	e_8 [x_8]	0	0	0.00000	-0.00498	0.01483	0.00000
3 -> 2	e_{10} [x_{10}]	0	0	0.00000	0.00000	0.00000	0.00000
3 -> 2	e_{11} [not x_{11}]	3	0	0.00000	0.00177	-0.03823	0.00000
3 -> 3	e_0 [not x_0]	0	0	1.00000	0.04394	0.11148	0.00000
3 -> 3	e_2 [not x_2]	0	0	0.00000	-0.02054	-0.17037	0.00000
3 -> 3	e_7 [not x_7]	0	0	0.00000	-0.00315	0.00256	0.00000
3 -> 3	e_9 [not x_9]	0	0	0.00000	-0.01064	0.00960	0.00000
3 -> 3	e_{12} [x_{12}]	3	0	0.00000	0.00757	0.00196	0.00000

Отметим, что для каждого состояния могут быть определены *2r* переходов, где *r* – число входных переменных (в рассматриваемом случае оно равно 13). Однако ни одно из четырех состояний не содержит их все.

Отсутствие перехода в таблице свидетельствует о том, что на этом переходе автомат сохраняет свое состояние и не вырабатывает никаких выходных воздействий.

Перед осуществлением «мертвой петли» двигатель включен. Поэтому в ходе ее выполнения в использовании стартера нет необходимости – столбец «стартер» содержит только нули. Этот орган управления не был исключен при генерации автоматов, так как возможны дополнительные обучающие примеры, в которых требуется переключение стартера (например, если двигатель может заглохнуть). Последний столбец также оказался нулевым, но и соответствующий ему орган управления может быть использован при применении других обучающим примеров.

В процессе наблюдения за ходом выполнения «мертвом петли» под управлением лучшего автомата было установлено, что в зависимости от параметров среды и самолета при запуске автомата возможны три варианта выполнения «петли»:

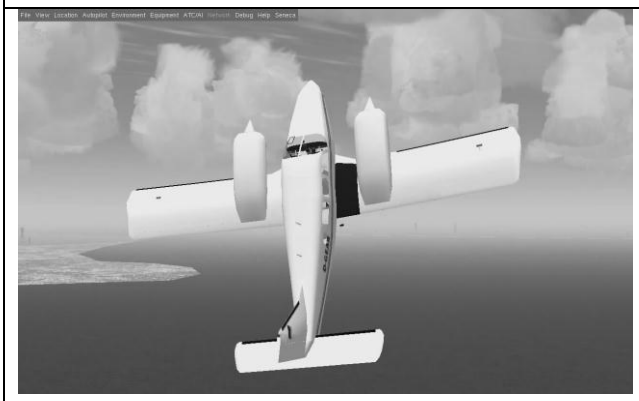
1. В большинстве случаев модель самолета, как и при управлении вручную, выполняет одну «мертвую петлю» и летит дальше.
2. Иногда модель самолета может выполнить несколько «мертвых петель» с некоторым интервалом. Это происходит в случае, когда значения параметров модели самолета в конце выполнения «мертвой петли» схожи со значениями параметров в начале ее выполнения. Это приводит к тому, что если автомат после осуществления «петли» находится в том же состоянии, в котором был в начале, то поведение модели зацикливается. В ходе экспериментов авторы неоднократно наблюдали выполнение двух «мертвых петель» подряд. Выполнение большего числа «петель» не наблюдалось.
3. Модель может вообще не выполнить «мертвую петлю», так как автомат не справляется с управлением, что, правда, бывает крайне редко.

Приведем ссылки на видеозаписи трех вариантов реализации «мертвой петли» под управлением лучшего автомата: выполнение «мертвой петли», близкое к «идеальному» (http://www.youtube.com/watch?v=TzrLoJjjVTA); выполнение «мертвой петли» с заваливанием на левый борт с последующим выравниванием (http://www.youtube.com/watch?v=C6WV7x2bqE8); последовательное выполнение двух «мертвых петель» (http://www.youtube.com/watch?v=yFiG4yz67Ks).

На рис. 37, а – л приведены 11 кадров первой из этих видеозаписей.

а)

б)



в)

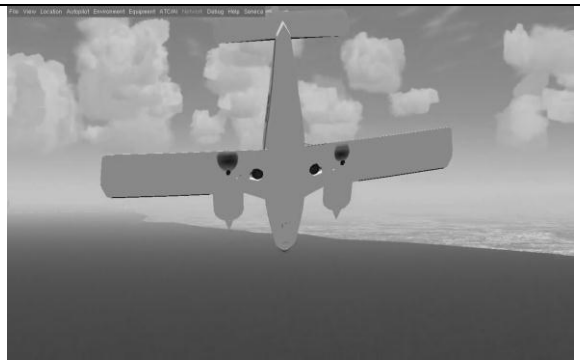
г)

д)

е)

ж)

з)



и)

к)

л)

Рис. 37. Кадры видеозаписи выполнения «мертвой петли» под управлением лучшего автомата

4.2. Внедрение результатов работы в учебный процесс

Для внедрения результатов работы в учебный процесс были разработаны две виртуальные лаборатории (на языках программирования *Java* и *C#*) для обучения построению конечных автоматов на основе эволюционных алгоритмов.

4.2.1. Виртуальная лаборатория на языке *Java*

Первая из разработанных виртуальных лабораторий реализована на языке программирования *Java* [168]. Она имеет модульную архитектуру – содержит ядро, предоставляющее базовую функциональность, которая может быть расширена за счет подключаемых модулей (плагинов).

Ядро виртуальной лаборатории позволяет просматривать и сохранять в виде файлов графики зависимости значений некоторых функций (например, максимального, минимального и среднего значения функции приспособленности особей поколения) от номера поколения. Кроме этого поддерживается возможность визуализации особей и их сохранения в текстовом формате. Также ядро программы поддерживает подключение плагинов «на лету» (во время работы программы).

В качестве подключаемых модулей выступают:

- решаемые задачи;
- функции, графики которых строятся ядром программы;
- реализации различных алгоритмов генетического программирования;
- реализации различных представлений особей и операций скрещивания и мутации для алгоритмов генетического программирования;
- визуализаторы особей (автоматов) и управляемых ими объектов (они зависят от конкретной задачи и от конкретного представления особи).

При этом отметим, что алгоритмы генетического программирования и особи в некотором смысле независимы – предполагается, что любой алгоритм рассматриваемого класса может работать с любым представлением особи.

Виртуальная лаборатория используется следующим образом. Решаемая задача формулируется в виде текстового описания задачи и выбранной функции приспособленности, для которой задается программная реализация. Задача оформляется в виде плагина к виртуальной лаборатории.

Для решаемой задачи выбирается метод представления конечных автоматов в виде особи эволюционного алгоритма. Программная реализация этого метода оформляется в виде соответствующего плагина.

Далее задаются параметры особи, например, число состояний автомата, и операторы скрещивания и мутации. Каждый оператор оформляется в виде отдельного плагина.

Далее выбирается эволюционный алгоритм, который будет применяться для построения конечных автоматов, и осуществляется настройка его параметров (размер поколения, вероятность мутации и т. д.). Алгоритм оформляется в виде плагина.

Далее осуществляется запуск алгоритма. В результате его работы пользователю представляются графики, например, график зависимости максимального значения функции приспособленности от номера поколения (рис. 38).

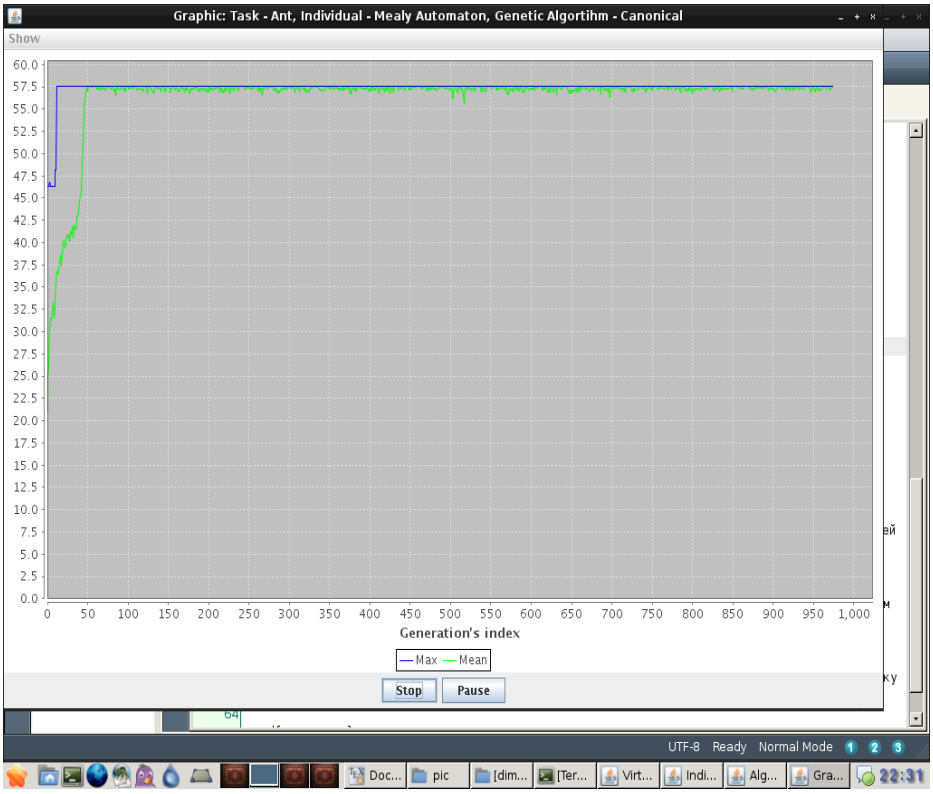


Рис. 38. Статистика работы генетического алгоритма

При необходимости работа алгоритма может быть приостановлена. В этом случае становится доступен выбор для просмотра любой особи из текущего поколения, а также лучших особей из каждого поколения (рис. 39).

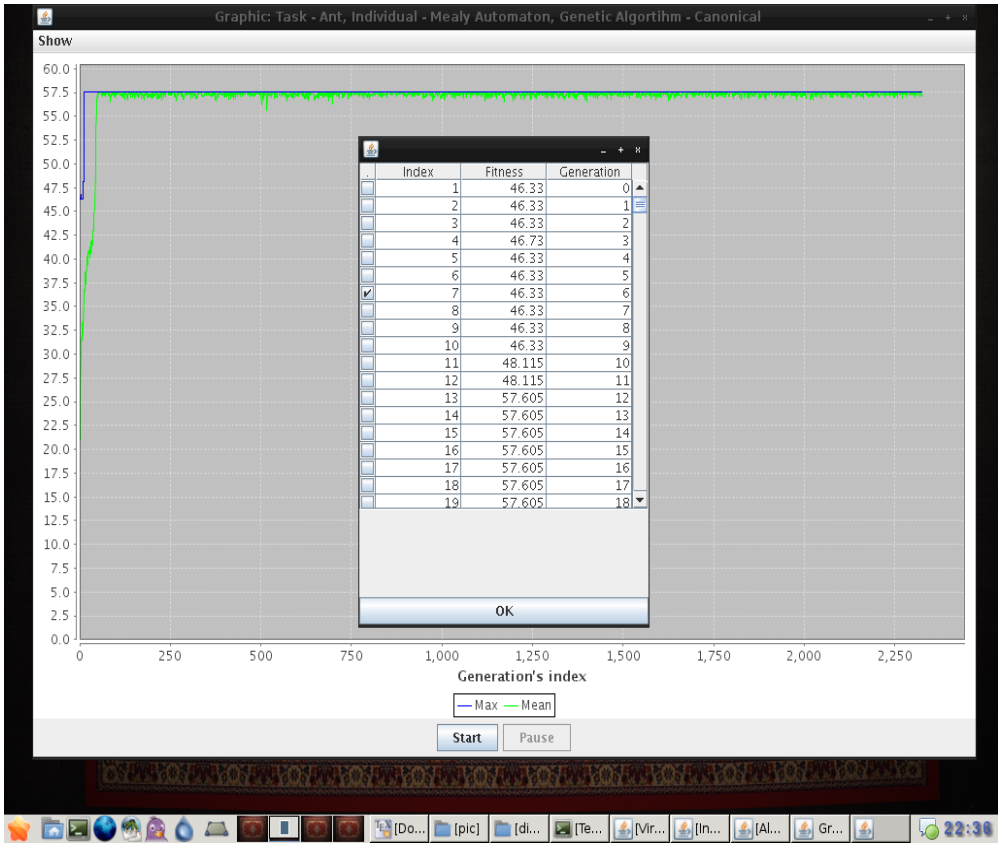


Рис. 39. Окно выбора особи для просмотра

Для просмотра особей используются их визуализаторы, которые также в виде плагинов к виртуальной лаборатории.

4.2.1.1. Структура виртуальной лаборатории

Ядро программы включает в себя три модуля.

- `common.jar` – содержит интерфейсы, необходимые для разработки плагинов;
- `core.jar` – ядро системы и пользовательский интерфейс;
- `util.jar` – содержит реализацию вспомогательных классов.

В состав виртуальной лаборатории входят следующие плагины:

- `algorithms/simple.jar` – генетический алгоритм;
- `functors/max.jar` – плагин для отображения графика зависимости максимального значения функции приспособленности от номера поколения;
- `functors/mean.jar` – плагин для отображения графика зависимости среднего значения функции приспособленности от номера поколения;
- `individuals/mealy` – особь эволюционного алгоритма, описывающая конечный автомат Мили.

4.2.1.2. Правила написания плагинов к виртуальной лаборатории

Как было сказано выше, виртуальная лаборатория имеет модульную архитектуру – содержит ядро, предоставляющее базовую функциональность, которая расширяется за счет подключаемых модулей (плагинов).

Отметим, что все упоминаемые ниже интерфейсы содержатся в файле `common.jar`, некоторые необязательные, но, возможно, полезные, при написании собственного модуля классы, содержатся в файле `util.jar`.

Создание плагина состоит в реализации соответствующего интерфейса и сборке `jar`-архива.

4.2.1.3. Плагин, содержащий представление особи

В рассматриваемой виртуальной лаборатории особь – это экземпляр класса, реализующего интерфейс `Individual`:

```
    public interface Individual extends Comparable<Individual>
{
        public double fitness();
        public Individual mutate(Random r);
        public Individual[] crossover(Individual p, Random r);
```

```
}
```

Метод `toString()` используется при сохранении особи в файл.

Для подключения модуля к ядру виртуальной лаборатории *JAR*-архив с реализацией особи должен быть помещен в директорию `individuals`. Этот *JAR*-архив должен содержать файл `MANIFEST.MF`, в котором должны быть определены следующие параметры: `Main-Class`, `Extension-Name`, `Comment`. Параметр `Main-Class` задает имя класса, реализующего интерфейс `Loader<IndividualFactory>`. Параметр `Extension-Name` содержит имя особи, отображаемое в диалоге выбора особи. Параметр `Comment` должен иметь вид «`arg1 arg2` ||| комментарий, отображаемый в диалоге выбора особи». Здесь `arg1` и `arg2` – параметры, необходимые для отображения графиков – максимальное значение и число знаков после десятичной точки, отображаемых в значении функции приспособленности.

4.2.1.4. Плагин, содержащий генетический алгоритм

Генетический алгоритм в виртуальной лаборатории – это экземпляр класса, реализующего интерфейс `GA`:

```
public interface GA {
    public List<Individual> getGeneration();
    public void nextGeneration();
    public void bigMutation();
    public Individual getBest();
}
```

Приведем описание методов, входящих в этот интерфейс:

- `getGeneration` – возвращает текущее поколение;
- `nextGeneration` – переход к следующему поколению;
- `bigMutation` – вызов большой мутации поколения;
- `getBest` – возвращает лучшую особь на данный момент.

Для подключения модуля к ядру виртуальной лаборатории *JAR*-файл с реализацией генетического алгоритма должен быть помещен в директорию `gas`. Этот *JAR*-архив должен содержать файл `MANIFEST.MF`, в котором должны быть определены следующие параметры: `Main-Class`, `Extension-Name`, `Comment`.

Параметр `Main-Class` – это имя класса, реализующего интерфейс `Loader<GA>`. Параметры `Extension-Name` и `Comment` будут отображаться в диалоге выбора алгоритма.

4.2.1.5. Плагин, содержащий визуализатор особи

Визуализатор особи – это экземпляр класса `java.awt.Frame`. Для подключения модуля к ядру виртуальной лаборатории *JAR*-файл с реализацией визуализатора должен быть помещен в директорию *emulators*. Этот *JAR*-архив должен содержать файл `MANIFEST.MF`, в котором должны быть определены следующие параметры: `Main-Class`, `Extension-Name`, `Comment`. Параметр `Main-Class` – имя класса, реализующего интерфейс `Loader<Frame>`. В данном случае метод `load` будет принимать одни параметр – возвращаемое значение метода `getAttributes()` экземпляра `Visualizable`.

```
public interface Visualizable extends Individual {
    public Object[] getAttributes();
}
```

Параметр `Comment` должен иметь вид «`arg1 arg2 ...|||` комментарий, отображаемый в диалоге выбора визуализатора». Параметр `Extension-Name` также используются при отображении в диалоге выбора визуализатора.

4.2.2. Виртуальная лаборатория на языке *C#*

Виртуальная лаборатория *GlOpt* [169] реализована на языке программирования C# на платформе *Microsoft.NET*. На эту виртуальную

лабораторию было получено свидетельство о регистрации программы для ЭВМ (третье свидетельство в Приложении 1).

Она состоит из ядра и подключаемых модулей (плагинов), которые реализуют конкретные задачи и методы их решения. Ядро представлено классом `Brain`, который ответственен за загрузку основных сущностей программного комплекса: задач, методов поисковой оптимизации и визуализаторов, а также выполняет функции распределения ресурсов между работающими алгоритмами.

4.2.2.1. Структура виртуальной лаборатории

Для описания задач используются абстрактные классы `Problem` и `Individual`, от которых наследуются классы, описывающие конкретные задачи, например задачу «Умный муравей».

Класс `Problem` содержит метод `OptimizationDirection`, возвращающий информацию о направления оптимизации (минимизация или максимизация) и метод `EvaluateIndividual`, предназначенный для вычисления функции приспособленности у конкретной особи. Наследники этого класса должны реализовывать эти методы, а также методы, обеспечивающие доступ к базовой информации о задаче: названии и ее кратком описании.

Алгоритмы решения описываются классами `Algorithm` и `SearchOperator`, от которых, в свою очередь, наследуются классы, реализующие конкретные методы поисковой оптимизации, например, метод спуска на основе случайных мутаций или генетический алгоритм.

Класс `SearchOperator` необходим для организации требуемых в задаче операций над объектами класса `Individual`. Он обеспечивает универсальность в применении методов решения к различным задачам.

Общая структура изображена на рис. 40.

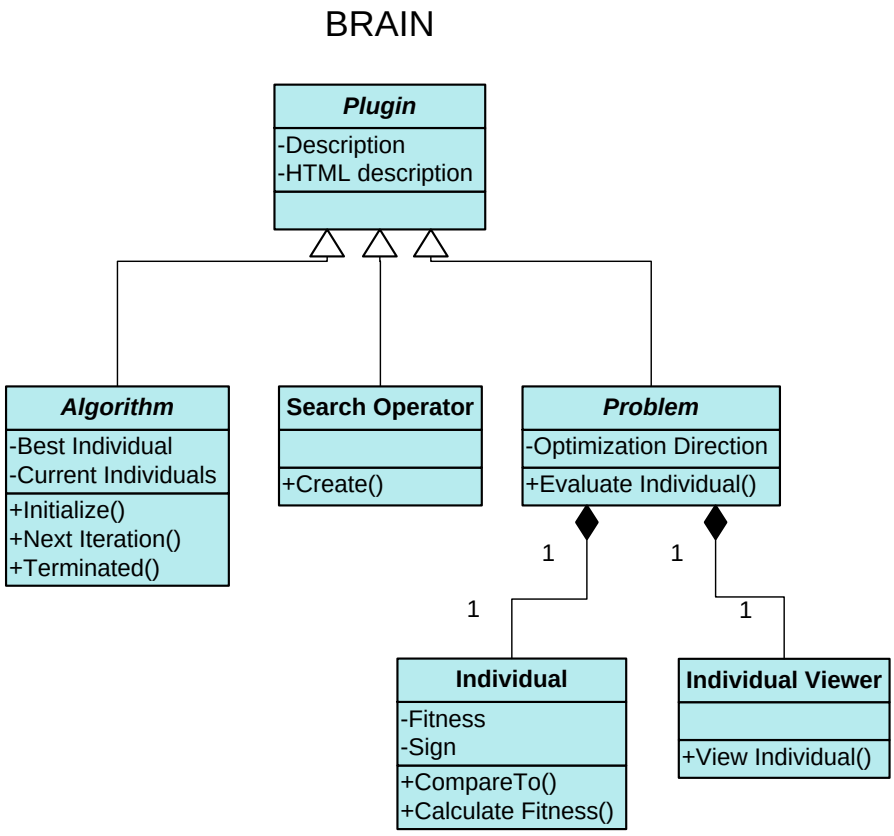


Рис. 40. Структура виртуальной лаборатории

Таким образом, в виртуальной лаборатории существует пять типов плагинов:

- рассматриваемые задачи (наследуется от абстрактного класса `Problem`);
- алгоритмы поисковой оптимизации (наследуется от абстрактного класса `Algorithm`);
- методы, адаптирующие алгоритмы к конкретным задачам (наследуется от абстрактного класса `SearchOperator`);
- визуализаторы для задач (пример внешнего вида визуализатора приведен на рис. 41);
- текстовые описания задач и алгоритмов.

Каждый плагин представляет собой `dll`-библиотеку. Особо отметим, что виртуальная лаборатория спроектирована таким образом, что

любую задачу можно решать, используя каждый из реализованных методов поисковой оптимизации.

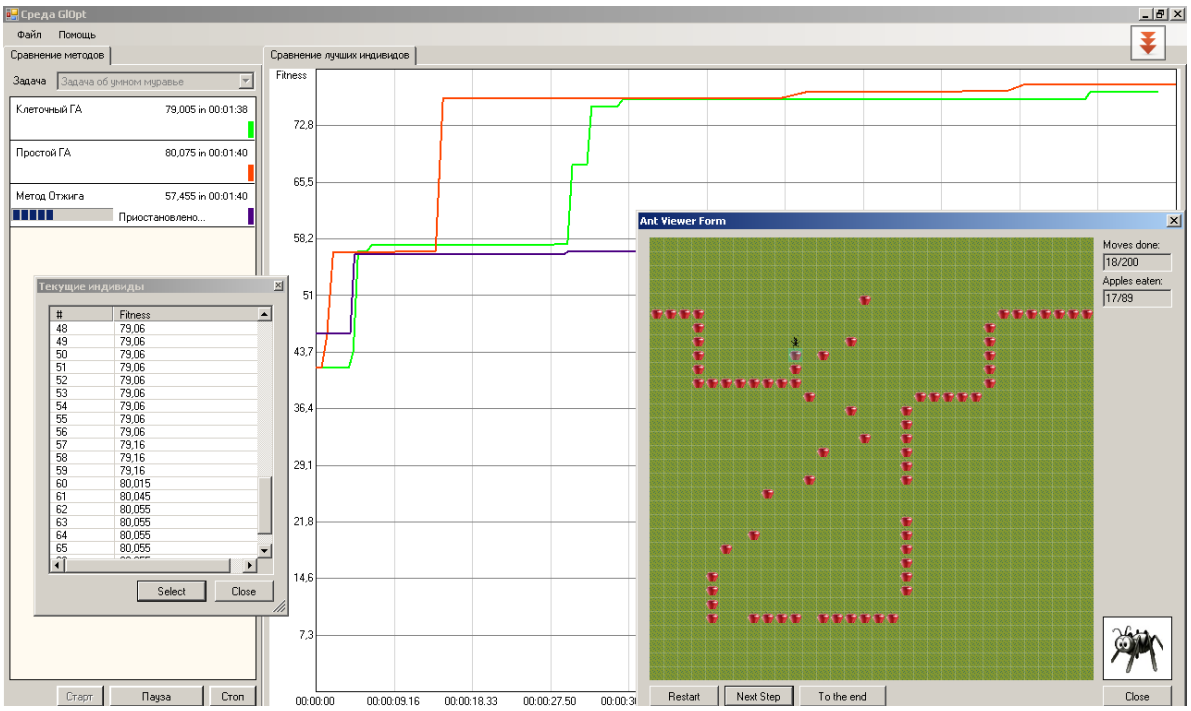


Рис. 41. Пример визуализатора

В состав виртуальной лаборатории входят следующие плагины:

- задачи: построение автомата по тестам, «Умный муравей», «Умный муравей-3»;
- методы поисковой оптимизации: генетический алгоритм, (1+1)-эволюционная стратегия;
- визуализатор для задачи «Умный муравей»;
- средства построения графиков зависимости значения функции приспособленности от номера поколения;
- документация, в том числе `html`-описания решаемых задач и используемых для их решения методов.

4.2.2.2. Правила написания плагинов к виртуальной лаборатории

Ниже приведены базовые рекомендации, которым необходимо следовать при разработке собственных модулей, определяющих алгоритм решения. Процесс создания плагинов описывается на примере создания плагина задачи и алгоритма решения.

Основные этапы при реализации новой задачи в виртуальной лаборатории *GLOpt*.

1. Реализация класса-наследника от абстрактного класса `Individual`, определяющего объект к которому в дальнейшем применяется алгоритм оптимизации. Так, например, в задаче о расстановке ферзей таким объектом выступает шахматная доска с расставленными на ней ферзями.
2. Реализация класса-наследника от абстрактного класса `Problem`. В данном классе должен быть определен метод `OptimizationDirection`. Также необходимо определить метод `EvaluateIndividual`, возвращающий значение типа `double` – значение целевой функции для данного индивида.
3. Для реализации компоненты визуализации текущего решения требуется определить класс `Viewer`, наследуемый от класса `IndividualViewer`, и, в частности, определить метод `ViewIndividual`, вызывающий графическую форму, либо представляющий данные о решении в любом другом удобном для пользователя виде.

Приведем основные этапы разработки подключаемого модуля, определяющего алгоритм.

1. Реализация алгоритма поиска оптимального решения – класса, наследованного от абстрактного класса `Algorithm`. Данный класс должен определять следующие методы.

- `OptimizationDirection` – возвращает одну из двух именованных констант: `OptimizationDirection.Minimize` или `OptimizationDirection.Maximize`.
- `Initialize` – описывает начальное состояние в алгоритме поиска решения.
- `NextIteration` – описывает действия, происходящие на очередной итерации алгоритма.
- В переменной `BestIndividual` типа `Individual` должна содержаться актуальная информация об объекте типа `Individual` с лучшей на данной итерации алгоритма целевой функцией.

2. Реализация интерфейса `SearchOperator`, наследованного от интерфейса `ICreateOperator`. В данном интерфейсе должен быть определен метод `Create`, возвращающий объект класса `Individual` (абстрактный класс, для каждой задачи используется своя реализация). Также в `SearchOperator` реализуются все необходимые методы для работы с объектами `Individual` – например, операции мутации и скрещивания.

4.2.3. Применение виртуальных лабораторий в учебном процессе

С 2008 г. в рамках дисциплины «Теория автоматов и программирование» проводятся лекционные занятия и лабораторные работы по методам применения эволюционных алгоритмов для построения управляющих конечных автоматов.

Для проведения лабораторных работ используются описанные выше виртуальные лаборатории. При этом виртуальная лаборатория на языке программирования *Java* используется с 2008 г., а на языке программирования C# – с 2009 г.

Целью лабораторных работ является изучение методов построения конечных автоматов на основе эволюционных алгоритмов. В задании на лабораторную работу формулируется задача, для которой необходимо построить конечный автомат, а также требования к используемому эволюционному алгоритму, методу представления особи, выполнения операций скрещивания и мутации, в том числе, и предложенные в настоящей работе.

Каждая лабораторная работа должна выполняться одним студентом. В рамках работы студент реализует подключаемые модули к виртуальной лаборатории на соответствующем языке программирования (*Java* или *C#*), а также выполняет несколько вычислительных экспериментов по построению управляющих конечных автоматов. По результатам выполнения лабораторной работы студент оформляет отчет. Пример отчета приведен в Приложении 2.

За время использования виртуальных лабораторий в учебном процессе лабораторные работы были выполнены более чем 150 студентами кафедры «Компьютерные технологии» НИУ ИТМО. Часть отчетов по лабораторным работам опубликована на сайте http://is.ifmo.ru/labs/.

Выводы по главе 4

1. Разработанные методы построения управляющих конечных автоматов использованы при построении автомата управления моделью беспилотного самолета.
2. Разработанные методы построения управляющих конечных автоматов внедрены в учебный процесс в рамках дисциплин «Теория автоматов и программирование» на кафедре «Компьютерные технологии» НИУ ИТМО.

ЗАКЛЮЧЕНИЕ

В результате диссертационного исследования получены следующие результаты:

1. Разработан метод построения управляющих конечных автоматов по обучающим примерам на основе эволюционных алгоритмов. Его основное отличие от известных состоит в том, что в эволюционные алгоритмы добавлен новый шаг «Расстановка выходных воздействий», который выполняется перед вычислением функции приспособленности.
2. Разработан метод выполнения операции скрещивания для генетических алгоритмов, учитывающий поведение автоматов на обучающих примерах. Показано, что генетический алгоритм, использующий разработанный метод выполнения операции скрещивания, осуществляет построение автоматов по обучающим примерам быстрее, чем генетический алгоритм, использующий традиционный метод выполнения операции скрещивания, эволюционная стратегия и метод спуска на основе случайных мутаций.
3. Разработан метод построения автоматов по обучающим примерам и темпоральным формулам на основе эволюционных алгоритмов и верификации. Его основное отличие от известных методов состоит в том, что для вычисления функции приспособленности совместно применяются обучающие примеры и верификация.
4. Разработана технология построения автоматов по обучающим примерам и темпоральным формулам.
5. Разработано инструментальное средство для автоматизации построения автоматов.
6. Разработанные методы использованы при построении автомата управления моделью беспилотного самолета и внедрены в

учебный процесс. Для внедрения в учебный процесс были разработаны две виртуальные лаборатории. С их использованием лабораторные работы были выполнены более чем 150 студентами кафедры «Компьютерные технологии» НИУ ИТМО.

СПИСОК ИСТОЧНИКОВ

ПЕЧАТНЫЕ ИЗДАНИЯ НА РУССКОМ ЯЗЫКЕ

1. *Букатова И. Л.* Эволюционное моделирование и его приложения. М.: Наука, 1979.
2. *Гладков Л. А., Курейчик В. В., Курейчик В. М.* Генетические алгоритмы. М.: Физматлит, 2006.
3. *Гуров В. С., Мазин М. А., Нарвский А. С., Шалыто А. А.* UML. SWITCH-технология. Eclipse //Информационно-управляющие системы. 2004. № 6, с. 12 – 17. http://is.ifmo.ru/works/uml-switch-eclipse/
4. *Вельдер С. Э., Лукин М. А., Шалыто А. А., Яминов Б. Р.* Верификация автоматных программ. СПб.: Наука, 2011. http://is.ifmo.ru/verification/velder_verification_posobie_nauka.pdf
5. *Данилов В. Р.* Метод представления автоматов деревьями решений для использования в генетическом программировании //Научно-технический вестник СПбГУ ИТМО. 2008. Выпуск 53. Автоматное программирование, с. 103 – 108. http://is.ifmo.ru/works/2008/Vestnik/53/08-genetic-automata-decision-tree-method.pdf
6. *Егоров К. В., Шалыто А. А.* Методика верификации автоматных программ // Информационно-управляющие системы. 2008. № 5, с. 15 – 21. http://is.ifmo.ru/works/_egorov.pdf
7. *Емельянов В. В., Курейчик В. М., Курейчик В. В.* Теория и практика эволюционного моделирования. М.: Физматлит, 2003.
8. *Заикин А. К.* Разработка методов построения конечных автоматов с использованием алгоритма имитации отжига на примере игры «Война за ресурсы» // Научно-технический вестник СПбГУ ИТМО. 2011. № 2, с. 49 – 54. http://is.ifmo.ru/works/_egorov.pdf
9. *Кларк Э., Грамберг О., Пелед Д.* Верификация моделей программ. М.: МЦНМО, 2002.

10. *Кормен* Т., *Лейзерсон* Ч., *Ривест* Р., *Штайн К.* Алгоритмы. Построение и анализ. М.: Вильямс, 2005.
11. *Курейчик В.М.* Генетические алгоритмы. Состояние, проблемы, перспективы // Известия РАН. Теория и системы управления. 1999. № 1, с. 144 – 160.
12. *Курейчик В.В., Курейчик В.М., Сороколетов П.В.* Анализ и обзор моделей эволюции // Известия РАН. Теория и системы управления. 2007. № 5, с. 114 – 126.
13. *Левенштейн В. И.* Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академии наук СССР. 1965. № 4, с. 845 – 848.
14. *Лобанов П. Г., Шалыто А. А.* Использование генетических алгоритмов для автоматического построения конечных автоматов в задаче о «Флибах» / Сборник докладов 4-й Всероссийской научной конференции «Управление и информационные технологии» (УИТ-2006). СПбГЭТУ «ЛЭТИ». 2006, с.144 – 149. http://is.ifmo.ru/works/flib
15. *Лобанов П. Г.* Использование генетических алгоритмов для генерации конечных автоматов. Диссертация на соискание ученой степени кандидата технических наук. СПбГУ ИТМО. 2008. http://is.ifmo.ru/disser/lobanov_disser.pdf
16. *Люгер Дж.* Искусственный интеллект: стратегии и методы решения сложных проблем. М: Вильямс, 2003.
17. *Мандриков Е.А., Кулев В.А.* Разработка инструментального средства для генерации конечных автоматов с использованием генетических алгоритмов //Научно-технический вестник СПбГУ ИТМО. 2008. Вып. 53. Автоматное программирование, с. 100 – 103. http://is.ifmo.ru/works/2008/Vestnik/53/07-genetic-automata-tool.pdf
18. *Непейвода Н. Н.* Стили и методы программирования. М.: Интернет-Университет Информационных технологий, 2005.

19. *Николенко С. И., Тулупьев А. Л.* Самообучающиеся системы. М.: МЦНМО, 2009.
20. *Норенков И. П., Арутюнян Н. М.* Метагенетический алгоритм оптимизации и структурного синтеза проектных решений // Информационные технологии. 2007. № 3, с. 10 – 13.
21. *Параще́нко 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/
22. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб.: Питер, 2009. http://is.ifmo.ru/books/_book.pdf
23. *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, 100 – 117. http://is.ifmo.ru/works/_polikarpova_samolet.pdf
24. *Потапов А. С.* Искусственный интеллект и универсальное мышление. СПб.: Политехника, 2012.
25. *Рассел С., Норвиг П.* Искусственный интеллект. Современный подход. М.: Вильямс. 2006.
26. *Хопкрофт Дж., Мотвани Р., Ульман Дж.* Введение в теорию автоматов, языков и вычислений. М.: Вильямс, 2002.
27. *Шалыто А. А.* Автоматное программирование / Материалы конференции с международным участием «Технические и программные средства систем управления, контроля и измерения» (УКИ’10). ИПУ РАН. 2010, с. 156 – 167. http://cmm.ipu.ru/proc/%D0%A8%D0%B0%D0%BB%D1%8B%D1%82%D0%BE%20%D0%90.%D0%90.%20.pdf

28. *Шалыто А. А.* Switch-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. http://is.ifmo.ru/books/switch/1
29. *Шалыто А. А.* Логическое управление. Методы аппаратной и программной реализации. СПб.: Наука, 2000. http://is.ifmo.ru/books/log_upr/1

Печатные издания на английском языке

30. *Afzal W., Torkar R., Feldt R.* A Systematic Review of Search-Based Testing for Non-Functional System Properties // Information and Software Technology. 2009. Vol. 51. № 6, pp. 957 – 976.
31. *Alba E., Chicano F.* ACOhg: Dealing with Huge Graphs / Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO ’07). NY. : ACM. 2007, pp. 10 – 17.
32. *Alba E., Chicano F.* Ant Colony Optimization for Model Checking / Proceedings of the 11th International Conference on Computer Aided Systems Theory (EUROCAST 2007), pp. 523 – 530.
33. *Alba E., Chicano F.* Finding Safety Errors with ACO / Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation (GECCO ’07). NY. : ACM. 2007, pp. 1066 – 1073.
34. *Ali S., Briand L., Hemmati H., Panesar-Walawege R. K.* A Systematic Review of the Application and Empirical Investigation of Search-Based Test-Case Generation // IEEE Transactions of Software Engineering. 2010. Vol. 36, № 6, pp. 742 – 762.
35. *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
36. *Back T.* Evolutionary Algorithms in Theory and Practice. Oxford University Press, 1996.

37. *Bagnall A.J., Rayward-Smith V.J., Whittley I.M.* The Next Release Problem // Information and Software Technology. Dec. 2001, pp. 883 – 890.
38. *Belz A., Eskikaya B.* A Genetic Algorithm for Finite State Automata Induction with Application to Phonotactics / Proceedings of the ESSLLI-98 Workshop on Automated Acquisition of Syntax and Parsing. Saarbruecken. 1998, pp. 9 – 17. http://www.itri.brighton.ac.uk/~Anja.Belz/Publications/A_GA_for_FSA_induction_with_an_application_to_phonotactics.ps
39. *Benson K.* Evolving Finite State Machines with Embedded Genetic Programming for Automatic Target Detection / Proceedings of the Congress on Evolutionary Computation. 2000. Vol. 2, pp. 1543 – 1549. http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=870838
40. *Beyer H. G.* The Theory of Evolution Strategies. Natural Computing Series. Berlin. NY: Springer-Verlag, 2001.
41. *Brave S.* Evolving Deterministic Finite Automata Using Cellular Encoding / Proceeding of First Annual Conference on Genetic Programming. 1996, pp. 39 − 44. http://citeseer.ist.psu.edu/131538.html
42. *Bryant R. E.* Graph-based algorithms for boolean function manipulation / IEEE Transactions on Computers. 1986, Vol. 35, pp. 677 – 691.
43. *Chambers L.* Practical Handbook of Genetic Algorithms. Complex Coding Systems. Volumes I. II, III. CRC Press, 1999.
44. *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.
45. *Clark J., Dolado J. J., Harman M., Hierons R., Jones B., Lumkin M., Mitchell B., Mancoridis S., Rees K., Roper M., Shepperd M.* Reformulating Software Engineering as a Search Problem. IEEE Proceedings-Software. 2003. Vol. 150. № 3, pp. 161 – 175.

46. *Das R., Mitchell M., Crutchfield J. P.* A genetic algorithm discovers particle-based computation in cellular automata // Lecture Notes in Computer Science. 1994. www.santafe.edu/research/publications/workingpapers/94-03-015.pdf
47. *De Castro L., Timmis J.* Artificial Immune Systems: A New Computational Intelligence Approach. Springer, 2002.
48. *De Jong K.* Evolutionary computation: a unified approach. MIT Press, Cambridge. MA, 2006.
49. *Dorigo M., Stutzle T.* Ant Colony Optimization. The MIT Press, 2004.
50. *Eisenstein J.* Evolving robot tank fighters. Technical Report AIM-2003-023. AI Lab. MIT, 2003.
51. *Ferrante N. Cotta C., Moscato P.* Handbook of Memetic Algorithms. Berlin, NY: Springer-Verlag, 2012.
52. *Fogel L.* Autonomous Automata // Industrial Research. 1962. V.4, pp. 14 – 19.
53. *Fogel L., Owens A., Walsh M.* Artificial Intelligence through Simulated Evolution. NY: Wiley, 1966.
54. *Frey C., Leugering G.* Evolving Strategies for Global Optimization – A Finite State Machine Approach /Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001). Morgan Kaufmann. 2001, pp. 27 – 33. http://citeseer.ist.psu.edu/456373.html
55. *Gastin P., Oddoux D.* Fast LTL to Büchi Automata Translation / 13th Conference on Computer Aided Verification (CAV'01). 2001, pp. 53 – 65.
56. *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.

57. *Ghnemat R., Khatatneh K., Oqeili S., Bertelle C., Duchamp G.* Automata-based adaptive behavior for economic modeling using game theory. 2005. http://arxiv.org/abs/cs/0510089
58. *Glover F.* Tabu search: A tutorial //Interfaces. 1990. Vol. 20, pp. 74 – 94.
59. *Glover F., Laguna M., Martí R.* Fundamentals of Scatter Search and Path Relinking // Control and Cybernetics, 2000. № 29 (3), pp. 653 – 684
60. *Godefroid P.* Model Checking for Programming Languages using Verisoft / Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages, pp. 174 – 186.
61. *Gold E. M.* Complexity of automaton identification from given data. // Information and Control. 1978. № 37, pp. 302 − 320.
62. *Goldberg D.* A Note on Boltzmann Tournament Selection for Genetic Algorithms and Population-Oriented Simulated Annealing // Complex Systems. 1990. Vol. 4, pp. 445 − 460.
63. *Goldsby H. J., Cheng B. H.* Avida-MDE: a digital evolution approach to generating models of adaptive software behavior / Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation (GECCO'08), pp. 1751 – 1758.
64. *Hamming R.* Error detecting and error correcting codes // Bell System Technical Journal 29 (2), pp. 147 – 160.
65. *Harel D., Pnueli A.* On the development of reactive systems / In «Logic and Models of Concurrent Systems». NATO Advanced Study Institute on Logic and Models for Verification and Specification of Concurrent Systems. Springer Verlag, 1985. pp. 477 – 498.
66. *Harman M., Hierons R., Proctor M.* A new representation and crossover operator for search-based optimization of software modularization / In GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference (NY, 2002). Morgan Kauffman Publishers, pp. 1351 – 1358.

67. *Harman M.* Automated Test Data Generation Using Search-Based Software Engineering / Proceedings of 2nd International Workshop Automation of Software Test (AST 07). IEEE CS Press, 2007, p. 2.
68. *Harman M., Tratt L.* Pareto Optimal Search-Based Refactoring at the Design Level / Proceedings of 9th Annual Conference on Genetic and Evolutionary Computation (GECCO 07). ACM Press. 2007, pp. 1106 – 1113.
69. *Harman M., Mansouri A., Zhang Y.* Search-Based Software Engineering: A Comprehensive Analysis and Review of Trends, Techniques, and Applications. Tech. report TR-09-03. Dept. of Computer Science. King's College London, 2009.
70. *Harman M.* Why the Virtual Nature of Software Makes It Ideal for Search-Based Optimization / Proceeding of 13th International Conference Fundamental Approaches to Software Engineering (FASE 10). IEEE CS Press. 2010, pp. 1 – 12.
71. *Harman M.* Software Engineering Meets Evolutionary Computation // Computer. 2011. Vol. 44. № 11, pp. 31 – 39.
72. *Harman M., Clark J.* Metrics are fitness functions too / Proceedings of 10th International Symposium on Software Metrics. 2004, pp. 58 – 69.
73. *Hoffman L.* Talking Model-Checking Technology // Communications of the ACM. 2008. V. 51. № 7, pp. 110 – 112.
74. *Holland J.* Adaptation in Natural and Artificial Systems. University of Michigan Press, 1975.
75. *Holland J.* ECHO: Explorations of Evolution in a Miniature World / Proceedings of the Second Conference on Artificial Life. Redwood City. CA: Addison-Wesley, 1990.
76. *Holzmann G. J.* The Model Checker *SPIN* // IEEE Transactions on software engineering. 1997. Vol. 23. Issue 5, pp. 279 – 295.

77. *Horihan J., Yung-Hsiang Lu.* Improving FSM evolution with progressive fitness functions / GLSVLSI '04: Proceedings of the 14th ACM Great Lakes symposium on VLSI. NY: ACM Press. 2004, pp. 123 − 126.
78. *Huang D.* MS Thesis Preproposal: Adaptive Incremental Fitness Evaluation in Genetic Algorithms. 2005. NY: Rochester. http://www.cs.rit.edu/~dxh6185/downloads/MS_Thesis/Documents/Presentation.pdf
79. *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
80. *Johnson C.* Genetic Programming with Fitness based on Model Checking. Lecture Notes in Computer Science. Springer Berlin / Heidelberg. 2007. Volume 4445/2007, pp. 114 – 124.
81. *Juillie H., Pollack J.* Coevolving the «Ideal» Trainer: Application to the Discovery of Cellular Automata Rules. 1998. http://citeseer.ist.psu.edu/16712.html
82. *Kennedy J., Eberhart R. C.* Swarm Intelligence. Morgan Kaufmann. 2001.
83. *Kirsopp C., Shepperd M., Hart J.* Search heuristic, case-based reasoning and software project effort prediction / GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference. NY: 2002, pp. 1367 – 1374.
84. *Koza J.* Genetic programming. On the Programming of Computers by Means of Natural Selection. MA: The MIT Press, 1998.
85. *Koza J.* Genetic Evolution and Co-Evolution of Computer Programs / Proceedings of Second Conference on Artificial Life. Redwood City, CA: Addison-Wesley. 1992. pp. 603 − 629. http://citeseer.ist.psu.edu/177879.html

86. *Larranaga P., Lozano J. A.* Estimation of distribution algorithms: A new tool for evolutionary computation. Kluwer Academic Publishers. Boston, 2002.
87. *Levy S.* Artificial Life: The Quest for a New Creation. NY: Pantheon, 1992.
88. *Lucas S., Reynolds J.* Learning DFA: Evolution versus Evidence Driven State Merging / The 2003 Congress on Evolutionary Computation (CEC '03). Vol. 1, pp. 351 – 358.
89. *Lucas S., Reynolds J.* Learning Deterministic Finite Automata with a Smart State Labeling Algorithm // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2005. Vol. 27. № 7, pp. 1063 – 1074.
90. *Lucas S.* Evolving Finite-State Transducers: Some Initial Explorations. Lecture Notes in Computer Science. Springer Berlin / Heidelberg. Volume 2610/2003, pp. 241 – 257. http://www.springerlink.com/content/41a34vg70fp1hltb/
91. *Mancoridis S., Mitchell B. S., Rorres C., Chen Y., Gansner E. R.* Using Automatic Clustering to Produce High-Level System Organizations of Source Code / Proceedings of the 6th International Workshop on Program Comprehension (IWPC ’98), pp. 45 – 52.
92. *McMillan K. L.* Symbolic model checking: an approach to the state explosion problem. PhD thesis. SCS. Carnegie Mellon University, 1992.
93. *McMinn P.* Search-Based Software Test Data Generation: A Survey // Software Testing, Verification and Reliability. 2004. Vol. 14. № 2, pp. 105 – 156.
94. *Metropolis N., Rosenbluth A., Rosenbluth M., Teller A., Teller E.* Equation of state calculations by fast computing machines. Journal of Chemical Physics. 1953. Vol. 21, pp. 1087 – 1092.
95. *Miller J.* The Coevolution of Automata in the Repeated Prisoner's Dilemma. Working Paper. Santa Fe Institute. 1989 // Journal of Economic Behavior & Organization. 1996. Vol. 29. Issue 1, pp. 87 – 112.

96. *Mitchell B. S.* A Heuristic Search Approach to Solving the Software Clustering Problem. PhD Thesis. Drexel University, Philadelphia.
97. *Mitchell B. S., Mancordis S.* Using heuristic search techniques to extract design abstractions from source code / GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference (NY, 2002), pp. 1375 – 1382.
98. *Mitchell M.* An Introduction to Genetic Algorithms. MA: The MIT Press, 1996.
99. *Mitchell M., Crutchfield J. P., Hraber P. T.* Evolving cellular automata to perform computations. Physica D. 1993. Vol. 75, pp. 361 – 391. http://web.cecs.pdx.edu/~mm/mech-imped.pdf
100. Mitchell M., *Holland J., Forrest S.* When Will a Genetic Outperform Hill Climbing? / Advances in Neural Information Processing Systems 6. San Mateo. CA: Morgan Kaufmann, 1994.
101. *Moonjoo K., Viswanathan M., Ben-Abdallah H., Kannan S., Lee I., Sokolsky O.* Formally specified monitoring of temporal properties / Proceedings of the 11th Euromicro Conference on Real-Time Systems, pp. 114 – 122.
102. *Naidoo A., Pillay N.* The Induction of Finite Transducers Using Genetic Programming / Proceedings of the 10th European conference on Genetic programming, pp. 371 – 380. http://saturn.cs.unp.ac.za/~nelishiap/papers/eurogp07.pdf
103. *Nedjah N., Mourelle L.* Mealy Finite State Machines: An Evolutionary Approach // International Journal of Innovative Computing, Information and Control. 2006. Vol. 2. Issue 4, pp. 789 – 806.
104. *Niebert P., Peled D., Pnueli A.* Discriminative Model Checking. Lecture Notes In Computer Science. Springer Berlin / Heidelberg. 2008. Vol. 5123/2008, pp. 504 – 516.
105. *Nocedal J., Wright S.* Numerical Optimization. 2006. Berlin, NY: Springer-Verlag.

106. *O'Keeffe M., O'Cinneide M.* Search-Based Software Maintenance / Proceedings of Conference on Software Maintenance and Reengineering (CSMR 06). IEEE CS Press. 2006, pp. 249 – 260.
107. *Petrovic P.* Simulated evolution of distributed FSA behaviour-based arbitration. Poster at The Eighth Scandinavian Conference on Artificial Intelligence (SCAI'03). 2003. http://www.idi.ntnu.no/grupper/ai/eval/incremental/scai03-poster-petrovic.pdf
108. *Petrovic P.* Evolving automatons for distributed behavior arbitration. Technical Report. Norwegian University of Science and Technology, 2005. http://www.idi.ntnu.no/~petrovic/shortpaper.pdf
109. *Petrovic P.* Comparing Finite-State Automata Representation with GP-trees. Technical Report. Norwegian University of Science and Technology, 2006. http://www.idi.ntnu.no/~petrovic/fsatr/fsatr.pdf
110. *Räihä O.* A Survey on Search-Based Software Design //Computer Science Rev. 2010. Vol. 4. № 4, pp. 203 – 249.
111. *Ray T.* An Approach to the Synthesis of Life / In Artificial Life II. MA: Addison-Wesley. 1992, pp. 371 − 408.
112. *Reynolds C. W.* Competition, Coevolution and the Game of Tag / Proceedings of Artificial Life IV. Cambridge. MA: MIT Press. 1994, pp. 59 − 69. http://www.red3d.com/cwr/papers/1994/alife4.html
113. *Schrijver A.* Theory of linear and integer programming. John Wiley and Sons, 1998.
114. *Spears W., Gordon D.* Evolving Finite-State Machine Strategies for Protecting Resources. Lecture Notes in Computer Science. Springer Berlin / Heidelberg. Volume 1932/2009, pp. 5 – 28.
115. *Spichakova M.* Genetic Inference of Finite State Machines. Masters thesis. Tallinn, 2007. http://s-ma-u-g.googlecode.com/files/thesis.pdf

116. *Tomita M.* Dynamic construction of finite automata from examples using hill climbing / Proceedings of the 4th Annual Cognitive Science Conference (USA, 1982), pp. 105 – 108.
117. *Tu M., Wolff E., Lamersdorf W.* Genetic Algorithms for Automated Negotiations: A FSM-based Application Approach /Proceedings of the 11-th International Conference on Database and Expert Systems, 2000. http://ieeexplore.ieee.org/iel5/7035/18943/00875153.pdf
118. *Von Neumann J., Burks A.* The Theory of Self-reproducing Automata. Urbana. Univ. of Illinois Press, 1966.
119. *Wegener J., Baresel A., Sthamer H.* Evolutionary test environment for automatic structural testing // Information and Software Technology Special Issue on Software Engineering using Metaheuristic Innovative Algorithms. 2001. Vol. 43. Issue 14, pp. 841 – 854.
120. *Whitley D.* The GENITOR Algorithm and Selective Pressure / Proceedings of the 3rd International Conference on Genetic Algorithms. Colorado State: Morgan Kaufmann. 1989, pp. 116 − 121. http://www.genalgo.com/index.php?option=com_content&task=view&id=80&Itemid=30
121. *Zhang Y., Finkelstein A., Harman M.* Search-Based Requirements Optimisation: Existing Work and Challenges / Proceedings of Internationall Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 08). LNCS 5025. Springer. 2008, pp. 88 – 94.
122. *Zomorodian A.* Context-free Language Induction by Evolution of Deterministic Push-Down Automata Using Genetic Programming. http://www.cs.dartmouth.edu/~afra/papers/aaai96/aaai96.pdf

Ресурсы сети Интернет

123. *Бедный Ю. Д., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов в задаче «Умный муравей». СПбГУ ИТМО, 2007. http://is.ifmo.ru/works/ant
124. Государственный контракт «Применение методов искусственного интеллекта в разработке управляющих программных систем». Промежуточный отчет по II этапу. СПбГУ ИТМО, 2010. http://is.ifmo.ru/science/_nk-408-15-2.pdf
125. Государственный контракт «Разработка технологии верификации управляющих программ со сложным поведением, построенных на основе автоматного подхода». Промежуточный отчет по I этапу «Выбор направления исследований и базовых компонентов». СПбГУ ИТМО, 2007. http://is.ifmo.ru/verification/_2007_01_report-verification.pdf
126. Государственный контракт «Разработка технологии верификации управляющих программ со сложным поведением, построенных на основе автоматного подхода». Промежуточный отчет по II этапу «Теоретические исследования поставленных перед НИР задач». СПбГУ ИТМО, 2007. http://is.ifmo.ru/verification/_2007_02_report-verification.pdf
127. Государственный контракт «Разработка технологии верификации управляющих программ со сложным поведением, построенных на основе автоматного подхода». Промежуточный отчет по III этапу «Экспериментальные исследования поставленных перед НИР задач». СПбГУ ИТМО, 2007. http://is.ifmo.ru/verification/_2007_02_report-verification.pdf
128. Государственный контракт «Разработка технологии верификации управляющих программ со сложным поведением, построенных на основе автоматного подхода». Заключительный отчет по IV этапу

«Обобщение и оценка результатов исследований». СПбГУ ИТМО, 2007. http://is.ifmo.ru/verification/_2007_02_report-verification.pdf

129. Государственный контракт «Технология генетического программирования для генерации автоматов управления системами со сложным поведением». Промежуточный отчет по этапу I «Выбор направлений исследований и базовых компонентов». СПбГУ ИТМО, 2007. http://is.ifmo.ru/genalg/_2007_01_report-genetic.pdf
130. Государственный контракт «Технология генетического программирования для генерации автоматов управления системами со сложным поведением». Промежуточный отчет по этапу II «Теоретические исследования поставленных перед НИР задач». СПбГУ ИТМО, 2007. http://is.ifmo.ru/genalg/_2007_02_report-genetic.pdf
131. Государственный контракт «Технология генетического программирования для генерации автоматов управления системами со сложным поведением». Промежуточный отчет по этапу III «Экспериментальные исследования поставленных перед НИР задач». СПбГУ ИТМО, 2007. http://is.ifmo.ru/genalg/_2007_03_report-genetic.pdf
132. Государственный контракт «Технология генетического программирования для генерации автоматов управления системами со сложным поведением». Промежуточный отчет по этапу IV «Обобщение и оценка результатов исследований». СПбГУ ИТМО, 2007. http://is.ifmo.ru/genalg/_2007_04_report-genetic.pdf
133. Государственный контракт «Разработка методов машинного обучения на основе генетических алгоритмов для построения управляющих конечных автоматов». Промежуточный отчет по этапу I. СПбГУ ИТМО, 2009. http://is.ifmo.ru/science/_nk-385-1-1.pdf

134. *Данилов В. Р.* Технология генетического программирования для генерации автоматов управления системами со сложным поведением. Бакалаврская работа. СПбГУ ИТМО, 2007. http://is.ifmo.ru/download/danilov_bachelor.pdf
135. *Данилов В. Р.* Методы представления функции переходов при генерации автоматов управления на основе генетического программирования. Магистерская диссертация. СПбГУ ИТМО, 2009. http://is.ifmo.ru/papers/_2010_02_25_danilov.pdf
136. Заочный тур всесибирской олимпиады 2005 по информатике. http://olimpic.nsu.ru/widesiberia/archive/wso6/2005/rus/1tour/problem/problem.html
137. *Колыхматов И. И., Рыбак О. О., Шалыто А. А.* Моделирование устройства для продажи газированной воды на инструментальном средстве *UniMod*. Проектная документация. СПбГУ ИТМО. 2006. http://is.ifmo.ru/download/vending-machine-ru.pdf
138. *Николенко С. И.* Лекции по генетическим алгоритмам. http://logic.pdmi.ras.ru/~sergey/teaching/ml/04-genetic.pdf
139. Сайт www.genetic-programming.com
140. Сайт http://www.flightgear.org/
141. *Яминов Б. Р.* Генетические алгоритмы. http://rain.ifmo.ru/cat/view.php/theory/unsorted/genetic-2005/
142. Analysis of variance (ANOVA). http://www.csse.monash.edu.au/~smarkham/resources/anova.htm
143. *Bogor*. Software Model Checking Framework. http://bogor.projects.cis.ksu.edu/
144. *Learning DFS from Noisy Samples*. A contest from GECCO 2004. http://cswww.essex.ac.uk/staff/sml/gecco/NoisyDFA.html
145. *Levent A. H., Mericli C., Mericli T. и др.* Cerberus'08 Team Report. http://www.tzi.de/spl/pub/Website/Teams2008/Cerberus.pdf

146. *LTL2BA* project. http://www.lsv.ens-cachan.fr/~gastin/ltl2ba/

147. The *R* Project for Statistical Computing. http://www.r-project.org/

Публикации автора

Статьи в журналах из перечня ВАК

148. *Царев Ф. Н.* Совместное применение генетического программирования, конечных автоматов и искусственных нейронных сетей для построения системы управления беспилотным летательным аппаратом //Научно-технический вестник СПбГУ ИТМО. 2008. Вып. 53. Автоматное программирование, с. 42 – 60. http://is.ifmo.ru/works/2008/Vestnik/53/03-genetic-neuro-automata-flying-plates.pdf

149. *Царев Ф. Н., Давыдов А. А., Соколов Д. О.* Применение генетического программирования и методов сокращенных таблиц переходов и деревьев решений для построения автоматов управления моделью беспилотного летательного аппарата //Научно-технический вестник СПбГУ ИТМО. 2008. Вып. 53. Автоматное программирование, с. 60 – 79. http://is.ifmo.ru/works/2008/Vestnik/53/04-genetic-automata-redused-transition-table-flying.pdf

150. *Царев Ф. Н.* Метод построения управляющих конечных автоматов на основе тестовых примеров с помощью генетического программирования // Информационно-управляющие системы. 2010. № 5, с. 31 – 36. http://is.ifmo.ru/works/_zarev.pdf

151. *Царев Ф. Н., Егоров К. В., Шалыто А. А.* Применение генетического программирования для построения автоматов управления системами со сложным поведением на основе обучающих примеров и спецификации // Научно-технический вестник Санкт-Петербургского государственного университета информационных технологий, механики и оптики. 2010. № 5, с. 81 – 86. http://is.ifmo.ru/works/_2010_10_13_egorov.pdf

152. *Царев Ф. Н., Александров А. В., Казаков С. В. Сергушичев А. А., Шалыто А. А.* Генерация конечных автоматов для управления моделью беспилотного самолета // Научно-технический вестник СПбГУ ИТМО. 2011. № 2, с. 3 – 11. http://is.ifmo.ru/works/2011/Vestnik/72-2/01-Aleksandrov-Kazakov-Tsarev-Shalyto.pdf

Другие статьи автора

153. *Царев Ф. Н., Егоров К. В., Шалыто А. А.* Совместное применение генетического программирования и верификации для построения автоматов управления системами со сложным поведением // Труды СПИИРАН. 2010. Вып. 15, с. 123 – 135.

Материалы конференций с участием автора

154. *Царев Ф. Н., Шалыто А. А.* Применение генетического программирования для построения мультиагентной системы одного класса /Международная научно-техническая мультиконференция «Проблемы информационно-компьютерных технологий и мехатроники». Материалы международной научно-технической конференции «Многопроцессорные вычислительные и управляющие системы» (МВУС`2007). Таганрог: НИИМВС. Т. 2, с. 46 – 51.

155. *Царев Ф. Н., Шалыто А. А.* Применение генетического программирования для генерации автоматов в задаче об «Умном муравье» /Сборник научных трудов. IV-я Международная научно-практическая конференция «Интегрированные модели и мягкие вычисления в искусственном интеллекте». М.: Физматлит. 2007, с. 590 – 597. http://is.ifmo.ru/genalg/_ant_ga.pdf

156. *Царев Ф. Н., Шалыто А. А.* О построении автоматов с минимальным числом состояний для задачи об «Умном муравье» //Сборник докладов X международной конференции по мягким вычислениям и изме-

рениям. СПбГЭТУ «ЛЭТИ». Т.2. 2007, с. 88 – 91. http://is.ifmo.ru/download/ant_ga_min_number_of_state.pdf

157. *Царев Ф. Н., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов с минимальным числом состояний для задачи об «Умном муравье» / Тезисы научно-технической конференции «Научно-программное обеспечение в образовании и научных исследованиях». СПбГПУ. 2008, с. 209 – 215. http://is.ifmo.ru/download/2008-02-25_tsarev_shalyto.pdf
158. *Tsarev F., Davydov A., Sokolov D.* Application of Genetic Algorithms for Construction of Moore Automation and Systems of Interacting Mealy Automata in «Artificical Ant» Problem / Proceeding of the Second Spring Young Researchers' Colloquium on Software Engineering (SYRCoSE'2008). SPbSU. 2008. Vol.1, pp. 51 – 54. http://is.ifmo.ru/genalg/_2008_07_03_ant.pdf
159. *Tsarev F., Davydov A., Sokolov D., Shalyto A.* Application of Genetic Programming for Generation of Controllers represented by Automata / Preprints of the 13th IFAC Symposium on Information Control Problems in Manufacturing. Moscow. Russia. 2009, pp. 684 – 689. http://is.ifmo.ru/articles_en/_ifac-2009.pdf
160. *Царев Ф. Н.* Построение автоматов управления системами со сложным поведением на основе тестов с помощью генетического программирования /Сборник докладов международной конференции по мягким вычислениям и измерениям (SCM`2009). СПбГЭТУ «ЛЭТИ». Т. 1, с. 231 – 234. http://is.ifmo.ru/works/_scm-2010_tsarev.pdf
161. *Царев Ф. Н.* Метод построения автоматов управления системами со сложным поведением на основе тестов с помощью генетического программирования / Материалы Международной научной конференции «Компьютерные науки и информационные технологии». Саратов: СГУ. 2009, с. 216 – 219.

162. *Tsarev F., Alexandrov A., Sergushichev A., Kazakov S.* Genetic Algorithm for Induction of Finite Automation with Continuous and Discrete Output Actions / Proceedings of the 2011 GECCO Conference Companion on Genetic and Evolutionary Computation. NY. : ACM. 2011, pp. 775 – 778. http://is.ifmo.ru/articles_en/2011/GECCO2011-Alexandrov-Kazakov-Sergushichev-Tsarev.pdf

163. *Царев Ф. Н., Егоров К. В.* Совместное применение генетического программирования и верификации моделей для построения автоматов управления системами со сложным поведением /Сборник трудов конференции «Информационные технологии и системы» (ИТИС`09). М.: Институт проблем передачи информации им. А. А. Харкевича РАН. 2009, с. 77 – 82. http://is.ifmo.ru/genalg/_2010_01_14_egorov_tsarev.pdf

164. *Царев Ф. Н., Александров А. В., Казаков С. В., Сергушичев А. А.* Генетическое 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. http://is.ifmo.ru/works/_scm-2010_autopilot.pdf

165. *Царев Ф. Н., Егоров К. В., Парфенов В. Г.* Совместное применение генетического программирования и верификации моделей для построения автоматов управления системами со сложным поведением /Труды XVII Всероссийской научно-методической конференции «Телематика`2010». Т. 2. СПбГУ ИТМО. 2010, с. 344, 345.

166. *Царев Ф. Н., Егоров К. В.* Применение генетического программирования для построения автоматов управления системами со сложным поведением на основе верификации моделей и обучающих примеров / Сборник «Список-2011». Материалы второй межвузовской научной

конференции по проблемам информатики». СПб.: ВВМ. 2011, с. 343 – 350. http://is.ifmo.ru/works/2011/SPISOK/07-Egorov-Tsarev.pdf

167. *Tsarev F., Egorov K.* Finite State Machine Induction using Genetic Programming Based on Testing and Model Checking / Proceedings of the 2011 GECCO Conference Companion on Genetic and Evolutionary Computation. NY.: ACM. 2011, pp. 759 – 762. http://is.ifmo.ru/articles_en/2011/GECCO2011-Tsarev-Egorov-FSM-induction.pdf
168. *Царев Ф. Н., Давыдов А. А., Соколов Д. О., Шалыто А. А.* Виртуальная лаборатория обучения генетическому программированию для генерации управляющих конечных автоматов / Сборник докладов III Международной научно-практической конференции «Современные информационные технологии и ИТ-образование». ВМК МГУ. М.: МАКС Пресс, 2008, с. 179 – 183. http://is.ifmo.ru/works/_2_93_davidov_sokolov.pdf
169. *Царев Ф. Н., Тяхти А. С., Чебатуркин А. А., Шалыто А. А.* Виртуальная лаборатория для обучения методам искусственного интеллекта для генерации управляющих конечных автоматов / Сборник докладов IV Международной научно-практической конференции «Современные информационные технологии и ИТ-образовние». М.: ИНТУИТ.РУ, МГУ. 2009, с. 222 – 227. http://is.ifmo.ru/works/_2010-10-01_tjahti.pdf
170. *Tsarev F., Chivilikhin D., Ulyantsev V.* Test-Based Extended Finite-State Machines Induction with Evolutionary Algorithms and Ant Colony Optimization / Proceedings of the 2012 GECCO Conference Companion on Genetic and Evolutionary Computation. NY. : ACM. 2012, pp. 603 – 606. http://is.ifmo.ru/articles_en/2012/GECCO12-Chivilikhin-Ulyantsev-Tsarev.pdf

ПРИЛОЖЕНИЕ 1. СВИДЕТЕЛЬСТВА О РЕГИСТРАЦИИ ПРОГРАММ ДЛЯ ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2010614197

Программное средство для построения управляющих конечных автоматов на основе обучающих примеров с использованием генетических алгоритмов

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

Автор(ы): ***Царев Федор Николаевич (RU)***

Заявка № **2010612486**
Дата поступления **5 мая 2010 г.**
Зарегистрировано в Реестре программ для ЭВМ
29 июня 2010 г.

Руководитель Федеральной службы по интеллектуальной собственности, патентам и товарным знакам

Б.П. Симонов

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2011615664

Программное средство для генерации конечных автоматов с дискретными и непрерывными выходными воздействиями

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

Автор(ы): ***Александров Антон Вячеславович, Казаков Сергей Владимирович, Сергушичев Алексей Александрович, Царев Федор Николаевич (RU)***

Заявка № **2011613852**
Дата поступления **25 мая 2011 г.**
Зарегистрировано в Реестре программ для ЭВМ
19 июля 2011 г.

Руководитель Федеральной службы по интеллектуальной собственности, патентам и товарным знакам

Б.П. Симонов

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2011616977

Виртуальная лаборатория для обучения методам искусственного интеллекта при построении конечных автоматов

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

Автор(ы): ***Тяхти Александр Сергеевич, Царев Федор Николаевич, Чебатуркин Александр Александрович (RU)***

Заявка № **2011615119**

Дата поступления **11 июля 2011 г.**

Зарегистрировано в Реестре программ для ЭВМ

8 сентября 2011 г.

Руководитель Федеральной службы по интеллектуальной собственности, патентам и товарным знакам

Б.П. Симонов

ПРИЛОЖЕНИЕ 2. ПРИМЕР ОТЧЕТА ПО ЛАБОРАТОРНОЙ РАБОТЕ

Отчет по лабораторной работе «Построение управляющих конечных автоматов с помощью эволюционных алгоритмов»

Выполнил: В. И. Ульянцев, гр. 3538

Учебный год: 2009-2010

Введение

Цель данной лабораторной работы – применение эволюционной стратегии для построения по тестам конечного автомата управления часами с будильником.

При выполнении работы использовался язык программирования *Java*.

Постановка задачи

Цель данной лабораторной работы – построение по тестам конечного автомата управления часами с будильником. В систему тестов для построения автомата управления часами с будильником включены тесты, описывающие его работу во всех трех режимах. Набор из 38 тестов взят из работы [1].

Структура хромосомы

Конечный автомат при использовании эволюционной стратегии представляется в виде объекта, содержащего множество состояний. Будем считать, что начальным состоянием автомата считается состояние с номером ноль, поэтому номер начального состояния в явном виде хранить не требуется. Каждое состояние описывается множеством переходов, инцидентных ему, и числом выходных воздействий, совершаемых при входе в данное состояние. Каждый переход описывается номером

состояния, в которое переходит автомат, и числом выходных воздействий, совершаемых на переходе.

Выбор представления графа переходов автомата с помощью списков ребер (в отличие от работы [2], 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 (граф переходов конечного автомата разрежен).

Параметром особи является максимальное число действий, совершаемых на переходах.

Рассматриваемый управляющий автомат является автоматом Мили), так как выходные воздействия вырабатываются только на переходах. Заметим, что автоматы Мура и Мили являются частным случаем автомата смешанного типа. При значении максимального числа действий, совершаемых на переходах автомата, равным нулю получим автомат Мура. При значении числа действий, совершаемых при входе в состояния, равным нулю получим автомат Мили.

В особи кодируется только «скелет» управляющего конечного автомата, а конкретные выходные воздействия, вырабатываемые на переходах и в состояниях, определяются с помощью алгоритма расстановки выходных воздействий (рис. 1).

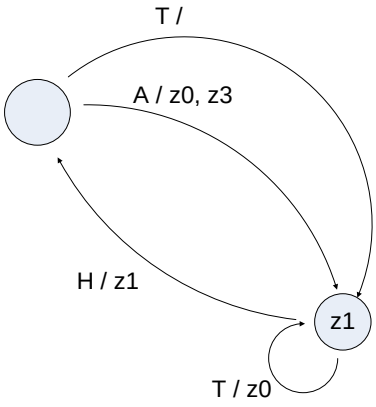


Рис. 1. Пример графа переходов управляющего автомата после применения алгоритма расстановки выходных воздействий

Операция мутации

При выполнении операции мутации с заданной вероятностью (по умолчанию, она равна 0.1) выполняется каждое из действий:

- изменение описания каждого из переходов (рис. 2);

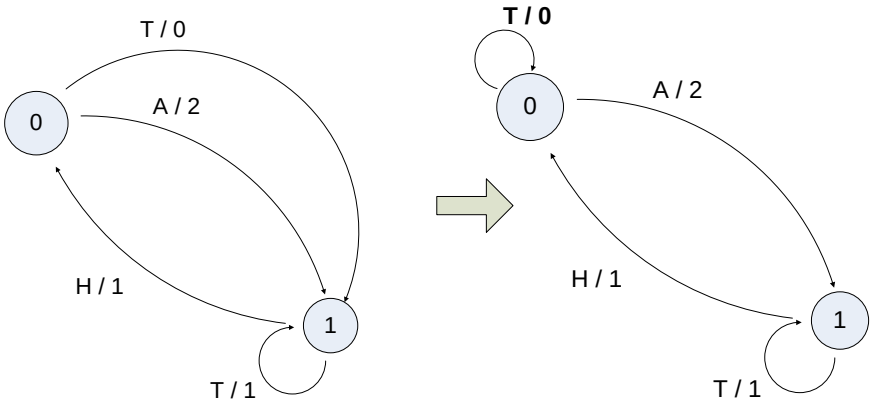


Рис. 2. Пример изменения описания перехода

- удаление или добавление перехода для каждого из состояний (рис. 3).

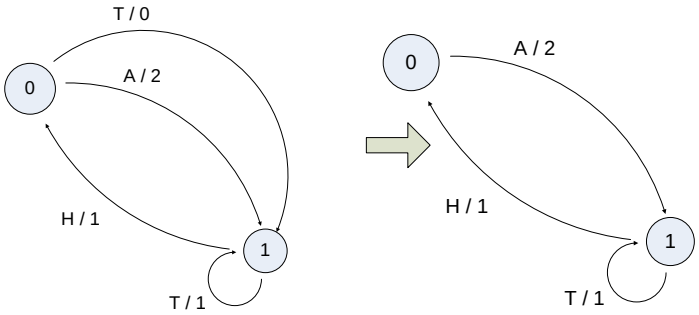


Рис. 3. Пример удаления перехода

При изменении описания перехода с некоторой вероятностью выполняется одно из следующих действий:

- изменение состояния, в которое ведет переход, – оно изменяется на случайно выбранное;
- изменение числа выходных воздействий, вырабатываемых на этом переходе, – с равной вероятностью изменяется на случайно выбранное число в диапазоне от нуля до максимально допустимого числа действий, совершаемых на переходах (предварительно заданная величина, по умолчанию равна пяти).

При изменении описания состояния выполняется изменение числа выходных воздействий, вырабатываемых при входе в вершину, – с равной вероятностью изменяется на случайно выбранное число в диапазоне от нуля до максимально допустимого числа действий, совершаемых при входе в вершину (предварительно заданная величина, по умолчанию равная трем).

Операция скрещивания

Скрещивание описаний автоматов производится следующим образом. Обозначим как *P1* и *P2* «родительские» особи, а как *S1* и *S2* – особи-«потомки». 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 состояний» с равной вероятностью может быть выбран один из двух методов.

При использовании *традиционного метода скрещивания* (рис. 4) списки переходов *S1.T[i]* и *S2.T[i]* строятся следующим образом:

1. Переходы состояния первого автомата *P1.T[i]* распределяются случайным образом между списками *S1.T[i]* и *S2.T[i]*.
2. Переходы состояния второго автомата *P2.T[i]* распределяются случайным образом между списками *S1.T[i]* и *S2.T[i]*. При этом, если в одном из списков уже присутствует соответствующий переход (из списка *P1.T[i]*), то переход добавляется в список переходов другого состояния.

Действия в самих состояниях распределяются также случайным образом.

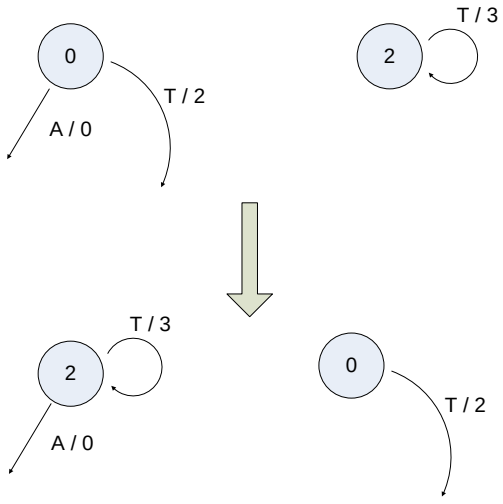


Рис. 4. Пример традиционного метода скрещивания для состояний

При использовании *метода скрещивания с учетом тестов* скрещивание производится аналогичным образом, за тем исключением, что некоторые переходы и состояния переходят напрямую из

«родительских» особей в особи-«потомки». Данные переходы и состояния определяются тестами (по умолчанию десятой частью всех тестов), на которых автоматы показали результат «лучше», чем на остальных тестах.

Функция приспособленности

Функция приспособленности основана на редакционном расстоянии. 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 для каждого теста вычисляется величина

$$FF_i = \begin{cases} \dfrac{0.1}{1 + ED(Output[i], Answer[i])}, ED(Output[i], Answer[i]) < 1 \\ 1, ED(Output[i], Answer[i]) = 1 \end{cases},$$

где ED*(A, B)* – 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 тесту, тем больше значение функции приспособленности.

Функция приспособленности особи зависит не только от того, насколько «хорошо» автомат работает на тестах, но и числа переходов, которые он содержит. Она вычисляется по формуле: $FF = \sum_{i=1}^{n} (FF_i \cdot (|Output[i]| + |Answer[i]|)) + \dfrac{0.001}{cnt}$, где как *cnt* обозначено число переходов в автомате. Эта функция приспособленности устроена таким образом, что при одинаковом уровне прохождения тестов преимущество имеет автомат, содержащий меньше переходов. Кроме этого, автомат, который хоть немного лучше проходит тесты, оценивается выше, чем автомат, проходящий тесты хуже, пусть даже и имеющий меньшее число переходов.

Вычислительные эксперименты

Построение конечного автомата управления часами с будильником проводилось при следующих параметрах эволюционной стратегии:

- размер параметра λ устанавливался в значения 1, 100, 1000;
- максимальное допустимое число действий в состояниях равно трем;
- максимальное допустимое число действий на переходах равно пяти;
- число состояний автомата равно четырем;

В результате работы эволюционной стратегии на одном из запусков был построен автомат, в котором из начального достижимы только три состояния из четырех. Если удалить недостижимое состояние, то этот граф переходов будет изоморфен построенному вручную.

График зависимости значения функции приспособленности от времени приведен на рис. 5 при одном из запусков алгоритма.

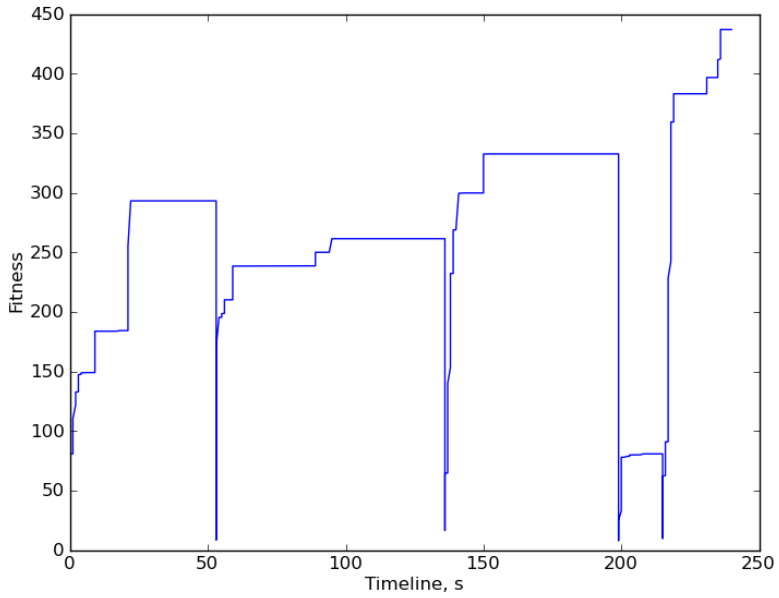


Рис. 5. Зависимость максимального значения функции приспособленности от времени работы эволюционной стратегии

Из графика видно, что требуемую особь удалось найти не сразу – до этого четыре раза достигался лишь локальный максимум, после чего эволюционный алгоритм перезапускался.

Источники

1. Государственный контракт «Разработка методов машинного обучения на основе генетических алгоритмов для построения управляющих конечных автоматов». Научно-технический отчет о выполнении этапа I. СПбГУ ИТМО, 2009. http://is.ifmo.ru/science/_nk-385-1-1.pdf

2. Lucas S. Evolving Finite-State Transducers: Some Initial Explorations. Lecture Notes in Computer Science. Springer Berlin / Heidelberg. Volume 2610/2003, pp. 241 – 257. http://www.springerlink.com/content/41a34vg70fp1hltb/