Федеральное агентство по образованию

УДК 004.4’242
ГРНТИ 20.01.01, 28.23.29
Инв. № 390163-1

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

НАУЧНО-ТЕХНИЧЕСКИЙ ОТЧЕТ

о выполнении 1 этапа Государственного контракта
№ П2174 от 09 ноября 2009 г.

Исполнитель: Государственное образовательное учреждение высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики»
Программа (мероприятие): Федеральная целевая программ «Научные и научно-педагогические кадры инновационной России» на 2009-2013 гг., в рамках реализации мероприятия № 1.3.2 Проведение научных исследований целевыми аспирантами.
Проект: Разработка методов машинного обучения на основе генетических алгоритмов для построения управляющих конечных автоматов
Руководитель организации: Васильев Владимир Николаевич
Руководитель проекта: Царев Федор Николаевич

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

Санкт-Петербург
2009 г.

СПИСОК ОСНОВНЫХ ИСПОЛНИТЕЛЕЙ

по Государственному контракту № П2174 от 09 ноября 2009 г.
на выполнение поисковых научно-исследовательских работ для государственных нужд

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

Руководитель темы:

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

Исполнители темы:

без ученой степени, без ученого звания — Егоров К. В.
_____
подпись, дата

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

без ученой степени, без ученого звания — Соколов Д. О.
_____
подпись, дата

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

РЕФЕРАТ

Отчет 58 с., 4 гл., 17 рис., 5 табл., 56 источников.

Ключевые слова: машинное обучение; конечные автоматы; конечны распознаватели; автомат Мура; автомат Мили; генетические алгоритмы; генетическое программирование; эволюционные алгоритмы.

В настоящем отчете излагаются результаты выполнения *первого этапа поисковых научно-исследовательских работ по направлению «Информатика» по проблеме «Разработка методов машинного обучения на основе генетических алгоритмов для построения управляющих конечных автоматов»*, выполняемых в рамках государственного контракта, заключенного между Федеральным агентством по образованию и государственным образовательным учреждением высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» в соответствии с решением *Единой комиссии (протокол от 13 октября 2009 г. № 3/НК-385П) по конкурсу № НК-385П* «Проведение поисковых научно-исследовательских работ по направлениям: «Информатика», «Механика», «Радиофизика, акустика и электроника», «География и гидрология суши», «Общая биология и генетика», «Геология. Горное дело» в рамках мероприятия 1.3.2 Программы» мероприятия 1.3.2 «Проведение научных исследований целевыми аспирантами» по направлению 1 «Стимулирование закрепления молодежи в сфере науки, образования и высоких технологий» федеральной целевой программы «Научные и научно-педагогические кадры инновационной России» на 2009 – 2013 годы» федеральной целевой программы «Научные и научно-педагогические кадры инновационной России» на 2009–2013 годы, утвержденной постановлением Правительства Российской Федерации от 28 июля 2008 года № 568 «О федеральной целевой программе «Научные и научно-педагогические кадры инновационной России» на 2009–2013 годы».

Целями настоящего этапа являются:

1. Выполнение аналитического обзора.
2. Выбор и обоснование оптимального варианта направления исследований.
3. Подготовка плана проведения теоретических и экспериментальных исследований.
4. Разработка, программная реализация и экспериментальное исследование метода машинного обучения на основе генетических алгоритмов для построения управляющих конечных автоматов Мили на основе тестовых примеров.

При выполнении первого этапа работ использовался следующий инструментарий:

1. Статьи в ведущих зарубежных и российских журналах, монографии и патенты за период 1998–2008 гг.
2. Результаты автоматического поиска в сети Интернет по ключевым словам и шаблонам.
3. Работы членов коллектива, опубликованные в рамках НИР по схожей тематике.
4. Аналитический обзор, составленный в рамках НИР.
5. Постановление Правительства Российской Федерации от 4 мая 2005 г. № 284 «О государственном учете результатов научно-исследовательских, опытно-конструкторских и технологических работ гражданского назначения».
6. Язык программирования *Java*.
7. Интегрированная среда разработки *Eclipse*.
8. Персональный компьютер.
9. ГОСТ 7.32-2001 «Отчет о научно-исследовательской работе. Структура и правила оформления».

Излагаются результаты выполнения аналитического обзора по следующим направлениям: машинное обучение; существующие алгоритмы машинного обучения для построения конечных автоматов, основанные на генетических алгоритмах; совместное применение генетических алгоритмов и верификации.

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

Описывается метод машинного обучения на основе генетических алгоритмов для построения управляющих конечных автоматов Мили. Приводится его программная реализация и результаты его экспериментальных исследований.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В настоящем отчете излагаются результаты выполнения *первого этапа поисковых научно-исследовательских работ по направлению «Информатика» по проблеме «Разработка методов машинного обучения на основе генетических алгоритмов для построения управляющих конечных автоматов»*, выполняемых в рамках государственного контракта, заключенного между Федеральным агентством по образованию и государственным образовательным учреждением высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» в соответствии с решением *Единой комиссии (протокол от 13 октября 2009 г. № 3/НК-385П) по конкурсу № НК-385П* «Проведение поисковых научно-исследовательских работ по направлениям: «Информатика», «Механика», «Радиофизика, акустика и электроника», «География и гидрология суши», «Общая биология и генетика», «Геология. Горное дело» в рамках мероприятия 1.3.2 Программы» мероприятия 1.3.2 «Проведение научных исследований целевыми аспирантами» по направлению 1 «Стимулирование закрепления молодежи в сфере науки, образования и высоких технологий» федеральной целевой программы «Научные и научно-педагогические кадры инновационной России» на 2009 – 2013 годы» федеральной целевой программы «Научные и научно-педагогические кадры инновационной России» на 2009-2013 годы, утвержденной постановлением Правительства Российской Федерации от 28 июля 2008 года № 568 «О федеральной целевой программе «Научные и научно-педагогические кадры инновационной России» на 2009–2013 годы».

Целями настоящего этапа являются:

1. Выполнение аналитического обзора.
2. Выбор и обоснование оптимального варианта направления исследований.
3. Подготовка плана проведения теоретических и экспериментальных исследований.
4. Разработка, программная реализация и экспериментальное исследование метода машинного обучения на основе генетических алгоритмов для построения управляющих конечных автоматов Мили на основе тестовых примеров.

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

- машинное обучение;
- существующие алгоритмы машинного обучения для построения конечных автоматов, основанные на генетических алгоритмах;
- совместное применение генетических алгоритмов и верификации.

Во второй главе приводится выбор и обоснование оптимального варианта направления исследований.

В третьей главе приводится план проведения теоретических и экспериментальных исследований.

В четвертой главе приводится описание метода машинного обучения на основе генетических алгоритмов для построения управляющих конечных автоматов Мили на основе тестовых примеров, его программная реализация на языке программирования *Java*, а также результаты экспериментального исследования.

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

Машинное обучение – раздел искусственного интеллекта, изучающий методы построения алгоритмов, способных обучаться. Как правило, рассматриваются два типа алгоритмов машинного обучения – реализующие обучение на основе обучающих примеров и реализующие обучение на собственных ошибках. Кроме этого, можно рассматривать еще один тип алгоритмов – обучение на

основе спецификации. Ряд алгоритмов машинного обучения нацелен на работу с моделями, содержащими состояния. Наиболее часто в качестве таких моделей используются скрытые Марковские модели.

Эти модели имеют вероятностный характер, для их обучения существует ряд алгоритмов (например, алгоритм Баума-Велча). Наиболее близкими к ним моделями, имеющими детерминированный характер, являются конечные автоматы.

Как правило, рассматривают следующие типы конечно-автоматных моделей: конечные автоматы-распознаватели, конечные автоматы-преобразователи, управляющие конечные автоматы, которые могут быть автоматами Мили, автоматами Мура или смешанными. Для обучения таких моделей могут применяться алгоритмы, реализующие все три типа обучения – на собственных ошибках, на примерах и на основе спецификации.

Исследования, проводимые, например, в университете Кента (Великобритания), в университете Эссекса (Великобритания), а также в центре искусственного интеллекта научно-исследовательской лаборатории военно-морского флота (Вашингтон, США) показали, что методы машинного обучения, основанные на генетических и эволюционных алгоритмах, на ряде задач эффективнее методов, построенных на других принципах. В настоящее время работы по построению конечных автоматов на основе генетических алгоритмов ведутся также в ряде зарубежных университетов, в том числе в Массачусетском технологическом институте и Университете Южной Калифорнии. Кроме этого, работы по этому направлению проводились в СПбГУ ИТМО в рамках исследования по теме «Технология генетического программирования для генерации автоматов управления системами со сложным поведением».

Отметим, что в работах, которые проводятся в указанных университетах, конечные автоматы-распознаватели и конечные автоматы-преобразователи, как правило, строятся на основе примеров, а управляющие конечные автоматы – на основе обучения на собственных ошибках. Кроме этого, известна работа по построению управляющих конечных автоматов на основе спецификации с использованием методов верификации моделей. Однако предлагаемый в этой работе алгоритм позволяет строить автоматы только достаточно простой структуры, что ограничивает область его применения.

Таким образом, в существующих в настоящее время работах построение управляющих конечных автоматов осуществляется без учета знаний человека (которые могут быть выражены в наборе обучающих примеров или в спецификации). Следовательно, целесообразна разработка методов машинного обучения для построения управляющих конеченых автоматов на основе обучающих примеров или спецификации. Эти методы будут разрабатываться на основе генетических и эволюционных алгоритмов, которые показали свою эффективность при построении конечных автоматов других типов.

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

1. АНАЛИТИЧЕСКИЙ ОБЗОР

В настоящем разделе приводятся результаты аналитического обзора. Аналитический обзор проводился по следующим направлениям:

- машинное обучение;
- алгоритмы машинного обучения для построения конечных автоматов, основанные на принципах эволюционных вычислений;
- совместное применение генетических алгоритмов и верификации.

1.1. МАШИННОЕ ОБУЧЕНИЕ

В настоящем разделе приводится аналитический обзор методов и алгоритмов машинного обучения, которые ориентированы на модели, основанные на состояниях.

1.1.1. Общие сведения

В соответствии с книгой [18] алгоритм является алгоритмом машинного обучения, если он улучшает свое поведение по мере накопления опыта. Это означает, что алгоритм обучает параметры модели либо на заранее подготовленных тестовых примерах, либо на собственных ошибках, и со временем решает поставленную задачу все лучше и лучше. Некоторые алгоритмы машинного обучения способны подмечать ранее неизвестные закономерности в данных, выделять знания, которых раньше не было.

Модели, основанные на состояниях, широко применяются в машинном обучении. Примером таких моделей являются скрытые Марковские модели. Они могут применяться в таких задачах, как распознавание речи, и для них разработан ряд алгоритмов обучения. Область их применения ограничивается тем, что эти модели имеют вероятностный характер. Наиболее близкими к ним моделями, имеющими детерминированный характер, являются конечные автоматы.

Как правило, в машинном обучении конечные автоматы рассматриваются как распознаватели регулярных языков или как преобразователи слов одного регулярного языка в слова другого. В случае построения управляющих автоматов обычно рассматриваются системы с малым числом входных переменных (одной или двумя), что существенно ограничивает область применения таких алгоритмов. Кроме этого, существующие алгоритмы машинного обучения для конечных автоматов основаны на принципе обучения «на собственных ошибках» и не являются достаточно эффективными для ряда задач, так как не позволяют эффективно учитывать знания человека и строить автоматы с гарантированным поведением.

1.1.2. Построение скрытых Марковских моделей

Скрытая Марковская модель [57] – это статистическая модель, в которой моделируемая система представляется в виде Марковской цепи с ненаблюдаемым состоянием. В отличие от обычных Марковских моделей внешний наблюдатель видит не состояние, а выходной символ, зависящий от состояния. При этом в каждом состоянии задано свое распределение вероятностей появления выходных символов. Скрытая Марковская модель $\lambda = (A, B, \pi)$ содержит три компонента: A – матрица вероятностей переходов между состояниями, B – матрица вероятностей генерации выходных символов, π – начальное распределение вероятностей.

Обычно рассматривают три задачи, связанные со скрытыми Марковскими моделями:

- по заданным параметрам модели и последовательности выходных символов найти вероятность того, что с ее помощью будет сгенерирована эта последовательность;

- по заданным параметрам модели и последовательности выходных символов найти наиболее вероятную последовательность состояний, которая привела к генерации этой последовательсности;
- по заданной выходной последовательности (или набору выходных последовательностей) построить модель, которая с наибольшей вероятностью (по сравнению с другими моделями) будет генерировать эти последовательности.

Первая задача решается с помощью динамического программирования, вторая – с помощью алгоритма Витерби, а третья – с помощью алгоритма Баума-Велча. Алгоритм Баума-Велча является разновидностью EM-алгоритма [39].

1.1.3. Генетические алгоритмы и эволюционные вычисления

В этом разделе приведено краткое описание *традиционного генетического алгоритма* (*conventional genetic algorithm*) [51] с одноточечной рекомбинацией и мутацией. В этом алгоритме каждая особь представляет собой битовую строку длины L, а размер популяции равен n.

Шаг 1. Генерации начальной популяции $\{x_i\}_{i=1}^{n}$. Для этого, например, можно сгенерировать n битовых строк, в которых каждый из L битов равен *0* или *1* с одинаковой вероятностью.

Шаг 2. Вычислить функцию приспособленности $f(x)$ для каждой особи из текущей популяции. Этот шаг обычно является наиболее трудоемким по времени во всем алгоритме. Отметим, что при вычислении функции приспособленности, как правило, необходимо осуществить декодирование некоторого объекта из двоичной строки.

Шаг 3. Переход к следующему поколению популяции. При создании нового поколения могут использоваться различные стратегии. В качестве примера приведем так называемый *метод рулетки* (*roulette wheel selector*) [6].

Создание нового поколения в этом случае разбивается на n итераций, на каждой из которых в популяцию добавляется одна особь. Для этого случайно с вероятностями, пропорциональными их функция приспособленности, из текущего поколения популяции выбираются две родительские особи x и y.

После этого с некоторой наперед заданной вероятностью происходит скрещивание (рекомбинация, кроссовер, кроссинговер) родительских особей. В результате образуются их «потомки» – z_1 и z_2. Происходит это следующим образом: случайно (с равномерным распределением на множестве $\{1, 2, \ldots, L\}$) выбирается число k. Хромосомы «потомков» z_1 и z_2 теперь строятся следующим образом: для z_1 – первые k символов совпадают с первыми k символами x, последние $(L-k)$ – с последними $(L-k)$ символами y, для z_2 – наоборот. Например, если $L=7$, $x=1001010$, $y=0001001$, $k=3$, то $z_1=1001001$, $z_2=0001010$. После этого равновероятно выбирается одна из особей z_1, либо z_2. Выбранную особь обозначим как z.

С некоторой (как правило, порядка *0.01*) вероятностью производится мутация особи z – в ней случайно выбирается один бит и изменяется.

Получившаяся в результате особь z добавляется в следующее поколение.

Шаг 4. Повторять шаги 2 и 3 до тех пор, пока не будет выполнено условие останова (максимальная приспособленность в популяции достигла целевого значения, прекратился рост максимальной приспособленности, число поколений достигло некоторого предела и т.д.).

Отметим, что кроме описанных выше методов одноточечного кроссовера, алгоритма рулетки и мутации изменением случайного бита, существуют и другие. Обзор приведен в работе [19].

Генетическое программирование (*genetic programming*), предложенное J.R. Koza в 1992 году [45], предполагает применение генетических алгоритмов для автоматизированного построения программ.

Основным отличием генетического программирования от традиционных генетических алгоритмов является способ кодирования особей. Если в генетическом алгоритме особи кодируются с помощью битовых строк, то в генетическом программировании используется более *высокоуровневое* представление: деревья разбора, тексты программ на языках программирования с несложной структурой, диаграммы переходов конечных автоматов и т. д.

Такой подход позволяет определить генетические операции скрещивания и мутации, которые лучше подходят для решаемой задачи. Например, если каждая особь представляет собой программу, то возможны так называемые *операции, изменяющие архитектуру* (*architecture-altering operations*), – например, добавление подпрограммы [23].

Генетические алгоритмы и генетическое программирование основаны на моделировании процесса эволюции и являются частью более общей группы методов – эволюционных вычислений [38].

1.1.4. Типы автоматных моделей

В настоящем разделе описываются типы автоматных моделей. Абстрактные конечные автоматы принято описывать в следующих терминах. Задано конечное множество символов *X*, которое называется (входным) алфавитом. Множество всех возможных цепочек (последовательностей, строк, слов), составленных из символов алфавита *X*, обозначается *X**. Пустая последовательность символов обозначается ε, $\varepsilon \in X^*$. Подмножество *L* множества всех цепочек над алфавитом *X*, $L \subset X^*$ называется языком. Рассматривается следующая проблема: задан язык $L \subset X^*$ и цепочка $\xi \in X^*$. Необходимо определить, принадлежит ли указанная цепочка языку ($\xi \in L$)?

Если абстрактный вычислитель способен решить эту проблему для определенного языка $L \subset X^*$ и произвольной строки $\xi \in X^*$, то говорят, что вычислитель распознает язык *L*. Таким образом, абстрактные автоматы описываются в терминах тех языков, которые они распознают. Различные автоматные модели могут распознавать разные классы языков или, другими словами, обладают разной вычислительной мощностью (вычислительная мощность модели абстрактных автоматов тем больше, чем шире класс распознаваемых ими языков).

1.1.4.1. Конечные автоматы-распознаватели

Детерминированный конечный автомат (ДКА) – это пятерка $\langle X, Y, \delta, y_0, F \rangle$, где *X* – конечный алфавит входных символов, *Y* – конечное множество состояний, $\delta : X \times Y \to Y$ – функция переходов, $y_0 \in Y$ – начальное (стартовое) состояние, $F \subset Y$ – множество допускающих состояний.

Расширенная функция переходов $\hat{\delta} : X^* \times Y \to Y$, сопоставляющая новое состояние текущему состоянию и цепочке символов, определяется индуктивно следующим образом [31]:

$$\forall y \in Y \left(\hat{\delta}(\varepsilon, y) = y\right);$$

$$\forall y \in Y \; \forall \xi \in X^* \; \forall x \in X \left(\hat{\delta}(\xi x, y) = \delta(x, \hat{\delta}(\xi, y))\right).$$

В таком случае, если $\hat{\delta}(\xi, y_0) \in F$ – стартуя в начальном состоянии и обработав цепочку ξ, автомат оказывается в одном из допускающих состояний, то говорят, что он допускает эту цепочку. Множество допускаемых цепочек образует язык *L*, распознаваемый детерминированным конечным автоматом (ДКА): $L = \left\{\xi \in X^* \mid \hat{\delta}(\xi, y_0) \in F\right\}$.

Класс языков, распознаваемых ДКА, называют регулярными языками. Известно, что он совпадает с классом языков, описываемых регулярными выражениями и автоматными грамматиками [31].

1.1.4.2. Конечные автоматы-преобразователи

Для того чтобы наделить модель абстрактного конечного автомата способностью не только давать ответ типа «да/нет», но и выполнять какие-то преобразования, в модель добавляют конечный алфавит выходных символов Z и функцию выходов φ. Если функция выходов имеет вид $\varphi: X \times Y \to Z$, то вычислитель называется автоматом Мили, а если $\varphi: Y \to Z$ – автоматом Мура. Таким образом, рассматривается шестерка $\langle X, Y, Z, \delta, \varphi, y_0 \rangle$. Она определяет автоматное отображение $f: X^* \to Z^*$ (преобразование, выполняемое автоматом) следующим образом:

$$f(\varepsilon) = \varepsilon$$
$$\forall \xi \in X^* \ \forall x \in X \left(f(\xi x) = f(\xi) \varphi(x, \hat{\delta}(\xi, y_0)) \right)$$

Автоматные отображения – это отображения «без предсказания»: перерабатывая цепочку слева направо, они «не заглядывают вперед». Например, отображение, которое сопоставляет цепочке ее саму, записанную в обратном порядке, не является автоматным.

1.1.4.3. Управляющие конечные автоматы

Понятие управляющего (или структурного) автомата Мура аналогично понятию абстрактного автомата Мура, введенному выше. В таком автомате выходное воздействие зависит только от состояния и не зависит от входного воздействия. Структурные схемы автоматов Мура первого и второго рода приведены на рис. 1. Автомат первого рода формирует выходные воздействия на основе текущих (не обновленных) значений внутренних переменных, а автомат второго рода, наоборот, сначала обновляет состояние, а затем использует его новое значение для вычисления выходного воздействия.

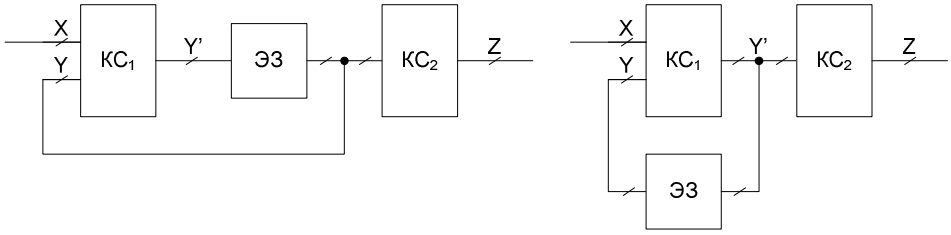

Рис. 1. Автоматы Мура: первого рода (слева) и второго рода (справа)

На приведенном рисунке функция переходов δ и функция выходов φ вычисляются соответственно схемами $КС_1$ и $КС_2$, а как ЭЗ обозначен элемент задержки. Подобное изображение связано с тем, что первоначально такие автоматы реализовывались аппаратно в виде набора логических схем.

На рис. 2 приведен пример графа переходов автомата Мура.

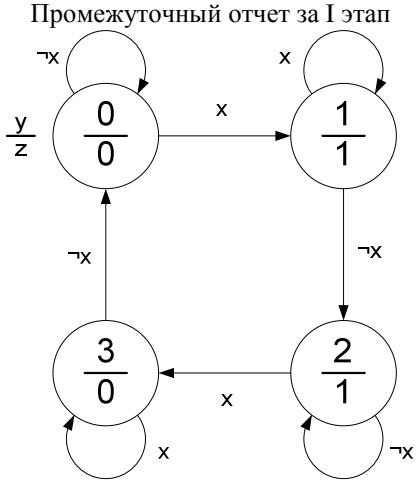

Рис. 2. Пример графа переходов автомата Мура

По аналогии с абстрактными автоматами, структурный автомат, выходные воздействия которого зависят не только от состояния, но и от входных воздействий, называется автоматом Мили (рис. 3).

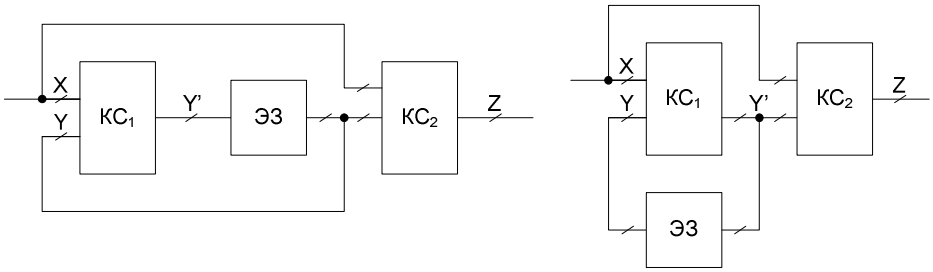

Рис. 3. Автоматы Мили: первого рода (слева) и второго рода (справа)

Известно [30], что для любого автомата Мили можно построить эквивалентный ему автомат Мура. Число состояний в таком автомате будет не меньше, чем в исходном.

В качестве примера рассмотрим последовательный двоичный одноразрядный сумматор, который выполняет сложение одноименных бит двух двоичных чисел с учетом переноса. Результатом работы сумматора является одноименный бит суммы и перенос в следующий разряд. На рис. 4 приведен автомат Мили, реализующий это «устройство».

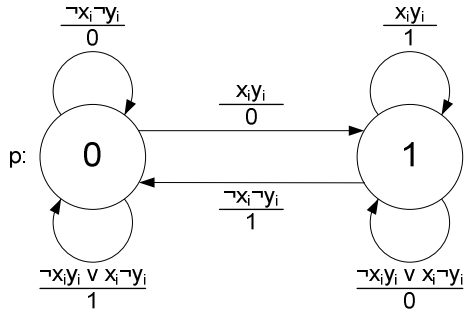

Рис. 4. Автомат Мили последовательного двоичного одноразрядного сумматора

На этом рисунке x_i и y_i – складываемые биты, состояния соответствуют значениям переноса, а значение бита суммы формируется как выходное воздействие на переходах.

Если часть выходных переменных автомата зависит только от состояний, а остальные – также и от входных воздействий, удобно разделить функцию выходов φ на две составляющие: $\varphi_1(y)$ и

$\varphi_2(x, y)$. В структурную схему автомата в этом случае вводится не один, как в автоматах Мура и Мили, а два выходных преобразователя (рис. 5). Такие автоматы называются смешанными, автоматами Мура-Мили или С-автоматами.

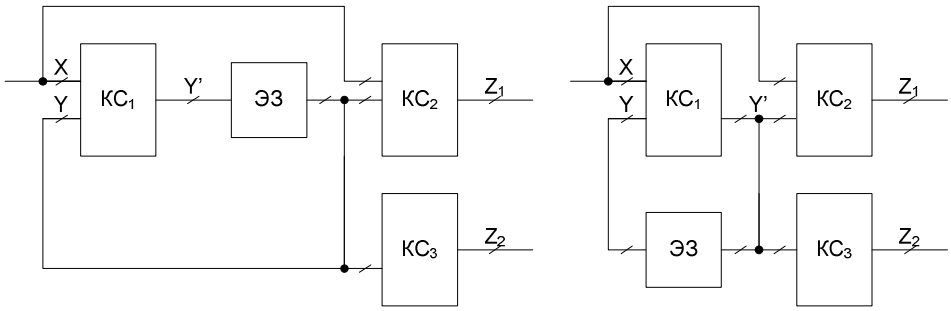

Рис. 5. Смешанные автоматы: первого рода (слева) и второго рода (справа

1.1.5. Совместное применение генетических алгоритмов и скрытых Марковских моделей для тестирования конечных автоматов

В работе [3] разрабатывается метод совместного применения генетических алгоритмов и скрытых Марковских моделей для тестирования конечных автоматов. В указанной работе рассматривается поиск ошибок типа «неучтенный переход» в конечных автоматах. Для этого на автомат подаются заранее заданные последовательности, записываются полученные выходные последовательности, и на их основе строится скрытая Марковская модель. При этом рассматриваются только так называемые «сильно детерминированные» модели, в которых из каждого состояния с вероятностью, близкой к единице, выполняется один переход, а вероятности других переходов из этого состояния близки к нулю.

Вычислительные эксперименты, проведенные в рамках указанной работы, показали, что сходимость алгоритма Баума-Велча (который является численным методом) сильно зависит от начальных параметров. Эти параметры и предлагается подбирать с помощью генетического алгоритма.

Особь в рассматриваемом генетическом алгортме представлялась в виде массива размерностью N на два, каждый элемент которого – целое число от 0 до N–1. Здесь N – число состояний. Таким образом, для каждого состояния записываются два перехода. Первоначально их вероятности принимаются равными. Далее они изменяются в процессе работы алгоритма Баума-Велча.

В рассматриваемой работе применяются стандартные генетические операции – отбор, мутация и скрещивание. Операция отбора организована таким образом, что особи сортируются по убыванию приспособленности, а затем выбираются пропорционально полученному после сортировки номеру. Такой вид отбора называется «*rank selection*» и подробно описан в работе [51]. Отбор пропорциональный значению приспособленности («*roulette wheel selection*») также был опробован в этой работе, но дал несколько худшие результаты.

Оператор мутации выполняется стандартным образом – с небольшой вероятностью (в проведенных экспериментах она равнялась 0.1) каждое из чисел хромосомы изменяется на целое число от 0 до N – 1. Новое значение может совпадать с предыдущим.

Скрещивание также осуществляется стандартным образом – применяется одноточечное скрещивание («*single-point crossover*»), рассмотренное в книге [51]. Единственное отличие состоит в том, что с некоторой вероятностью (в проведенных экспериментах равной 0.3) особи не подвергаются скрещиванию, а случайным образом выбирается одна из них и возвращается как результат скрещивания.

Приспособленность $f(x)$ особи x, задающей некоторую модель $\lambda = (A, B, \pi)$, определялась как $f(x) = -\frac{1}{\ln P(O \mid \lambda)}$, где знаменатель – логарифм вероятности пронаблюдать данные O в модели λ.

Результаты вычислительных экспериментов показали, что поиск неучтенных переходов выполнялся генетическими алгоритмами, а частота их использования (в ходе работы автомата) определялась алгоритмом Баума-Велша. Следовательно, основная заслуга в успешном построении моделей максимального правдоподобия для рассматриваемого случая принадлежит генетическим алгоритмам.

1.2. Существующие алгоритмы машинного обучения для построения конечных автоматов, основанные на принципах эволюционных вычислений

В настоящем разделе приводится обзор существующих алгоритмов машинного обучения для построения конечных автоматов, основанных на генетических алгоритмах.

1.2.1. Методы машинного обучения на основе генетических и эволюционных алгоритмов, использующие обучающие примеры

В настоящем разеделе приводится обзор алгоритмов машинного обучения на основе генетических и эволюционных алгоритмов.

1.2.1.1. Построение конечных распознавателей

Из теории формальных языков известно, что конечные детерминированные автоматы способны распознавать регулярные языки [31]. В связи с этим актуальна задача построения автомата, распознающего по множеству примеров некий язык. Для решения этой задачи успешно применялись различные генетические алгоритмы.

Задача может быть усилена до построения автомата с минимальным числом состояний. Однако, в работе [40] показано, что эта задача является *NP*-трудной.

В работе [47] производится сравнение алгоритмов машинного обучения для построения конечных автоматов-распознавателей. При этом рассматриваются два типа алгоритмов: эволюционные и основанные на построении префиксного дерева (бора) с дальнейшим объединением состояний на основе свидетельств (*Evidence-Driven State Merging – 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 n состояний равно n^{2n}. Если учитывать то, что каждое состояние автомата может быть допускающим или недопускающим, то общий размер пространства поиска равен $2^n \cdot n^{2n}$.

Для сокращения пространства поиска в работе [47] 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 $h[f(t)][c]$, где $f(t)$ – номер состояния, в которое приходит рассматриваемый

конечный автомат после обработки строки *t*, а число *c* равно единице, если строка s должна допускаться автоматом, и равно нулю, если она не должна допускаться. 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 n^{2n}.

В качестве эволюционного алгоритма в рассматриваемой работе применяется так называемая (1, 1)-эволюционная стратегия. Этот алгоритм не использует операцию скрещивания. Его работа осуществляется следующим образом: хранится текущее решение и на каждой итерации к текущему решению применяется операция мутации. Если она приводит к увеличению значения функции приспособленности, то эта мутация принимается и текущее решение заменяется новым. В противном случае, текущее решение остается без изменений. Если за заданное заранее число итераций не происходит увеличение значения функции приспособленности, то текущее решение записывается, а процесс поиска решения перезапускается. Отметим, что рассматриваемый алгоритм не использует операцию скрещивания.

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 примеров. Первый набор тестовых данных предложен в работах [50, 56]. На этом наборе сравнение проводилось не только с алгоритмом *EDSM*, но и с алгоритмом генетического программирования, предложенным в работе [50]. Это сравнение показывает, что разработанный алгоритм эволюционного программирования превосходит и алгоритм *EDSM*, и алгоритм генетического программирования, 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* – в среднем 37 миллисекунд. Вычислительные эксперименты проводились на компьютере с процессором Pentium IV с тактовой 2.4 ГГц для реализаций на языке программирования *Java*.

Второй набор тестовых данных строился следующим образом:

1. Выбиралось число *n*.
2. Генерировался случайный ориентированный граф из $5n/4$ 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 конечного автомата определялось, допускается она или нет.

На этом тестовом наборе разработанный эволюционный алгоритм при $n = 4, 8, 16$ превосходил алгоритм *EDSM*, но при $n = 32$ 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 обучающее множество примеров содержит небольшую долю строк заданной длины.

В работе [48] также рассматривается задача построения конечных автоматов-распознавателей с помощью эволюционных алгоритмов. Предлагаемый в рассматриваемой работе эволюционный алгоритм является улучшенной версией алгоритма, предложенного в работе [47].

Конечный распознаватель в рассматриваемой работе представляется так же, как и в работе [47] – в хромосому входит таблица переходов, а пометки на состояниях расставляются с помощью специального алгоритма. В качестве эволюционного алгоритма в работе [48] также используется (1+1)-эволюционная стратегия, но операцию мутации предлагается выполнять не так, как в работе [47].

В этой работе 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 $c(t)$. Число строк, неправильно классифицированных автоматом и при обработке которых использовался переход t, обозначается как $e(t)$. После этого вероятность того, что переход t будет выбран при выполнении операции мутации, вычисляется по следующей формуле: $p(t) = \frac{e(t)}{c(t) + e(t)}$. При этом считается, что $p(t)$=0, если $e(t)$=0. Такой подход к выполнению операции мутации требует выполнения двух проходов для вычисления значения функции приспособленности и указанных вероятностей. На первом из этих проходов неизвестно, какие состояния являются допускающими, а какие – нет, так как он необходим для алгоритма пометки состояний. На втором известно, какие состояния являются допускающими. Поэтому указанные вероятности могут быть вычислены.

Кроме описанного способа выполнения операции мутации в рассматриваемой работе также предлагается следующий – 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 машинного обучения в рассматриваемой работе применялось два набора тестовых данных. Первый строился так же, как второй набор тестовых данных в работе [47]. Второй набор тестовых данных был зашумленным – для некоторых строк из обучающего набора было неправильно указано, должны они допускаться автоматом или нет.

На первом наборе тестовых данных проводилось сравнение четырех эволюционных методов машинного обучения:

- метода, который не использует алгоритм расстановки пометок (*plain*);
- метода из работы [47] (*smart labeling*);
- метода, использующего первый из предлагаемых в настоящей работе подходов к выполнению операции мутации (*sampled*);
- метода, использующего второй из предлагаемых в настоящей 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 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* – пять из 50, *smart* – 21 из 50, *sampled* – 26 из 50, *quick-sampled* – четыре из 50. Если же в исходном автомате было 32 состояния, то

только два алгоритма справились с задачей: *smart* правильно строил автомат в шести случаях из 50, а *sampled* – в 12 случаях из 50.

На втором наборе тестовых данных, который содержал зашумленные обучающие наборы. На этом наборе данных сравнение проводилось с алгоритмом *EDSM* и с другими алгоритмами, участвовавшими в соревновании *GECCO 2004* [46]. Результаты вычислительных экспериментов показали, что предлагаемый в рассматриваемой работе эволюционный алгоритм (*sampled*) превосходит все существующие методы машинного обучения конечных автоматов на этом обучающем наборе по точности построения автоматов. 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 [37] автоматы представлялись с помощью таблицы переходов. Разработанный в этой работе метод позволяет поддерживать в популяции автоматы с разным числом состояний. Функция приспособленности учитывает три компонента – число верно распознанных тестовых примеров, число состояний и переходов автомата, степень общности языка, соответствующего построенному автомату. Это позволяет сузить область поиска, и находить языки, соответствующие определенным критериям. В обучающий набор могут входить примеры слов, которые как принадлежат, так и не принадлежат языку.

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

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

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

В работе [55] также рассматривается задача построения конечных автоматов-распознавателей. В качестве метода представления конечных автоматов выбраны двоичные строки, в качестве операций мутации и скрещивания – 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 применяется метод рулетки.

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

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

1.2.1.2. Построение конечных преобразователей

По построению конечных преобразователей на основе обучающих примеров существует намного меньше работ, чем по построению конечных распознавателей. Одной из них является работа [49]. Автоматы-преобразователи в ходе обработки символов входного слова при выполнении переходов записывают в выходной поток символы выходного алфавита, тем самым осуществляя преобразование слов одного формального языка в слова другого.

Так же, как и в рассмотренных выше работах [47, 48], посвященных построении конечных автоматов-распознавателей, в работе [49] применяется (1, 1)-эволюционная стратегия. Конечный преобразователь представляется в виде набора двух таблиц – таблицы значений функции переходов и таблицы 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 (*strict*);
- на основе вычисления расстояния Хэмминга [41] (*hamming*);
- на основе вычисления редакционного расстояний (расстояния Левенштейна) [14] (*edit*).

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 t, выдал строку q. При использовании строго сравнения значение функции приспособленности f будет вычисляться по следующей формуле: $f = \begin{cases} 0, t = q \\ 1, t \neq q \end{cases}$.

При использовании расстояния Хэмминга функция приспособленности вычисляется по формуле: $f = \frac{\sum_{i=1}^{\min(|t|,|q|)} \Delta(t_i, q_i)}{\max(|t|,|q|)}$, где как $|t|$ и $|q|$ обозначены длины строк t и q соответственно. Как Δ в формуле обозначена функция, значение которой равно нулю, если значения ее аргументов совпадает, и единице – в противном случае.

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

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

Результаты экспериментов показали, что наилучшие результаты на «сложном» тестовом наборе 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 *strict* и *hamming* меняются местами.

На основании полученных в рассматриваемой работе результатов сделан вывод, что разработка методов машинного обучения на основе эволюционных (в частности, генетических) алгоритмов, является перспективной областью.

1.2.2. Построение управляющих конечных автоматов на основе принципа «обучения на собственных ошибках»

В работе [54] рассматривается задача построения управляющего конечного автомата на основе принципа «обучения на собственных ошибках». В указанной работе 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 клектах (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 принадлежащеми либо первому агенту, либо второму. Победителем признается тот агент, которому в 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 конечного автомата для управления первым агентом, который позволял бы достаточно часто выигрывать игру. Стратегия второго агента при этом была такой: на каждом ходе при наличии хотя бы одной никому не принадлежащей соседней клетки, премещение 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 [54] применяется генетический алгоритм. Конечный автомат представляется в виде таблицы значений функции переходов и таблицы значений функции действий. В работе рассматривалось построение автоматов, содержащих от одного до десяти состояний. При построении автомата из n состояний размер таблиц значений указанных функции составляет $n \times 80$, так как существует 80 комбинаций входных переменных. Это объяняется тем, что имеется четыре переменные, каждая из которых может принимать три возможных значения. 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 действий. При этом с вероятностью p изменялось состояние, в которое ведет переход, а с вероятностью 1 - p – действие, выполняемое на этом переходе. Кроме этого, использовалась операция однородного скрещивания. Для каждой позиции (i, j) в таблицах значений функций переходов и действий 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 конечным 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 10000 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 пропорциональна значению функции приспособленности.

Вычислительные эксперименты проводились при значениях n = 1..10. 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 конечных автоматов.

Анализ поведения агентов, управляемых построеннымы автоматами, показал, что поведение агента достаточно быстро зацикливается, что приводит к ухудшению его результатов. Для того, чтобы устранить этот недостаток, в рассматриваемой работе был предложен 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 [52].

В рамках исследований по теме «Технология генетического программирования для генерации автоматов управления системами со сложным поведением» на кафедре «Технологии

программирования» СПбГУ ИТМО был разработан ряд методов генерации конечных автоматов с помощью генетических алгоритмов [24–27].

Один из этих методов – метод сокращенных таблиц переходов был предложен в работе [20]. Этот метод позволяет решить проблему экспоненциального роста размера описания автомата, которая возникает при использовании полных таблиц переходов, так как число строк в таблице равно 2^n, где n – число предикатов. Опыт показывает, что в реальных задачах управляющие автоматы, построенные вручную, имеют гораздо меньше переходов, чем $|S| \cdot 2^n$ (здесь |S| – число состояний автомата). Причина этого состоит в том, что в большинстве задач предикаты имеют «локальную природу» по отношению к управляющим состояниям. В каждом состоянии *значимым* является лишь определенный, небольшой поднабор предикатов, остальные же не влияют на значение управляющей функции. Именно это свойство 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 предикатов 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 предикатов ограничивается некоторой константой r.

К таблице, задающей сужение управляющей функции на данное состояние, в этом случае добавляется битовый вектор, описывающий множество значимых предикатов (рис. 6).

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

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

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

Число строк таблицы в этом случае 2^r, однако, константа r обычно невелика. Ее выбор зависит от сложности задачи. Как показывает опыт, для большинства автоматизированных объектов среднее по всем состояниям значение r не больше пяти.

Второй метод – метод представления автоматов деревьями решений предложен в работах [8, 9]. 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 выходов автомата выражаются с помощью деревьев решений. Более формально: зададим

для каждого состояния $q \in Q$ функцию $\sigma_q : X \to Q \times Y$, такую что $\sigma_q(x) = (\delta(q, x), \lambda(q, x))$ для $\forall x \in X$. Здесь Q – множество состояний автомата, X – множество входных воздействий, Y – множество выходных воздействий, $\delta : Q \times X \to Q$ – функция переходов, $\lambda : Q \times X \to Y$ – функция выходов. Каждая из этих функций может быть определена собственным деревом решений. *Таким образом, автомат в целом может быть представлен упорядоченным набором деревьев решений и стартовым состоянием.*

Каждое состояние автомата представляется с помощью соответствующего дерева решений. Деревья решений состоят из узлов, среди которых выделяется корень. Каждый узел представляется следующим образом:

- номер переменной, соответствующей метке узла;
- 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).

Следовательно, при использовании описываемого метода автомат представляется объектом следующего вида на языке *Java*:

```
class TreeAutomata {
    Tree[] trees;
    int startState;
}

class Tree {
    Node root;

    private class Node {
        Node left;
        Node right;
        int transState;
        int action;
    }
}
```

Третий метод – метод совместного применения конечных автоматов и нейронных сетей был предложен в работах [33, 34]. При использовании этого метода для управления системой со сложным поведением предлагается применять нейронную сеть и конечный 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. 7).

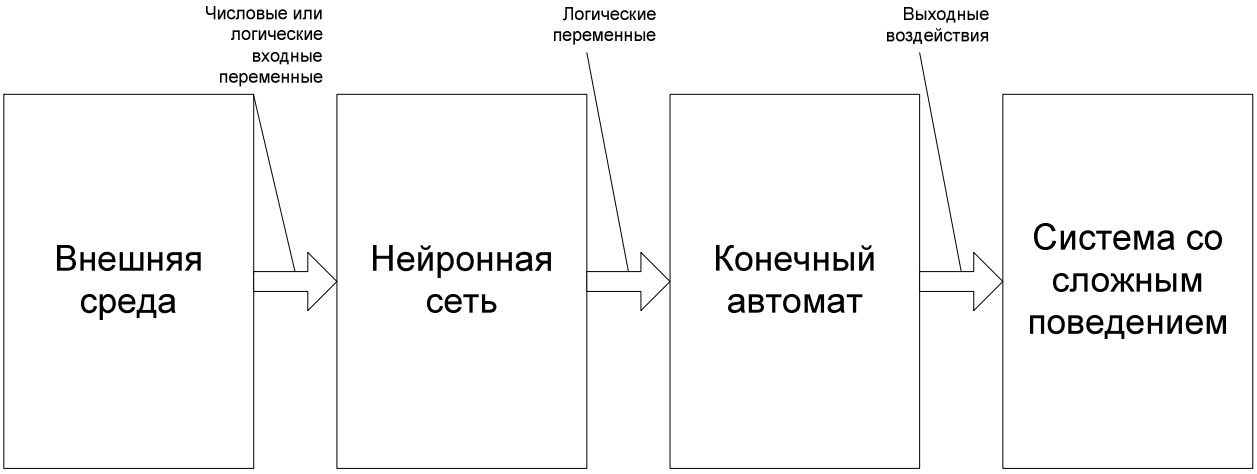

Рис. 7. Структурная схема системы управления при использовании метода совместного применения конечных автоматов и нейронных сетей

Одна из возможных структур нейронной сети и способ ее взаимодействия с конечным автоматом показаны на рис. 8. Отметим, что для решения других задач структура нейронной сети может быть изменена.

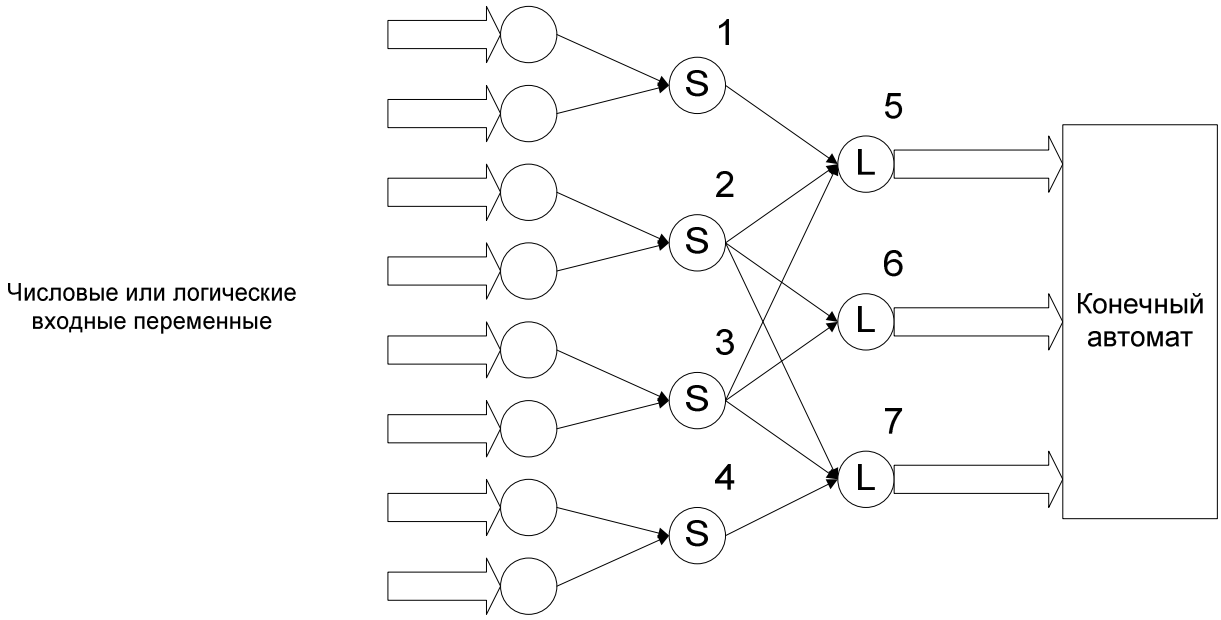

Рис. 8. Возможная структура нейронной сеть и ее взаимодействие с конечным автоматом

Символами *S* на рис. 8 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 конечного автомата.

Экспериментальное исследование указанных трех методов проводилось на построении автомата для задачи «Умный муравей-3».

Приведем сначала описание классической постановки задачи «Умный муравей» [36, 42]. Используется двумерный тор размером 32 на 32 клетки (рис. 9). На некоторых клетках поля расположены яблоки – черные клетки на рис. 9. Яблоки расположены вдоль некоторой ломаной линии, но не на всех ее клетках. Клетки ломаной, на которых яблок нет – серые. Белые клетки – не принадлежат ломаной и не содержат яблок. Всего на поле 89 яблок.

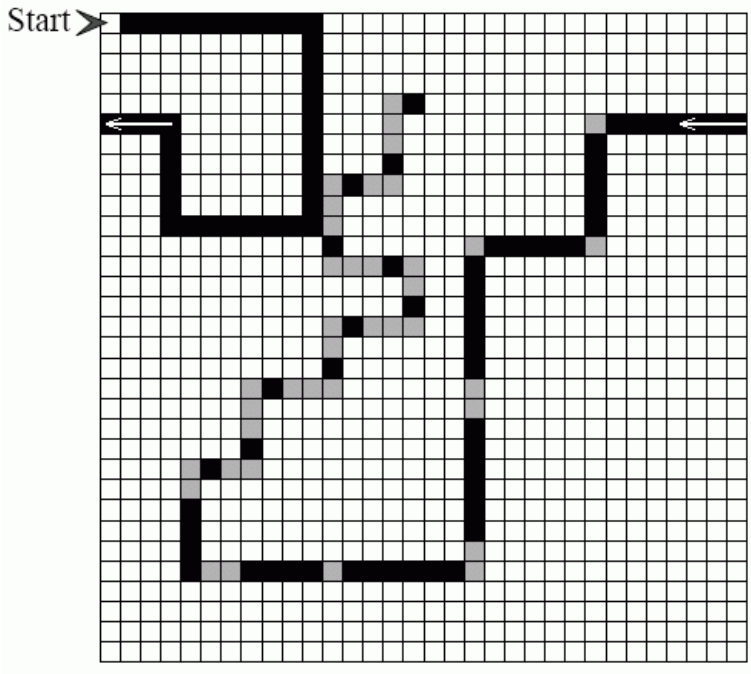

Рис. 9. Поле с яблоками

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

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

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

Цель игры – создать муравья, который не более чем за 200 ходов съест как можно больше яблок. Муравьи, съевшие одинаковое число яблок, заканчивают игру с одинаковым результатом вне зависимости от числа ходов, затраченных каждым из них на процесс еды. Однако эта задача может иметь различные модификации, например, такую, в которой при одинаковом числе съеденных яблок, лучшим считается муравей, съевший яблоки за меньшее число ходов. Ниже будет показано, что

поведение муравья может быть задано конечным автоматом. При этом может быть поставлена задача о построении автомата с минимальным числом состояний для муравья, съедающего все яблоки, или автомата для муравья, съедающего максимальное число яблок при заданном числе состояний.

Постановка задачи «Умный муравей-3», предложенной в работе [2], содержит несколько существенных отличий.

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. 10 изображена область обзора муравья (клетка, в которой находится муравей, обозначена серым цветом).

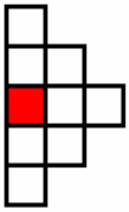

Рис. 10. Область видимости муравья

Во-вторых, расположение еды 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 еды – битовых матриц 32 x 32. Каждому исходу ω_i, содержащему k единиц, поставим в соответствие вероятность $p(\omega_i) = \mu^k(1-\mu)^{n-k}$, где $n = 32 \times 32$.

Для вычисления этой величины в общем случае необходимо перебрать все возможные битовые матрицы размером 32 x 32. Поэтому для оценки эффективности автомата, задающего поведение муравья, вместо точного вычисления этого математического ожидания, оно будет вычислять приближенно – с помощью моделирования поведения муравья на 10000 случайно сгенерированных полях.

Экспериментальные исследования, проведенные в работе [27], показали, что при построении автомата для задачи «Умный муравей-3» указанные три метода представления автоматов показали примерно одинаковые результаты – максимальное достигнутое значение функции приспособленности составило примерно 27 для автоматов из 4–16 состояний.

1.3. Совместное применение генетических алгоритмов и верификации

В настоящем разделе приводится обзор методов совместного применения генетических алгоритмов и верификации.

1.3.1. Совместное применение генетических алгоритмов и верификации для построения взаимодействующих конечных автоматов

Работа [43] посвящена построению взаимодействующих конечных автоматов с использованием генетического программирования с функцией приспособленности, при вычислении которой используется верификация моделей [13]. 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 взамодействующих конечных автоматов. Это означает, что конечные автоматы, входящие в систему, имеют общие глобальные переменные и каналы связи, по которым могут передаваться сообщения.

В эволюционном алгоритме каждый конечный автомат представляется в виде графов переходов, каждый из которых представляет собой набор вершин и дуг. Вершины соответствуют состояниям, а дуги – переходам.

В качестве входных данных для вычисления функции приспособленности выступает описание системы конечных автоматов, а также набор темпоральных свойств, которые требуется проверить. Результатом вычисления функции приспособленности является число верных для системы конечных автоматов свойств. При этом рекомендуется включать в систему тестов помимо тех темпороальных свойств, которые требуется проверить, их ослабленные версии и подформулы, так как это позволяет ускорить процесс построения системы конечных автоматов.

В качестве эволюционного алгоритма в рассматриваемой работе применяется так называемая $(1+\lambda)$-эволюционная стратегия. В процессе работы этого алгоритма поддерживается текущее решение задачи, на каждом шаге с помощью операций мутации из него строятся λ (в рассматриваемой работе $\lambda = 20$) новых кандидатов. После этого из имеющихся $\lambda+1$ систем автоматов выбирается лучшая, и она переходит в следующее поколение. 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;
- с вероятностью 0.2 изменяется пометка перехода.

Отметим, что перечисленные операции реализуют так называемую накапливающую эволюционную стратегию (*accumulative evolution strategy*), которая состоит в том, что процесс эволюции начинается с простых структур, которые затем усложняются за счет добавления новых элементов. Этот подход противоположен подходу, в котором сначала происходит эволюция автоматов с достаточно большим числом состояний.

Также отметим, что в рассматриваемой работе не используется операции скрещивания. В рассматриваемой работе это объясняет тем, что нет четкого понимания того, как в автомате выделить функциональные блоки.

Экспериментальное исследование разработанного алгоритма проводилось на задаче построения автомата управления кофе-машиной. Использовалось два тестовых набора. Первый из них содержал восемь темпоральных свойств, а второй – 17. Второй набор описывал кофе-машину, обладающую большей функциональностью, чем описываемую первым набором. Задача состояла в нахождении системы из двух автоматов, один из которых описывал поведение пользователя, а второй – поведение кофе-машины.

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

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

1.3.2. Совместное применение генетических алгоритмов и верификации для построения алгоритмов взаимного исключения

В работе [44] рассматривается задача автоматизированного построения алгоритмов взаимного исключения. Алгоритмы взаимного исключения играют важную роль в разработке распределенных программ. Цель таких алгоритмов – предовратить одновременный доступ нескольких процессов к общему ресурсу.

Алгоритм взаимного исключения в рассматриваемой работе представляется в виде дерева программы, в которой разрешается использовать операторы `if`, `while` и оператор присваивания. При этом разрешается обращаться к трем общим битовым переменным `A[0]`, `A[1]` и `A[2]`.

В рассматриваемой работе не используется операция скрещивания, а применяется только операция мутации. Для ее выполнения выбирается одна из следующих операций:

- замена случайно выбранного поддерева на случайно сгенерированное;
- удаление случайно выбранного поддерева;
- добавление родительского узла для случайно выбранного узла;
- удаление родительского узла для случайно выбранного узла.

В качестве темпоральной логики для записи свойств алгоритмов взаимного исключения используется *EmCTL** [53]. В этой темпоральной логике возможно получение более детальных ответов от верификатора, нежели «свойство выполняется» или «свойство не выполняется». Таким образом, эта темпоральная логика лучше применима для вычисления функции приспособленности в генетических алгоритмах.

Исходными данными для построения алгоритмов взаимного исключения были существующие алгоритмы взаимного исключения, которые не удовлетворяли некоторым требованиям. В результате проведенных вычислительных экспериментов были построены несколько новых алгоритмов взаимного исключения, удовлетворяющие всем требованиям.

1.4. Сравнение рассмотренных алгоритмов машинного обучения

В табл. 1 приведено сравнение рассмотренных в обзоре алгоритмов машинного обучения для построения конечных автоматов. Сравнение алгоритмов проводится по следующим параметрам:

- тип автоматов, которые строятся (распознаватель, преобразователь, управляющий автомат);
- задача, на которой проводится экспериментальное исследование;
- метод представления конечного автомата;
- тип эволюционного алгоритма;
- особенности предлагаемого эволюционного алгоритма;
- тип алгоритма машинного обучения.

Таблица 1. Сравнение рассмотренных в обзоре алгоритмов машинного обучения, основанных на эволюционных вычислениях, для построения конечных автоматов

Название работы	Рассматриваемая модель	Метод представления используемой модели	Тип эволюционного алгоритма	Особенности эволюционного алгоритма	Тип алгоритма машинного обучения
Применение генетических алгоритмов для генерации автоматов при построении модели максимального правдоподобия и в задачах управления (2008)	Скрытая Марковская модель, конечные автоматы	Таблица значений функции переходов	Генетический алгоритм	Генетический алгоритм используется для построения начального приближения, окончательная оптимизация параметров производится с помощью алгоритма Баума-Велча	Обучение на примерах
Learning DFA: Evolution versus Evidence Drive State Merging (2003)	Конечные автоматы-распознаватели	Таблица значений функции переходов	(1+1)-эволюционная стратегия	Для определения допускающих состояний используется алгоритм расстановки пометок	Обучение на примерах
Learning Deterministic Finite Automata with a Smart State Labeling Algorithm (2005)	Конечные автоматы-распознаватели	Таблица значений функции переходов	(1+1)-эволюционная стратегия	Для определения допускающих состояний используется алгоритм расстановки пометок, предложен метод выполнения операции мутации, учитывающий поведение автомата на обучающих примерах	Обучение на примерах
A Genetic Algorithm for Finite State Automata Induction with Application to Phonotactics (1998)	Конечные автоматы-распознаватели	Таблица значений функции переходов	Генетический алгоритм	Функция приспособленности учитывает не только долю правильно классифицированных обучающих примеров, но и размер конечного автомата	Обучение на примерах

Genetic Inference of Finite State Machines (2007)	Конечные автоматы-распознаватели	В виде битовых строк	Генетический алгоритм	Анализируется не тип состояния, в которое пришел автомат после обработки строки, а последовательность типов состояний, которые были посещены в процессе обработки строки. Используется два типа функции приспособленности.	Обучение на примерах
Evolving Finite-State Transducers: Some Initial Explorations (2003)	Конечные автоматы-преобразователи	Таблица значений функции переходов и таблица значений функции выходов	Эволюционная стратегия (1+1)	В работе рассматриваются три варианта функции приспособленности	Обучение на примерах
Evolving Finite-State Machine Strategies for Protecting Resources (2000)	Управляющие конечные автоматы	Таблица значений функции переходов и таблица значений функции выходов	Генетический алгоритм	Для предотвращения зацикливания поведения автомата применялась динамическая верификация	Обучение на собственных ошибках
Применение генетического программирования для генерации автоматов с большим числом входных переменных (2008)	Управляющие конечные автоматы	Функции переходов и действий в виде сокращенных таблицы	Генетический алгоритм	Сокращенные таблицы позволяют учитывать в каждом состоянии только так называемые «значимые предикаты»	Обучение на собственных ошибках
Метод представления автоматов деревьями решений для использования в генетическом программировании (2008)	Управляющие конечные автоматы	Функции переходов и действий в виде деревьев решений	Генетический алгоритм	Деревья решений позволяют задавать в каждом состоянии свой приоритет переменных	Обучение на собственных ошибках

Совместное применение генетического программирования, конечных автоматов и искусственных нейронных сетей для построения системы управления беспилотным летательным аппаратом (2008)	Управляющие конечные автоматы	Таблица значений функции переходов и таблица значений функции выходов, нейронная сеть для обработки входных переменных	Генетический алгоритм	Нейронные сети выполняют преобразование входных вещественных переменных в логические	Обучение на собственных ошибках
Genetic Programming with Fitness based on Model Checking (2007)	Система двух взаимодействующих конечных автоматов	Графы с пометками на вершинных и дугах (графы переходов)	$(1+\lambda)$-эволюционная стратегия	Для вычисления функции приспособленности используется верификация моделей на основе темпоральной логики *CTL*	Обучение на основе спецификации
Genetic Programming and Model Checking: Synthesizing New Mutual Exclusion Algorithms (2008)	Алгоритмы взаимного исключения	Дерево программы	Генетический алгоритм, не используется операция скрещивания	Для вычисления функции приспособленности используется верификация моделей на основе темпоральной логики *EmCTL*	Обучение на основе спецификации

Выводы по главе 1

1. Алгоритмы машинного обучения настраивают параметры модели либо на заранее подготовленных тестовых примерах, либо на собственных ошибках. При этом достаточно распространены модели, содеражащие состояния. Они могут быть как вероятностными (скрытые Марковские модели), так и детерминированными (конечные автоматы).
2. Конечные автоматы могут использоваться в задачах распознаваения языков, в задачах формирования последовательностей, а также в задачах управления.
3. В большинстве работ по разработке алгоритмов машинного обучения конечных автоматов рассматриваются конечные автоматы-распознаватели. Эксперименты, проведенные в рамках работ по построению других типов конечных автоматов, показывают, что имеется надежда на то, что алгоритмы машинного обучения, основанные на эволюционных вычислениях, будут эффективны и для решения задач построения автоматов других типов.
4. Как правило, в эволюционных алгоритмах число состояний всех автоматов, рассматриваемых в процессе работы алгоритма, одинаково (при этом, однако часть состояний в каждом из автоматов может быть недостижима).
5. Методы верификации моделей могут применяться совместно с эволюционными вычислениями для построения конечных автоматов с гарантированным поведением. Кроме этого, совместное применение верификации и генетического программирования успешно применяется в задачах построения других типов программ (например, алгоритмов взаимного исключения). Метод машинного обучения, использующий верификацию модели, можно назвать «обучением на основе спецификации».

2. ВЫБОР И ОБОСНОВАНИЕ ОПТИМАЛЬНОГО ВАРИАНТА НАПРАВЛЕНИЯ ИССЛЕДОВАНИЙ

На основании результатов выполненного аналитического обзора предлагается следующий вариант направления исследований: будут разработаны новые основанные на генетических алгоритмах методы машинного обучения для построения управляющих конечных автоматов. На первом этапе такие методы будут разработаны для построения автоматов Мили, а на втором этапе – для автоматов Мура и смешанных автоматов. При этом методы машинного обучения будут основаны на концепции обучения на тестовых примерах, а для учета знаний человека будет разработан метод машинного обучения на основе генетических алгоритмов с использованием верификации темпоральных свойств при вычислении функции приспособленности. Кроме этого, будут выполнены экспериментальные исследования указанных методов на задаче построения автомата управления системой со сложным поведением.

Выполненный аналитический обзор показывает, что модели, основанные на состояниях, широко применяются в машинном обучении. Примером таких моделей являются Марковские цепи и скрытые Марковские модели. Они могут применяться в таких задачах, как распознавание речи, и для них разработан ряд алгоритмов обучения. Область их применения ограничивается тем, что эти модели имеют вероятностный характер. Наиболее близкими к ним моделями, имеющими детерминированный характер, являются конечные автоматы.

Как правило, в машинном обучении конечные автоматы рассматриваются как распознаватели регулярных языков или как преобразователи слов одного регулярного языка в слова другого. В случае построения управляющих автоматов обычно рассматриваются системы с малым числом входных переменных (одной или двумя), что существенно ограничивает область применения таких алгоритмов. Кроме этого, существующие алгоритмы машинного обучения для конечных автоматов основаны на принципе обучения «на собственных ошибках» и не являются достаточно эффективными для ряда задач, так как не позволяют эффективно учитывать знания человека, и не позволяют строить автоматы с гарантированным поведением.

Методы, которые разрабатываются в рамках работ по настоящему Государственному контракту, будут основаны на принципах «обучения на примерах» и лишены указанных недостатков.

Выводы по главе 2

1. Выбрано и обосновано оптимальное направление проведения исследований.

3. ПЛАН ПРОВЕДЕНИЯ ТЕОРЕТИЧЕСКИХ И ЭКСПЕРИМЕНТАЛЬНЫХ ИССЛЕДОВАНИЙ

В настоящем разделе представлен план проведения теоретических и экспериментальных исследований.

3.1. План проведения первого этапа теоретических и экспериментальных исследований

На первом этапе проведения теоретических и экспериментальных исследований планируется провести следующие работы:

- разработка, программная реализация и экспериментальное исследование метода машинного обучения на основе генетических алгоритмов для построения управляющих конечных автоматов Мили на основе тестовых примеров (09.11.2009 – 22.11.2009):
  - разработка метода представления управляющего конечного автомата в виде хромосомы генетического алгоритма;
  - разработка алгоритма скрещивания хромосом, учитывающего тестовые примеры;
  - разработка метода вычисления функции приспособленности, учитывающего поведение автомата на тестовых примерах;
  - апробация метода на примере построения управляющего автомата для системы со сложным поведением.

Результатами работ первого этапа являются:

- метод машинного обучения на основе генетических алгоритмов для построения управляющих конечных автоматов Мили на основе тестовых примеров;
- научно-технический отчет, содержащий аналитический обзор и описание указанного метода.

3.2. План проведения второго этапа теоретических и экспериментальных исследований

На втором этапе проведения теоретических и экспериментальных исследований планируется провести следующие работы:

- разработка, программная реализация и экспериментальное исследование метода машинного обучения на основе генетических алгоритмов для построения управляющих автоматов Мура и смешанных автоматов (Мили-Мура) на основе тестовых примеров (01.01.2010 – 28.02.2010):
  - разработка метода представления управляющего конечного автомата в виде хромосомы генетического алгоритма;
  - разработка алгоритма скрещивания хромосом;
  - разработка метода вычисления функции приспособленности, учитывающего поведение автомата на тестовых примерах.
  - апробация метода на примере построения управляющего автомата для системы со сложным поведением;
- разработка, программная реализация и экспериментальное исследование метода машинного обучения на основе генетических алгоритмов с использованием верификации темпоральных свойств при вычислении функции приспособленности (01.03.2010 – 30.04.2010):

- разработка метода представления управляющего конечного автомата в виде хромосомы генетического алгоритма;
- разработка алгоритма скрещивания хромосом;
- разработка метода вычисления функции приспособленности, учитывающего результаты верификации темпоральных свойств;
- апробация метода на примере построения управляющего автомата для системы со сложным поведением;

- обобщение и оценка результатов исследований (01.05.2010 – 15.07.2010):
  - оценка полноты решения задачи и достижения поставленных целей;
  - сопоставление и обобщение результатов анализа научно-информационных источников и теоретических и экспериментальных исследований;
  - оценка эффективности полученных результатов в сравнении с современным научно-техническим уровнем;
  - разработка рекомендаций по использованию результатов НИР при создании научно-образовательных курсов.

Результатами работ второго являются:

- метод машинного обучения на основе генетических алгоритмов для построения управляющих автоматов Мура и смешанных автоматов (Мили-Мура) на основе тестовых примеров;
- метод машинного обучения на основе генетических алгоритмов с использованием верификации темпоральных свойств при вычислении функции приспособленности.
- одна статья в журналах из перечня ВАК;
- свидетельство о регистрации программы для ЭВМ;
- протоколы экспериментов;
- рекомендации по использованию результатов НИР при создании научно-образовательных курсов;
- научно-технический отчет.

Выводы по главе 3

1. Разработан план проведения теоретических и экспериментальных исследований.
2. На первом этапе будет выполнена разработка, программная реализация и экспериментальное исследование метода машинного обучения на основе генетических алгоритмов для построения управляющих конечных автоматов Мили на основе тестовых примеров.

4. РАЗРАБОТКА, ПРОГРАММНАЯ РЕАЛИЗАЦИЯ И ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ МЕТОДА МАШИННОГО ОБУЧЕНИЯ НА ОСНОВЕ ГЕНЕТИЧЕСКИХ АЛГОРИТМОВ ДЛЯ ПОСТРОЕНИЯ УПРАВЛЯЮЩИХ КОНЕЧНЫХ АВТОМАТОВ МИЛИ НА ОСНОВЕ ТЕСТОВЫХ ПРИМЕРОВ

В настоящем разделе приводится описание метода машинного обучения на основе генетических алгоритмов для построения управляющих конечных автоматов Мили на основе тестовых примеров.

4.1. Постановка задачи

При применении парадигмы автоматного программирования для реализации сущности со сложным поведением выделяется система управления и объект управления. На начальном этапе проектирования программы выделяются события (*e1*, *e2*, …), входные переменные (*x1*, *x2*, …) и выходные воздействия (*z1*, *z2*, …). После этого проектирование программы может идти разными путями. Один из них состоит в написании сценария работы программы, по которому далее эвристически строится автомат. Пример построения автомата таким способом приведен в работе [15].

Другой подход, который практически не применяется для построения автоматных программ, но достаточно широко распространен при разработке традиционных программ, состоит в разработке на основе тестов (*test-driven development*) [4]. При применении этого метода процесс написания кода на языке программирования идет параллельно с написанием тестов для программы. При этом добавление функциональности в программу выполняется только после того, как создан тест для проверки этой функциональности. Таким образом, функциональность программы описывается набором тестов для нее.

При применении автоматного программирования в качестве тестов для управляющего конечного автоматов естественно рассматривать пары последовательностей, одна из которых описывает события и входные переменные, поступающие на вход автомату, а вторая – выходные воздействия, которые должен вырабатывать автомат при обработке этих событий. Таким образом, задача построения управляющего конечного автомата становится похожей на задачу построения конечного преобразователя (разд. 1.2.1.2), для решения которой успешно применяются генетические алгоритмы. Кроме этого, как отмечалось выше, построение конечного автомата управления системой со сложным поведением вручную является достаточно трудоемкой задачей. Поэтому естественно возникает идея об автоматизации этого процесса с использованием генетических алгоритмов.

Далее в настоящей главе описан метод построения с помощью генетического программирования автоматов управления системой со сложным поведением на основе тестов.

4.2. Описание предлагаемого метода

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

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

Отметим, что для таких тестов справедливо свойство, которое можно сформулировать следующим образом – «префиксы тестов являются тестами». При этом если из входной

последовательности событий удалить часть событий, находящихся в ее конце, то результат обработки автоматом этой последовательности будет префиксом исходной выходной последовательности. Поэтому естественно в набор тестов включать все префиксы теста.

4.2.1. Представление конечного автомата в виде хромосомы генетического алгоритма

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

Таким образом, в особи кодируется только «скелет» управляющего конечного автомата, а конкретные выходные воздействия, вырабатываемые на переходах, определяются с помощью алгоритма расстановки пометок, который аналогичен предложенному в работе [47].

Выбор представления графа переходов автомата с помощью списков ребер (в отличие от работы [49], 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.

4.2.1.1. Обработка входных переменных

Отметим, что в общем случае функция переходов и функция действий зависят не только от события, поступившего на вход автомату, но и от значений входных переменных, которые, как правило, имеют логический тип. При построении автомата вручную, например с использованием инструментального средства *UniMod* [7], переходы обычно помечаются не только событиями, но и так называемыми «охранными условиями» (*guard conditions*) – логическими формулами, которые задают ограничения на значения входных переменных.

4.2.1.2. Алгоритм расстановки пометок

Идея алгоритма расстановки пометок состоит в том, что с помощью генетического алгоритмома строится только «скелет» конечного автомата, а пометки на переходах (вырабатываемые на них выходные воздействия) расставляются на основе тестов. При этом расстановка пометок происходит таким образом, чтобы получившийся в результате автомат как можно лучше «соответствовал» тестам.

В работе [49] аналогичный принцип использовался при построении конечных автоматов для распознавания регулярных языков. Главное отличие состоит в том, что в этой работе расставлялись пометки на состояниях – на основе тестов определялось, будет состояние допускающим или нет.

Опишем алгоритм расстановки пометок на переходах, применяемый в настоящей работе. Как было сказано выше, для каждого из перехода в особи генетического алгоритма записано, сколько выходных воздействий должно вырабатываться при его выборе. Подадим на вход конечному автомату последовательность событий, соответствующую одному из тестов, и будем наблюдать, за тем, какие переходы выполняет автомат. Зная эти переходы и информацию о том, сколько выходных воздействий должно быть сгенерировано на каждом из переходов, 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 *z5, z5, z4, z3, z5, z5*, 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. 11).

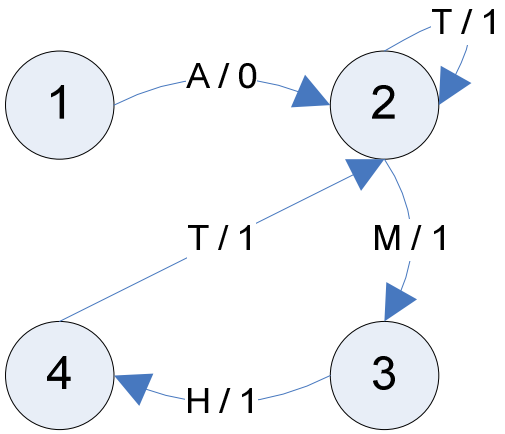

Рис. 11. Некоторые переходы «скелета» автомата

На рис. 11 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 *z5*, на переходе по событию *M* из второго состояния в третье должно вырабатываться выходное воздействие *z4*, на переходе по событию *H* из третьего состояния в четвертое должно вырабатываться *z5*, а на переходе по событию *T* из четвертого состояния во второе – *z3* (рис. 12).

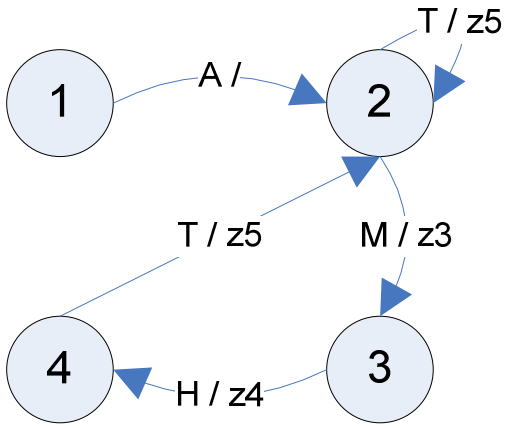

Рис. 12. Помеченные на основании теста переходы автомата

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 – дело обстоит иначе. Во-первых, понятно, что рассматриваемый «скелет» автомата не позволяет пройти все тесты. С другой стороны, на этом переходе должны выполняться некоторые выходные воздействия. Поэтому предлагается пометить этот переход 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, образующую последовательность *zs*. После этого каждый переход помечается той последовательностью zs_0, для которой величина $C[T][zs_0]$ максимальна.

4.2.1.3. Операция мутации

При выполнении операции мутации с заданной вероятностью (по умолчанию, она равна 0.05) выполняется каждое из действий:

- изменение начального состояния;
- изменение описания каждого из переходов;
- удаление или добавление перехода для каждого из состояний.

При изменении начального состояния оно заменяется на выбранное случайным образом. При изменении описания перехода с равной вероятностью выполняется одно из следующих действий:

- изменение состояния, в которое ведет переход, – оно изменяется на случайно выбранное;
- изменение события, по которому выполняется этот переход, – оно изменяется на случайно выбранное;
- изменение числа выходных воздействий, вырабатываемых на этом переходе, – с равной вероятностью либо уменьшается на единицу, либо увеличивается на единицу, но при этом не может стать отрицательным или превзойти некоторое заданное ограничение.

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

4.2.1.4. Операция удаления дублирующихся переходов

Для удаления дублирующихся переходов для каждого состояния проделывается следующие операции: последовательно просматривается список переходов из этого состояния, и запоминаются события, переходы по которым определены для этого состояния. Если очередной переход *T* происходит по событию, для которого в списке уже есть переход, то этот переход удаляется из списка.

4.2.1.5. Операция скрещивания

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

- *S1.is = P1.is* и *S2.is = P2.is*;
- *S1.is = P2.is* и *S2.is = P1.is*.

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

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

1. Строится общий список переходов, в который помещаются переходы, входящие как в *P1.T[i]*, так и в *P2.T[i]*.
2. К полученному списку применяется случайная перестановка.
3. Далее возможны два равновероятных варианта:
   - либо в *S1.T[i]* помещаются первые |*P1.T[i]*| переходов из полученного списка, а в *S2.T[i]* – оставшиеся переходы;
   - либо в *S1.T[i]* помещаются первые |*P2.T[i]*| переходов из полученного списка, а в *S2.T[i]* – оставшиеся переходы.

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

1. В автоматах *P1* и *P2* помечаются те переходы, которые выполняются при обработке 10% тестов, для которых нормированное редакционное расстояние между «правильным ответом» *Answer* и последовательностью *Output* выходных воздействий, генерируемой автоматом, $\frac{\mathrm{ED}(\mathrm{Output}[i],\mathrm{Answer}[i])}{\max(|\mathrm{Output}[i]|,|\mathrm{Answer}[i]|)}$ минимально.
2. *Помеченные* переходы копируются в *S1.T[i]* и *S2.T[i]* напрямую.
3. Строится общий список переходов, в который помещаются *непомеченные* переходы, входящие как в *P1.T[i]*, так и в *P2.T[i]*.
4. К полученному списку *L* применяется случайная перестановка.
5. Список *S1.T[i]* дополняется первыми переходами из списка *L* до размера |*P1.T[i]*|, а список *S2.T[i]* дополняется оставшимися переходами.

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

4.2.1.6. 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.

4.2.2. Генетический алгоритм

В качестве основной стратегии формирования следующего поколения используется элитизм [6]. 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.

4.2.3. Вычисление функции 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 вычисляется величина $\mathrm{FF}_1 = \frac{\sum_{i=1}^{n}(1 - \frac{\mathrm{ED}(\mathrm{Output}[i], \mathrm{Answer}[i])}{\max(|\mathrm{Output}[i]|, |\mathrm{Answer}[i]|)})}{n}$, где как *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 тестам, тем больше значение функции приспособленности.

Функция приспособленности зависит не только от того, насколько «хорошо» автомат работает на тестах, но и числа переходов, которые он содержит. Она вычисляется по формуле:

$$\mathrm{FF}_2 = \begin{cases} 10 \cdot \mathrm{FF}_1 + 0.01 \cdot (100 - \mathrm{cnt}), \mathrm{FF}_1 < 1 \\ 20 + 0.01 \cdot (100 - \mathrm{cnt}), \mathrm{FF}_1 = 1 \end{cases},$$

где как *cnt* – число переходов в автомате.

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

Учет числа переходов в функции приспособленности необходим по двум причинам. Во-первых, минимизация числа переходов приводит к тому, что в результирующем автомате отсутствуют неиспользуемые в тестах переходы (так как они не используются, то могут быть удалены из автомата без ущерба для его поведения на тестах). Во-вторых, чем меньше в автомате переходов, тем более «общее» поведение он задает. Таким образом, частично решается проблема «переобучения», заключающаяся в том, что автомат демонстрирует правильное поведение только на тестовых входных последовательностях. Две указанные особенности должны учитываться при построении набора обучающих тестов.

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

4.3. Программная реализация

Программная реализация разработанного метода машинного обучения выполнялась на языке программирования *Java*. Программа состоит из шести классов (рис. 13).

Рис. 13. Структура программы

Класс *Starter* загружает из файла описание тестовых примеров и запускает генетический алгоритм, который реализован в классе `GeneticAlgorithm`. 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`, реализующий конечный автомат Мили, хранит номер начального состояния, число состояний, набор переходов для каждого состояния, набор входных событий и набор выходных воздействий:

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

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

В этом классе также реализованы операции мутации, скрещивания и алгоритм расстановки пометок. Приведем фрагмент программы, реализующей операцию мутации:

```
public MealyFSM mutate() {
     int newInitialState;
     if (RANDOM.nextDouble() < MUTATION_THRESHOLD) {
          newInitialState = RANDOM.nextInt(stateNumber);
     } else {
```

```
            newInitialState = initialState;
        }
        Transition[][] newStates = new Transition[stateNumber][];
        for (int i = 0; i < stateNumber; i++) {
            newStates[i] = new Transition[states[i].length];
            for (int j = 0; j < states[i].length; j++) {
                newStates[i][j] = states[i][j].copy();
            }
        }
        for (int i = 0; i < stateNumber; i++) {
            for (int j = 0; j < newStates[i].length; j++) {
                if (RANDOM.nextDouble() < MUTATION_THRESHOLD) {
                    newStates[i][j] = newStates[i][j].mutate(setOfInputs,
setOfOutputs, stateNumber);
                }
            }
        }
        for (int i = 0; i < stateNumber; i++) {
            if (RANDOM.nextDouble() < MUTATION_THRESHOLD) {
                if (RANDOM.nextBoolean()) {
                    // Удаление перехода
                    if (newStates[i].length > 0) {
                        newStates[i] = deleteTransition(newStates[i]);
                    }
                } else {
                    // Добавление перехода
                    if (newStates[i].length < setOfInputs.length) {
                        newStates[i] = addTransition(newStates[i]);
                    }
        }
            }
        }
        for (int i = 0; i < stateNumber; i++) {
            newStates[i] = removeDuplicates(newStates[i]);
        }
        return new MealyFSM(newStates, newInitialState, setOfInputs,
setOfOutputs);
}
```

Класс `Transition` хранит событие, по которому осуществляется данный переход, число выходных воздействий, вырабатываемых на нем, а также номер состояния, в которое ведет этот переход. Кроме этого, он содержит массив для записи выходных воздействий после применения алгоритма расстановки пометок:

```
final String input;
private final int outputSize;
private String[] output;

private final int newState;
```

Класс `GeneticAlgorithm` реализует описанный выше генетический алгоритм. Класс `FitnessCalculator` реализует вычисление функции приспособленности на основе редакционного расстояния:

```
public double calcFitnessForTest(MealyFSM fst, Test test) {
     String[] output = fst.transform(test.getInput());
     String[] answer = test.getOutput();
     double t;
     if (output == null) {
          t = 1;
     } else {
          t = editDistance(output, answer) / Math.max(output.length, an-
swer.length);
     }
     return 1 - t;
}

public double calcFitness(MealyFSM fst) {
     double sum = 0;
     fst.doLabelling(tests);
     int cntOk = 0;
     for (Test p : tests) {
          String[] output = fst.transform(p.getInput());
          String[] answer = p.getOutput();
          double t;
          if (output == null) {
               t = 1;
          } else {
               t = editDistance(output, answer) / Math.max(output.length,
answer.length);
          }
          sum += 1 - t;
          if (same(output, answer)) {
               cntOk++;
          }
     }
     if (cntOk == tests.size()) {
          return 20 + 0.01 * (100 - fst.getTransitionsCount());
     } else {
          return 10 * (sum / tests.size()) + 0.01 * (100 -
fst.getTransitionsCount());
     }
}
```

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

```
private final ArrayList<String> input;
private final ArrayList<String> output;
```

4.4. Экспериментальное исследование

В настоящем разделе описывается применение предлагаемого метода для построения конечного автомата управления часами с будильником [20]. Эти часы имеют три кнопки (рис. 14), которые предназначены для изменения режима их работы и для настройки текущего времени или времени срабатывания будильника.

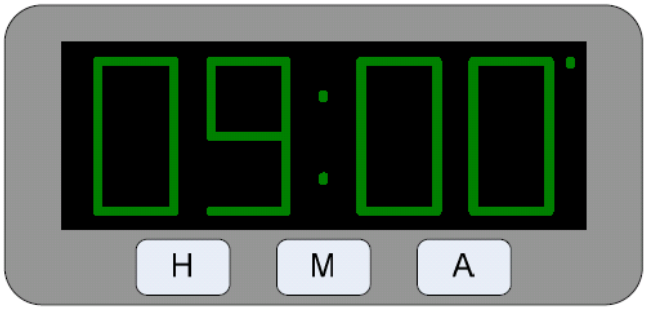

Рис. 14. Внешний вид часов с будильником

Если будильник выключен, то кнопки «H» и «M» служат, соответственно, для увеличения на единицу числа часов и минут в текущем времени. Кнопка «A» в этом режиме предназначена для перехода в режим настройки времени срабатывания будильника. В этом режиме кнопки «H» и «M» используются для увеличения на единицу числа 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 с будильником являются системой со сложным поведением, так как в ответ на одни и те же входные события (нажатия кнопок) в зависимости от режима работы генерируются различные выходные воздействия.

Эта система со сложным поведением имеет четыре входных 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:

- *x1* – верно ли, что время срабатывания будильника совпадает с текущим временем;
- *x2* – верно ли, что текущее время превышает время срабатывания будильника ровно на одну минуту.

Кроме этого эта система имеет семь выходных воздействий:

- *z1* – увеличить число часов текущего времени;
- *z2* – увеличить число минут часов текущего времени;
- *z3* – увеличить число часов времени срабатывания будильника;
- *z4* – увеличить число минут времени срабатывания будильника;
- *z5* – прибавить минуту к текущему времени;

- *z6* – включить звонок будильника;
- *z7* – выключить звонок будильника.

Поведение часов с будильником может быть описано графом переходов конечного автомата, который приведен в работе [20] и построен вручную (рис. 15).

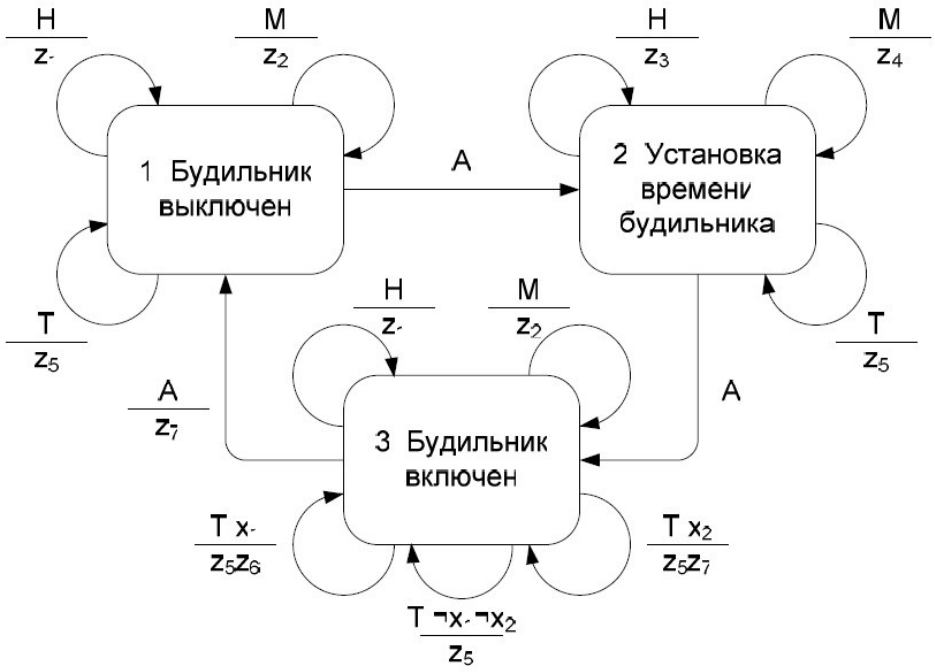

Рис. 15. Граф переходов конечного автомата управления часами с будильником

Начальным состоянием этого автомата является состояние «1. Будильник выключен».

4.4.1. Система тестовых примеров

В систему тестов для построения автомата управления часами с будильником включены тесты, описывающие его работу во всех трех состояниях. Тесты, описывающие поведение часов с будильником в состоянии «Будильник выключен», приведены в табл. 2.

Таблица 2. Тесты для состояния «Будильник выключен»

Тест	Комментарий
Input: *T, T, T, T* Answer: *z5, z5, z5, z5*	Описывает обработку часами события «Сработал таймер». При возникновении этого события текущее время должно быть увеличено на минуту.
Input: *H, H, H, H* Answer: *z1, z1, z1, z1*	Описывает обработку часами нажатия кнопки «H». При нажатии на эту кнопку число часов в текущем времени должно быть увеличено на единицу.
Input: *M, M, M, M* Answer: *z2, z2, z2, z2*	Описывает обработку часами нажатия кнопки «M». При нажатии на эту кнопку число минут в текущем времени должно быть увеличено на единицу.
Input: *T, M, H, T, T, T, M, T, H, H, T, M* Answer: *z5, z2, z1, z5, z5, z5, z2, z5, z1, z1, z5, z2*	Описывает обработку событий H, M и T в состоянии «Будильник выключен».
Input: *A, A, A, T, T, T, T* Answer: *z7, z5, z5, z5, z5*	После трех нажатий кнопки «A» часы должны находиться в состоянии «Будильник выключен». Аналог первого теста.
Input: *A, A, A, H, H, H, H* Answer: *z7, z1, z1, z1, z1*	После трех нажатий кнопки «A» часы должны находиться в состоянии «Будильник выключен». Аналог второго теста.
Input: *A, A, A, M, M, M, M* Answer: *z7, z2, z2, z2, z2*	После трех нажатий кнопки «A» часы должны находиться в режиме «Будильник выключен». Аналог третьего теста.

Тесты, описывающие поведение часов с будильником в состоянии «Установка времени будильника», приведены в табл. 3.

Таблица 3. Тесты для состояния «Установка времени будильника»

Тест	Комментарий
Input: *A, T, T, T, T* Answer: *z5, z5, z5, z5*	Описывает обработку часами события «Сработал таймер». При возникновении этого события текущее время должно быть увеличено на минуту.
Input: *A, H, H, H, H* Answer: *z3, z3, z3, z3*	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: *z4, z4, z4, z4*	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: *z5, z4, z3, z5, z5, z5, z4, z5, z3, z3, z5, z4*	Описывает обработку событий H, M и T в состоянии «Установка времени будильника».
Input: *A, A, A, A, T* Answer: *z7, z5*	После четырех нажатий кнопки «A» часы должны находиться в состоянии «Установка времени будильника». Описывается обработка срабатывания таймера.
Input: *A, A, A, A, T, T, T, T* Answer: *z7, z5, z5, z5, z5*	После четырех нажатий кнопки «A» часы должны находиться в состоянии «Установка времени будильника». Описывается обработка нескольких срабатываний таймера.
Input: *A, A, A, A, H* Answer: *z7, z3*	После четырех нажатий кнопки «A» часы должны находиться в состоянии «Установка времени будильника». Описывается обработка нажатия кнопки «H».
Input: *A, A, A, A, H, H, H, H* Answer: *z7, z3, z3, z3, z3*	После четырех нажатий кнопки «A» часы должны находиться в состоянии «Установка времени будильника». Описывается обработка нескольких нажатий кнопки «H».
Input: *A, A, A, A, M* Answer: *z7, z4*	После четырех нажатий кнопки «A» часы должны находиться в состоянии «Установка времени будильника». Описывается обработка нажатия кнопки «M».
Input: *A, A, A, A, M, M, M, M* Answer: *z7, z4, z4, z4, z4*	После четырех нажатий кнопки «A» часы должны находиться в состоянии «Установка времени 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: *z1, z1, z1, z1*	Описывает обработку часами нажатия кнопки «H». При нажатии на эту кнопку число часов в текущем времени должно быть увеличено на единицу.
Input: *A, A, M, M, M, M* Answer: *z2, z2, z2, z2*	Описывает обработку часами нажатия кнопки «M». При нажатии на эту кнопку число минут в текущем времени должно быть увеличено на единицу.
Input: *A, A, T [!x1 & !x2]* Answer: *z5*	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 [!x1 & !x2], T [!x1 & !x2], T [!x1 & !x2]* Answer: *z5, z5, z5*	Расширение предыдущего теста.
Input: *A, A, T [!x1 & !x2], T [x1], T [x2]* Answer: *z5, z5, z6, z5, z7*	Описывает обработку события «Сработал таймер» при различных значениях входных переменных.
Input: *A, A, T [!x1 & !x2], T [x1]* Answer: *z5, z5, z6*	Префикс предыдущего теста.
Input: *A, A, T [!x1 & !x2], T [x1], T [x2], H* Answer: *z5, z5, z6, z5, z7, z1*	Описывает обработку события «Сработал таймер» при различных значениях входных переменных, а также обработку нажатие кнопки «H».
Input: *A, A, T [!x1 & !x2], T [x1], T [x2], M* Answer: *z5, z5, z6, z5, z7, z2*	Описывает обработку события «Сработал таймер» при различных значениях входных переменных, а также обработку нажатие кнопки «M».
Input: *A, A, T [!x1 & !x2], T [x1], T [x2], T [!x1 & !x2]* Answer: *z5, z5, z6, z5, z7, z5*	Описывает обработку события «Сработал таймер» при различных значениях входных переменных.
Input: *A, A, T [x1]* Answer: *z5, z6*	Описывает обработку события «Сработал таймер» при условии, что текущее время совпадает со временем срабатывания будильника.
Input: *A, A, T [x2]* Answer: *z5, z7*	Описывает обработку события «Сработал таймер» при условии, что текущее время превышает на минуту время срабатывания будильника.
Input: *A, A, T [x1], T [x2]* Answer: *z5, z6, z5, z7*	Объединяет два предыдущих теста.
Input: *A, A, T [!x1 & !x2], M, H, T [!x1 & !x2], T [!x1 & !x2], T [!x1 & !x2], M, T [!x1 & !x2], H, H, T [!x1 & !x2], M* Answer: *z5, z2, z1, z5, z5, z5, z2, z5, z1, z1, z5, z2*	Описывает обработку событий H, M и T в режиме «Будильник включен».

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

Таблица 5. Тесты, описывающие поведение автомата в нескольких состояниях

Тест	Комментарий
Input: *A, A, T [!x1 & !x2], T [x1], T [x2], A, H* Answer: *z5, z5, z6, z5, z7, z7, z1*	Описывает поведение часов с будильником в двух состояниях: «Будильник включен» и «Будильник выключен».
Input: *A, A, T [!x1 & !x2], T [x1], T [x2], A, M* Answer: *z5, z5, z6, z5, z7, z7, z2*	Описывает поведение часов с будильником в двух состояниях: «Будильник включен» и «Будильник выключен».
Input: *A, A, T [!x1 & !x2], T [x1], T [x2], A, T* Answer: *z5, z5, z6, z5, z7, z7, z5*	Описывает поведение часов с будильником в двух состояниях: «Будильник включен» и «Будильник выключен».
Input: *A, A, T [!x1 & !x2], T [x1], T [x2], A, A, H* Answer: *z5, z5, z6, z5, z7, z7, z3*	Описывает поведение часов с будильником в двух режимах: «Будильник включен» и «Установка времени будильника».
Input: *A, A, T [!x1 & !x2], T [x1], T [x2], A, A, M* Answer: *z5, z5, z6, z5, z7, z7, z4*	Описывает поведение часов с будильником в двух режимах: «Будильник включен» и «Установка времени будильника».
Input: *A, A, T [!x1 & !x2], T [x1], T [x2], A, A, T* Answer: *z5, z5, z6, z5, z7, z7, z5*	Описывает поведение часов с будильником в двух режимах: «Будильник включен» и «Установка времени будильника».
Input: *A, A, T [!x1 & !x2], T [x1], A, A, T* Answer: *z5, z5, z6, z7, z5*	Описывает поведение часов с будильником в двух режимах: «Будильник включен» и «Установка времени будильника».

Набор тестов должен быть в определенном смысле достаточно полным. Во-первых, если система со сложным поведением, для управления которой строится конечный автомат, имеет несколько состояний, то система тестов должна содержать тесты для каждого из состояний 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 в целом должен содержать все существующие в системе входные события и выходные воздействия.

4.4.2. Результаты применения генетического алгоритма

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

- размер поколения – 2000 особей;
- доля «элиты» – наиболее приспособленных особей, напрямую переходящих в следующее поколение, – 10 %;
- число поколений до малой «мутации поколения» – 100 поколений;
- число поколений до большой «мутации поколения» – 150 поколений;

- размер автоматов в начальном поколении – четыре состояния.

Было проведено 1000 запусков алгоритма с указанными параметрами. Цель каждого из них состояла в том, чтобы построить автомат, содержащий 14 переходов и соответствующий всем тестам (значение функции приспособленности для такого автомата – 20.86).

В результате работы алгоритма генетического программирования на одном из запусков был построен автомат (рис. 16), в котором из начального (отмечено «жирной» рамкой) достижимы только три состояния из четырех. Если удалить недостижимое состояние, то граф переходов будет изоморфен построенному вручную. В результате остальных запусков были построены автоматы, изоморфные указанному и отличающиеся от него только нумерацией состояний.

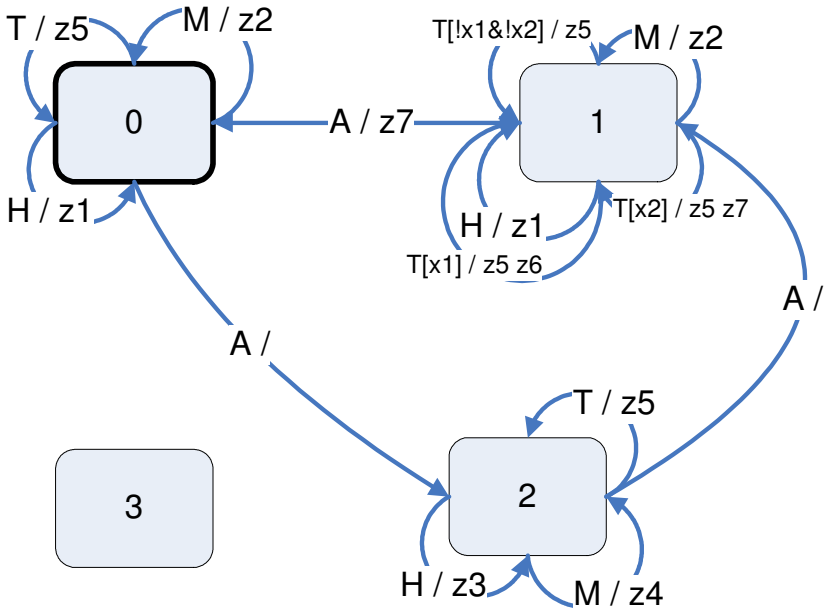

Рис. 16. Граф переходов автомата, построенного с помощью алгоритма генетического программирования

График зависимости максимального значения функции приспособленности от номера поколения при одном из запусков алгоритма приведен на рис. 17.

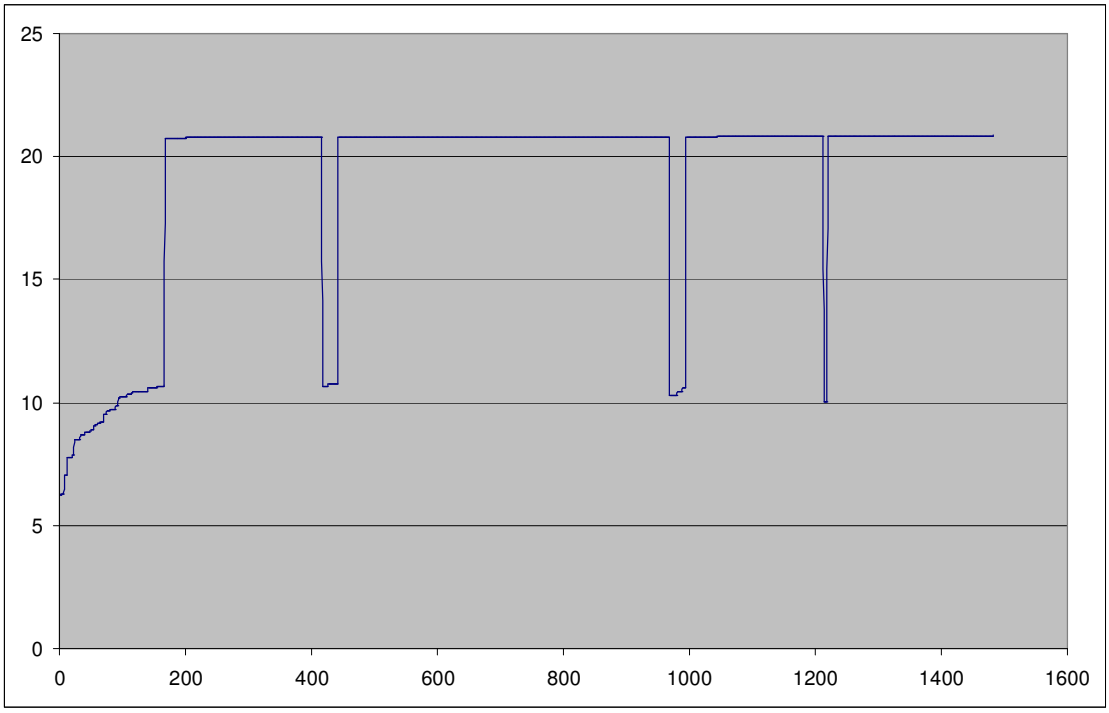

Рис. 17. Зависимость максимального значения функции приспособленности от номера поколения

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

Для каждого из запусков запоминалось число вычислений функции приспособленности в процессе построения автомата, которое равно числу просмотренных в процессе работы автоматов. Минимальное значение этой величины составило 256063, максимальное – 9239523, среднее значение – 1450467.28 (стандартное отклонение – 1106266.586).

Выводы по главе 4

1. Предложен метод машинного обучения на основе генетических алгоритмов для построения управляющих конечных автоматов Мили на основе тестовых примеров.
2. Этот метод состоит из алгоритма генетического программирования, осуществляющего построение структуры управляющего автомата Мили, и алгоритма расстановки пометок.
3. Предложен метод скрещивания описаний конечных автоматов, учитывающий поведение автоматов на тестовых примерах.
4. Приведена программная реализация этого метода на языке программирования *Java*.
5. Проведено экспериментальное исследование предложенного метода на задаче построения конечного автомата управления часами с будильником.

ЗАКЛЮЧЕНИЕ

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

- проведен аналитический обзор;
- проведены выбор и обоснование оптимального варианта направления исследований;
- подготовлен план проведения теоретических и экспериментальных исследований;
- осуществлены разработка, программная реализация и экспериментальное исследование метода машинного обучения на основе генетических алгоритмов для построения управляющих конечных автоматов Мили на основе тестовых примеров.

Аналитический обзор выполнен по следующим направлениям:

- машинное обучение;
- существующие алгоритмы машинного обучения для построения конечных автоматов, основанные на генетических алгоритмах;
- совместное применение генетических алгоритмов и верификации.

В результате выполнения аналитического обзора разработан план проведения теоретических и экспериментальных исследований. В соответствии с этим планом в рамках теоретических исследований будут созданы следующие методы машинного обучения для построения управляющих конечных автоматов на основе генетических алгоритмов:

- разработка, программная реализация и экспериментальное исследование метода машинного обучения на основе генетических алгоритмов для построения управляющих автоматов Мура и смешанных автоматов (Мили-Мура) на основе тестовых примеров;
- разработка, программная реализация и экспериментальное исследование метода машинного обучения на основе генетических алгоритмов с использованием верификации темпоральных свойств при вычислении функции приспособленности.

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

ИСТОЧНИКИ

1. *Ахо А., Сети Р., Ульман Дж.* Компиляторы: принципы, технологии и инструменты. М.: Вильямс, 2001.
2. *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/
3. *Бедный Ю. Д.* Применение генетических алгоритмов для генерации автоматов при построении модели максимального правдоподобия и в задачах управления. СПбГУ ИТМО. 2008. Магистерская диссертация. http://is.ifmo.ru/papers/bednij/
4. *Бек К.* Экстремальное программирование: разработка через тестирование. СПб: Питер, 2003.
5. *Воронин О., Дьюдни А.* Дарвинизм в программировании //Мой компьютер. 2004. № 35. http://www.mycomp.kiev.ua/text/7458
6. *Гладков Л. А., Курейчик В. В., Курейчик В. М.* Генетические алгоритмы. М.: Физматлит. 2006.
7. *Гуров В. С., Мазим М. А., Нарвский А. С., Шалыто А. А.* UML. SWITCH-технология. Eclipse // Информационно-управляющие системы. 2004. № 6. с. 12–17.
8. *Данилов В. Р.* Метод представления автоматов деревьями решений для использования в генетическом программировании //Научно-технический вестник СПбГУ ИТМО. Выпуск 53. Автоматное программирование. 2008, с. 103–108.
9. *Данилов В. Р.* Технология генетического программирования для генерации автоматов управления системами со сложным поведением. СПбГУ ИТМО. Бакалаврская работа. 2007.
10. http://is.ifmo.ru/papers/
11. *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
12. *Заде Л., Дезоер Ч.* Теория линейных систем. Метод пространства состояний. М.: Мир, 1970.
13. *Кларк Э., Грамберг О., Пелед Д.* Верификация моделей программ. М.: МЦНМО, 2002.
14. *Левенштейн В. И.* Двоичные коды с исправлением выпадений, вставок и замещений символов // Доклады Академии Наук СССР. 1963. №4, с. 845–848.
15. *Мазин М. А. Парфенов В. Г., Шалыто А. А.* Разработка интерактивных приложений *Macromedia Flash* на базе автоматной технологии. Проектная документация. СПбГУ ИТМО. 2003. http://is.ifmo.ru/projects/flash/
16. *Непейвода Н. Н.* Стили и методы программирования. М.: Интернет-Университет Информационных технологий, 2005.
17. *Николенко С. И.* Лекции по генетическим алгоритмам. http://logic.pdmi.ras.ru/~sergey/teaching/ml/
18. *Николенко С. И., Тулупьев А. Л.* Самообучающиеся системы. М.: МЦНМО, 2009.
19. *Норенков И. П., Арутюнян Н. М.* Метагенетический алгоритм оптимизации и структурного синтеза проектных решений //Информационные технологии. 2007. № 3.
20. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирования. СПб: Питер, 2009.
21. *Поликарпова Н. И., Точилин В. Н., Шалыто А. А.* Применение генетического программирования для генерации автоматов с большим числом входных переменных //Научно-технический вестник СПбГУ ИТМО. Выпуск 53. Автоматное программирование. 2008, с. 24–42.
22. *Полимеразная цепная реакция.* http://en.wikipedia.org/wiki/Polymerase_chain_reaction
23. *Сайт по генетическому программированию*. www.genetic-programming.com

24. *Технология генетического программирования* для генерации автоматов управления системами со сложным поведением. Промежуточный отчет по этапу I «Выбор направлений исследований и базовых компонентов». http://is.ifmo.ru/genalg/_2007_01_report-genetic.pdf
25. *Технология генетического программирования* для генерации автоматов управления системами со сложным поведением. Промежуточный отчет по этапу II «Теоретические исследования поставленных перед НИР задач». http://is.ifmo.ru/genalg/_2007_02_report-genetic.pdf
26. *Технология генетического программирования* для генерации автоматов управления системами со сложным поведением. Промежуточный отчет по этапу III «Экспериментальные исследования поставленных перед НИР задач». http://is.ifmo.ru/genalg/_2007_03_report-genetic.pdf
27. *Технология генетического программирования* для генерации автоматов управления системами со сложным поведением. Промежуточный отчет по этапу IV «Обобщение и оценка результатов исследований». http://is.ifmo.ru/genalg/_2007_04_report-genetic.pdf
28. *Туккель Н. И., Шалыто А. А.* Система управления дизель-генератором (фрагмент). Программирование с явным выделением состояний. Проектная документация. 2002. http://is.ifmo.ru/projects/dg/
29. *Туккель Н. И., Шалыто А. А.* От тьюрингова программирования к автоматному // Мир ПК. 2002. № 2, с. 144–149. http://is.ifmo.ru/works/turing/
30. *Фридман А., Меннон П.* Теория и проектирование переключательных схем. М.: Мир, 1978.
31. *Хопкрофт Дж., Мотвани Р., Ульман Дж.* Введение в теорию автоматов, языков и вычислений. М.: Вильямс, 2002.
32. *Царев Ф. Н., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов с минимальным числом состояний для задачи об «Умном муравье» /Тезисы научно-технической конференции «Научно-программное обеспечение в образовании и научных исследованиях». СПбГУ ПУ. 2008, с. 209–215.
33. *Царев Ф. Н.* Разработка метода совместного применения генетического программирования и конечных автоматов на примере игры «Летающие тарелки». СПбГУ ИТМО. Бакалаврская работа. 2007. http://is.ifmo.ru/papers/
34. *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. Выпуск 53. Автоматное программирование. 2008, с. 42–60.
35. *Шалыто А. А.* Switch-технология. Алгоритмизация и программирование задач логического управлениюю. СПб.: Наука, 1998.
36. *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
37. *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
38. *De Jong K.* Evolutionary computation: a unified approach. MA: Cambridge. MIT Press, 2006.
39. *Dempster A. P., Laird N. M., Rubin D. B.* Maximum likelihood from incomplete data via the EM algorithm // Stat. Soc. 1977. Vol. 39. № 1, pp. 1 – 38.
40. *Gold E. M.* Complexity of automaton identification from given data // Information and Control. 1978. № 37, pp. 302–320.

41. *Hamming R.* Error detecting and error correcting codes //Bell System Technical Journal. 29 (2), pp.147–160.
42. *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
43. *Johnson C.* Genetic Programming with Fitness based on Model Checking. Lecture Notes in Computer Science. Springer Berlin/Heidelberg. 2007. Volume 4445, pp. 114–124.
44. *Katz G., Peled D.* Genetic Programming and Model Checking: Synthesizing New Mutual Exclusion Algorithms. Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2008. Volume 5311, pp. 33–47.
45. *Koza J.* Genetic programming. On the Programming of Computers by Means of Natural Selection. MA: The MIT Press, 1998.
46. Learning DFS from Noisy Samples. A contest from GECCO 2004. http://cswww.essex.ac.uk/staff/sml/gecco/NoisyDFA.html
47. *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.
48. *Lucas S., Reynolds J.* Learning Deterministic Finite Automata with a Smart State Labeling Algorithm //IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol. 27. №7. 2005, pp. 1063–1074.
49. *Lucas S.* Evolving Finite-State Transducers: Some Initial Explorations. Lecture Notes in Computer Science. Springer Berlin/Heidelberg. 2003. Volume 2610, pp. 241–257. http://www.springerlink.com/content/41a34vg70fp1hltb/
50. *Mitchell M., Holland J., Forrest S.* When will a genetic algorithm outperform hill climbing? / Advances in Neural Information Processing Systems 6. California. 1994, pp. 51–58.
51. *Mitchell M.* An Introduction to Genetic Algorithms. MA: The MIT Press, 1996.
52. *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. UK. York. 1999, pp. 114–122.
53. *Niebert P., Peled D., Pnueli A.* Discriminative Model Checking. Lecture Notes //In Computer Science. Springer Berlin / Heidelberg. 2008. Vol. 5123, pp. 504–516.
54. *Spears W., Gordon D.* Evolving Finite-State Machine Strategies for Protecting Resources. Lecture Notes in Computer Science. Springer Berlin / Heidelberg. 2009. Volume 1932, pp. 5–28.
55. *Spichakova M.* Genetic Inference of Finite State Machines. Masters thesis. Tallinn. 2007. http://s-ma-u-g.googlecode.com/files/thesis.pdf
56. *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.
57. *Rabiner L. R.* A tutorial on hidden markov models and selected applications in speech recognition / Proceedings of the IEEE. 1989. Vol. 77. № 2, pp. 257 – 286. http://ieeexplore.ieee.org/iel5/5/698/00018626.pdf?arnumber=18626