МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»
(СПБГУ ИТМО)
ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ПРОГРАММИРОВАНИЯ
КАФЕДРА «КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ»

Ф. Н. ЦАРЕВ

РАЗРАБОТКА МЕТОДОВ СОВМЕСТНОГО ПРИМЕНЕНИЯ ГЕНЕТИЧЕСКОГО И АВТОМАТНОГО ПРОГРАММИРОВАНИЯ

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

НАУЧНЫЙ РУКОВОДИТЕЛЬ – ДОКТ. ТЕХН. НАУК, ПРОФЕССОР, ЗАВ. КАФЕДРОЙ «ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ» СПБГУ ИТМО А. А. ШАЛЫТО

САНКТ-ПЕТЕРБУРГ
2009

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Генетические алгоритмы [4, 35, 55] являются одним из современных и быстро развивающихся направлений в искусственном интеллекте [20]. С их помощью могут быть найдены решения многих задач в различных областях. Примерами таких задач являются: задачи синтеза расписаний, задачи планирования работ и распределения ресурсов, задачи маршрутизации транспортных средств, синтез топологии сетей различного назначения, построение искусственных нейронных сетей, деревьев принятия решений, автоматическое построение программ, проектирование *конечных автоматов* и клеточных автоматов.

Генетическое программирование [49] – разновидность генетических алгоритмов, в которой вместо низкоуровневого представления объектов в виде битовых строк используется высокоуровневое представление: деревья разбора программ, диаграммы переходов конечных автоматов и т.д. С помощью генетического программирования наиболее эффективно решаются задачи автоматического построения некоторых классов программ, конечных автоматов и клеточных автоматов.

Автоматное программирование [17] – парадигма программирования, при использовании которой программу предлагается строить в виде совокупности *автоматизированных объектов управления*, каждый из которых содержит систему управления (взаимодействующие конечные автоматы) и объект управления.

Объект управления характеризуется множеством *вычислительных состояний*, а также двумя наборами функций: множеством *предикатов*, отображающих вычислительные состояния в логические значения (истина или ложь), и множеством *действий*, позволяющих изменять вычислительные состояния.

Управляющий автомат определяется конечным множеством *управляющих состояний*, *функцией переходов* и *функцией выходов (действий)*. Взаимодействие между автоматами может осуществляться различными способами: за счет вложенности автоматов, с помощью обмена сообщениями, с помощью обмена номерами состояний и т.д.

Управляющие конечные автоматы часто характеризуются сложным поведением, как, например, в задаче «Умный муравей–3», рассматриваемой в настоящей работе. При сложном поведении эвристическое проектирование автоматов представляет собой весьма трудоемкую задачу. Возникает естественное желание – автоматизировать процесс проектирования автоматов, поручив основную работу компьютеру.

В настоящей работе в качестве метода автоматизированного построения автоматов выбрано *генетическое программирование*. *Цель работы* состоит в разработке нового метода:

- совместного применения генетического и автоматного программирования, основанного на использовании тестов;
- представления конечных автоматов, который может быть использован в контексте традиционного подхода к совместному применению генетического и автоматного программирования.

В существующих работах по совместному применению автоматного и генетического программирования для вычисления функции приспособленности используется моделирование работы системы со сложным поведением в некоторой внешней среде. Одним из недостатков этого метода является то, что при его применении для новой задачи необходимо полностью «с нуля» программно реализовывать вычисление функции приспособленности. Кроме этого, моделирование часто связано с больш*и*ми затратами вычислительных ресурсов. Эти недостатки ограничивают возможность применения подхода на

основе моделирования (далее он будет называться традиционным подходом) на практике.

В работе разрабатывается метод совместного применения генетического и автоматного программирования, лишенный указанных недостатков. Он основан на использовании тестов. Его применение на практике упрощено, так как тесты достаточно активно применяются в разработке программного обеспечения (ПО). Поэтому этот метод проще встроить в существующие процессы разработки ПО. Для достижения этой цели предполагается решение *следующих задач*:

- разработка структуры хромосомы алгоритма генетического программирования;
- разработка алгоритмов мутации и скрещивания;
- апробация метода на примере задачи построения автомата управления часами с будильником [17].

С другой стороны, метод, основанный на тестах, предполагает, что разработчик уже обладает представлением не только о желаемых результатах поведения системы, но и о самом поведении. Если же необходимо строить систему со сложным поведением только на основе информации о желаемом результате ее поведения, то единственным методом, который может быть применен, является традиционный подход, основанный на моделировании.

В работе также производится исследование возможностей традиционного подхода при использовании новых методов представления автоматов с большим числом входных переменных. Для представления таких автоматов ранее были разработаны такие методы как метод сокращенных таблиц переходов, и метод представления автоматов деревьями решений [17].

В настоящей работе для представления автоматов вместо деревьев решений предлагается использовать конечные преобразователи. Для достижения этой цели предполагается решение следующих задач:

- разработка структуры хромосомы алгоритма генетического программирования;
- разработка алгоритмов мутации и скрещивания;
- сравнение нового метода с существующим на примере задачи «Умный муравей–3».

В начале работы излагаются общие концепции генетических алгоритмов, генетического программирования и автоматного программирования, приводится перечень задач, в которых успешно применяются указанные методы. Далее описывается метод представления управляющих конечных автоматов с помощью конечных преобразователей, применение которого иллюстрируется на примере задачи «Умный муравей–3». Приводится его сравнение с решением, построенным вручную, и с решением, построенным с помощью метода представления автоматов деревьями решений. После этого описывается метод построения конечных автоматов управления системами со сложным поведением на основе тестов. Предложена структура особи алгоритма генетического программирования, алгоритм расстановки пометок, операции мутации и скрещивания для разработанной структуры особи. Применение этого метода иллюстрируется на примере построения автомата управления часами с будильником.

Метод построения автоматов на основе тестов интересен тем, что при его применении нет необходимости строить программную модель внешней среды для вычисления функции приспособленности – поэтому требуется меньше вычислительных ресурсов. При этом для новой задачи необходимо подготовить только новый набор тестов, а «ядро» вычисления функции приспособленности остается неизменным.

В заключении работы анализируются ее результаты, и приводится ряд открытых вопросов и направлений для дальнейших исследований в этой области.

ГЛАВА 1. АВТОМАТНОЕ И ГЕНЕТИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

В настоящей главе приводятся общие концепции автоматного и генетического программирования. Рассматривается понятие «система со сложным поведением» и обосновывается применение автоматного подхода для описания систем этого типа. Приводятся примеры задач, в которых генетические алгоритмы успешно применяются для автоматизированного построения конечных автоматов.

1.1. Основные концепции автоматного программирования

В настоящем разделе описывается понятие «системы со сложным поведением» и приводятся основные идеи автоматного программирования.

1.1.1. Системы со сложным поведением

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

Будем считать, что система обладает «нетривиальным» поведением, если в ответ на возникновение некоторого *события* она, в зависимости от предыстории, может совершить одно из нескольких *действий.*

При традиционной программной реализации систем с таким поведением программистами используются переменные, называемые *флагами*, которым не соответствуют никакие элементы предметной области. Предназначение флагов – участвовать в многочисленных конструкциях ветвления, реализующих логику поведения. Флаги неявно задают отдельные компоненты состояний. Использование флагов трудно для понимания, подвержено ошибкам и практически не расширяемо.

Вместо этого в последнее время предлагается описывать системы со сложным поведением, приписывая каждой из них некоторое множество *управляющих состояний*. В этих состояниях поведение системы обычно является простым и может быть описано явно (противном случае выполняется декомпозиция автоматов и/или состояний). Связь управляющих состояний с действиями и механизм переходов между состояниями удобно описывать с помощью *конечных автоматов* с выходами [23]. При этом все поведение системы оказывается сосредоточенной в автомате или системе взаимодействующих автоматов. Такой подход к описанию поведения называют *автоматным* [12].

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

Одна из центральных идей автоматного программирования [17] состоит в отделении описания *логики* поведения (при каких условиях необходимо выполнить те или иные действия) от описания его *семантики* (собственно смысла каждого из действий). Кроме того, описание логики при автоматном подходе жестко структурировано. Эти два свойства делают автоматное описание сложного поведения ясным и удобным.

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

Если в системах логического управления [24] в качестве входных воздействий используются опрашиваемые программой двоичные входные

переменные и предикаты, соответствующие определенным состояниям автоматов, взаимодействующих с рассматриваемым автоматом, то для «реактивных» систем это понятие расширено. Во-первых, в качестве входных переменных применяются любые подпрограммы (функции), возвращающие двоичные значения, а, во-вторых, введены события, не только обеспечивающие возможность выполнения переходов в автоматах, но и инициирующие запуск автоматов. События могут также инициировать реализацию выходных воздействий в случае, когда состояние автомата не изменяется.

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

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

Кроме того, если в системах логического управления наиболее целесообразно применять такую структурную модель как автомат Мура, то в «реактивных» (событийных) системах часто более рационально использовать другую модель – смешанный автомат, совмещающий в себе свойства автоматов Мура и Мили.

В качестве примера системы со сложным поведением, управляемой автоматами, приведем систему управления дизель-генератором [22]. На рис. 1 изображена схема взаимодействия автоматов, которые образуют эту систему.

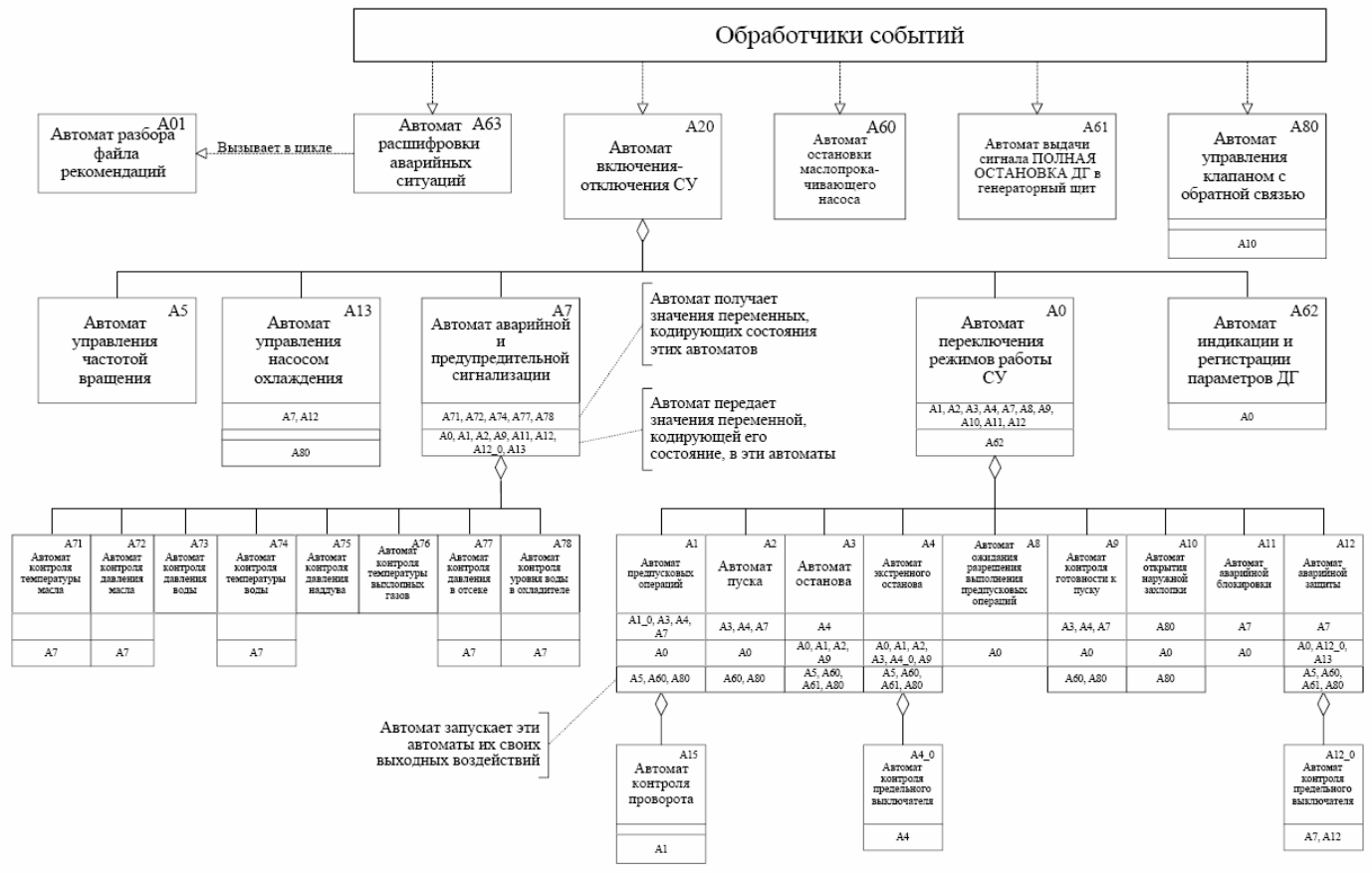

Рис. 1. Схема взаимодействия автоматов, управляющих системой со сложным поведением

1.1.2. Автоматное программирование

Парадигма автоматного программирования [17] состоит в представлении систем со сложным поведением в виде автоматизированных объектов управления. Автоматизированный объект управления представляет собой объект управления, интегрированный с системой управления в одно устройство. При этом система управления обладает сложным поведением и представляется в виде системы взаимодействующих конечных автоматов, а объект управления обладает простым поведением и реализуется традиционными методами.

В случае, когда автоматы описывают системы со сложным поведением, их проектирование является нетривиальной и трудоемкой задачей. Поэтому возникает естественное желание – *автоматизировать процесс построения автоматов*, поручив основную работу компьютеру. При этом нет необходимости,

как при традиционном подходе, заранее учитывать все особенности решаемой задачи и действия, которые должна предпринимать программа. Одним из возможных методов проектирования автоматов, соответствующих этим требованием, являются *генетические алгоритмы.*

1.2. Генетические алгоритмы

Генетический алгоритм (*genetic algorithm*) [4, 20, 35, 55] представляет собой стохастический метод оптимизации. Основная идея генетических алгоритмов состоит в использовании принципа *естественного отбора*, который составляет основы теории эволюции живых организмов, предложенной Чарльзом Дарвином.

Подобные идеи возникали у ряда исследователей. В 1962 г. Л. Фогель занялся изучением интеллектуального поведения индивида и его развития в процессе эволюции [38]. При этом поведение индивида задавалось конечным автоматом. Продолжая данные исследования, Л. Фогель, А. Оуэнс и М. Уолш предложили в 1966 г. схему эволюции конечных автоматов, решающих задачи предсказания [39]. Примерно в это же время (в середине 60-х годов) Дж. Холланд разработал новый метод поиска оптимальных решений – генетические алгоритмы. Результатом работы Дж. Холланда стала книга [45], вышедшая в 1975 г. Эти работы заложили основы эволюционных вычислений.

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

В генетических алгоритмах используются те же принципы. В качестве особей выступают элементы пространства возможных решений некоторой задачи (маршруты коммивояжера, диаграммы переходов автомата и т. п.). Задан набор генетических операций, с помощью которых из существующих особей

формируются новые. Кроме этого определена так называемая *функция приспособленности* (*fitness-function*), которая показывает, насколько «хорошим» решением задачи является особь.

Процесс работы генетического алгоритма состоит в генерации поколений особей до тех пор, пока не будет выполнено некоторое условие останова (например, достигнуто целевое значение функции приспособленности или сгенерировано заданное число поколений). На рис. 2 приведена общая схема работы генетического алгоритма.

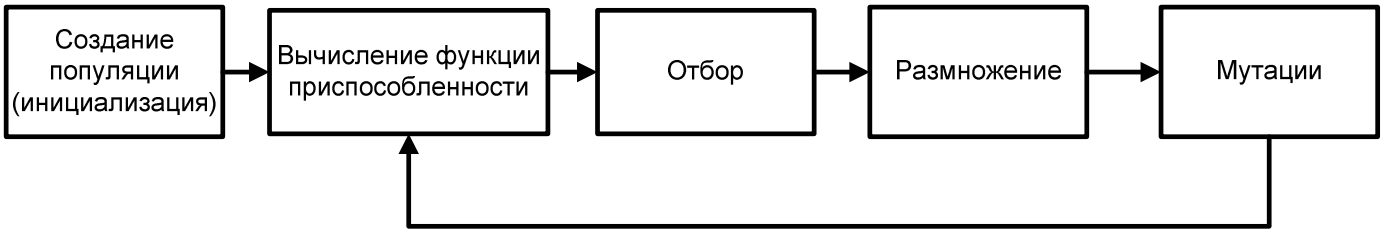

Рис. 2. Общая схема работы генетического алгоритма

По сравнению с традиционными методами оптимизации генетические алгоритмы имеют ряд преимуществ:

- генетические алгоритмы легко модифицируются для параллельных вычислений;
- они хорошо подходят для оптимизации недиффернцируемых функций.

Основными недостатками генетических алгоритмов являются:

- высокая трудоемкость, ограничивающая их область применения;
- сложность оценки степени пригодности конкретных генетических операций для конкретной задачи.

1.2.1. Основные определения

Приведем ряд определений, которые играют важную роль в теории эволюционных вычислений и необходимы для описания генетических алгоритмов.

Поколение (популяция) – это совокупность особей, способная к устойчивому самовоспроизводству, относительно обособленная от других групп, с представителями которых потенциально возможен генетический обмен.

Индивид (особь) – единичный представитель популяции.

Фенотип – совокупность характеристик, присущих индивиду на определённой стадии развития. Фенотип формируется на основе генотипа, опосредованного рядом факторов внешней среды.

Генотип – наследственная информация, закодированная в хромосомах, которая вместе с факторами внешней среды определяет фенотип организма. Генотип не всегда соответствует одному и тому же фенотипу. Некоторые гены проявляются в фенотипе только в определенных условиях.

Хромосома – структура, содержащая генетический код индивида.

Ген – определенная часть хромосомы, кодирующая врожденное качество индивида.

Генетические алгоритмы – это оптимизационный метод, базирующийся на принципах естественной эволюции популяции особей (индивидов). Каждая особь характеризуется приспособленностью – функцией ее генов. Задача оптимизации состоит в максимизации функции приспособленности. В процессе эволюции – в результате отбора, рекомбинаций, мутаций геномов особей, а также применения ряда других генетических операторов, происходит поиск особей с высокой приспособленностью. По сравнению с обычными оптимизационными методами генетические алгоритмы имеют особенности: параллельный поиск, случайные мутации и рекомбинации уже найденных хороших решений. Приведем описание генетических операторов.

Как известно в теории эволюции, важную роль играет то, каким образом признаки родителей передаются потомкам. В генетических алгоритмах за передачу признаков родителей потомкам отвечает оператор, который называется

скрещивание. В литературе по генетическим алгоритмам этот оператор называют также кроссинговер, кроссовер, рекомбинация.

Мутация – случайное изменение одной или нескольких позиций в хромосоме. Мутации, которые проявляются на уровне фенотипа, могут иметь как отрицательные последствия, так и положительные – приводить к появлению у индивида новых полезных признаков. Таким образом, мутации являются двигателем естественного отбора, так как являются механизмом поддержания разнообразия особей в популяции.

Из-за избыточности и неоднозначности кодирования одному и тому же фенотипу может соответствовать множество генотипов. При этом часто порядок генов в хромосоме является критическим для строительных блоков, позволяющих осуществлять эффективную работу алгоритма. В ряде работ были предложены методы для *переупорядочения* позиций генов в хромосоме во время работы алгоритма. Одним из таких методов является инверсия, выполняющая реверсирование порядка генов между двумя случайно выбранными позициями в хромосоме. Когда используются методы переупорядочения, гены должны содержать некоторую «метку», такую, чтобы их можно было правильно идентифицировать независимо от их позиции в хромосоме. Цель переупорядочения состоит в том, чтобы попытаться найти порядок генов, который имеет лучший эволюционный потенциал. Многие исследователи использовали инверсию в своих работах, однако мало кто пытался ее обосновать или определить ее вклад. Переупорядочение также значительно расширяет область поиска. Мало того, что генетический алгоритм пытается находить «хорошие» множества генов, но он также одновременно пробует находить хорошее упорядочение генов. Это гораздо более трудная задача для решения. Еще раз отметим, что оператор переупорядочения не изменяет фенотипа индивидуума, а изменяет лишь структуру хромосомы.

Как правило, выделяют еще один оператор для генетических алгоритмов – *оператор генерации случайной особи*, который может быть использован при создании начальной популяции, при пополнении популяции случайными особями и при некоторых мутациях.

1.2.2. Традиционный генетический алгоритм

В этом разделе приведено краткое описание *традиционного генетического алгоритма* (*conventional genetic algorithm*) [55] 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*) [4].

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, …, 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 [15].

1.2.3. Генетическое программирование

Генетическое программирование (*genetic programming*), предложенное J.R. Koza в 1992 году [49], 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 [21].

1.3. Применение генетических алгоритмов для построения конечных автоматов

В данном разделе приведены примеры задач, в которых применение эволюционных алгоритмов для автоматизированного построения автоматов позволило улучшить существующие решения или получить решения, сравнимые по эффективности с существующими. Кроме того, следует учесть, что трудозатраты человека при подходе, базирующемся на эволюционных алгоритмах, как правило, значительно меньше, чем при эвристическом построении автоматов. Важно отметить и то обстоятельство, что в некоторых случаях автоматизировано построенные автоматы могут быть изучены для выявления структуры «хороших» решений задачи.

При использовании эволюционных методов не требуется точно указывать, как решать задачу. Вместо этого достаточно ее сформулировать в терминах некоторой «цели» и «допустимых затрат». При этом программа будет создана самой вычислительной системой в ходе эволюционного процесса. *Высказывание «машина никогда не знает больше программиста» при этом оказывается просто неверным.*

1.3.1. Эксперименты Фогеля

Один из создателей эволюционного программирования Л. Фогель рассматривал интеллектуальное поведение индивида, как способность успешно предсказывать поведение среды, в которой он находится, и действовать в

соответствии с этим. В 60-х годах прошлого века Л. Фогель поставил ряд экспериментов [39] по созданию искусственных систем, способных адаптироваться к первоначально не известной им среде.

В проведенных экспериментах Л. Фогель моделировал поведение простейшего живого существа, которое способно предсказывать изменения параметра среды, обладающего *периодичностью*. Это живое существо моделировалось конечным автоматом с действиями на переходах – автоматом Мили. В качестве среды выступала последовательность символов над двоичным алфавитом, например, $(1111010010)^*$. На вход автомата в каждый момент времени подавалась битовое значение параметра окружающей среды. Автомат формировал значение выходной переменной – возможное значение рассматриваемого параметра в следующий момент времени. На рис. 3 приведен пример одного из автоматов, который построен для распознавания, указанной выше последовательности.

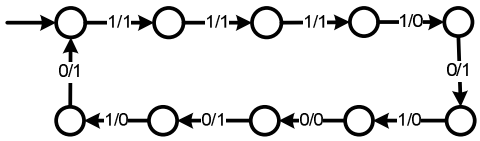

Рис. 3. Граф переходов эвристически построенного автомата

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

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

- добавление состояния;
- удаление состояния (в случае, если число состояний больше единицы);
- замена начального состояния (в случае, если число состояний больше единицы);
- замена перехода;
- замена действия на переходе.

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

По мнению Л. Фогеля, результаты экспериментов показали, что эволюционное программирование может быть успешно применено для

построения «интеллектуальных» искусственных систем. При этом было отмечено, что построение вручную автоматов столь же результативных и простых, как те, что были построены эволюционным алгоритмом, является крайне сложной задачей.

1.3.2. Задача «Умный муравей»

Приведем описание задачи «Умный муравей» на основе работ [28, 45]. Используется двумерный тор размером 32 на 32 клетки (рис. 4). На некоторых клетках поля расположены яблоки – черные клетки на рис. 4. 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.

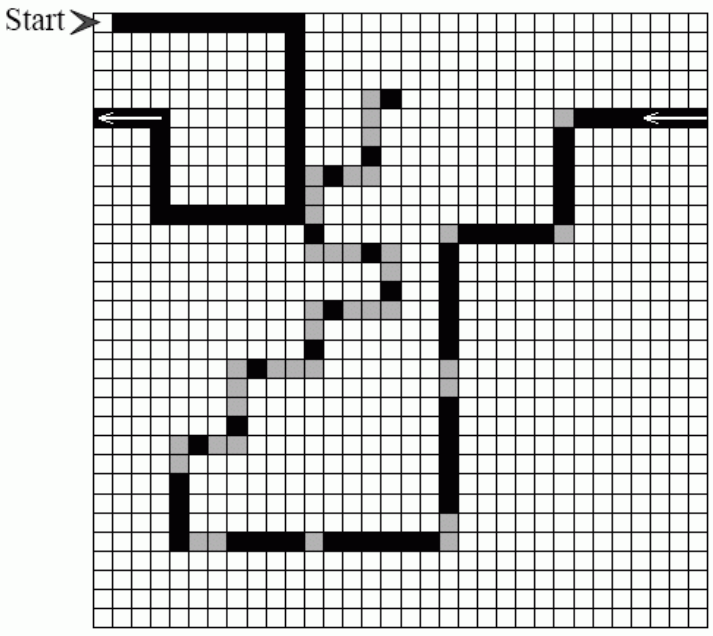

Рис. 4. Поле с яблоками

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

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

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

Цель игры – создать муравья, который за 200 ходов съест как можно больше яблок. Муравьи, съевшие одинаковое число яблок, заканчивают игру с одинаковым результатом вне зависимости от числа ходов, затраченных каждым из них на процесс еды. Однако эта задача может иметь различные модификации, например, такую, в которой при одинаковом числе съеденных яблок, лучшим считается муравей, съевший яблоки за меньшее число ходов. Ниже будет показано, что поведение муравья может быть задано конечным автоматом. При этом может быть поставлена задача о построении автомата с минимальным числом состояний для муравья, съедающего все яблоки, или автомата для муравья, съедающего максимальное число яблок при заданном числе состояний.

Конечный автомат, изображенный на рис. 5, содержит всего пять состояний.

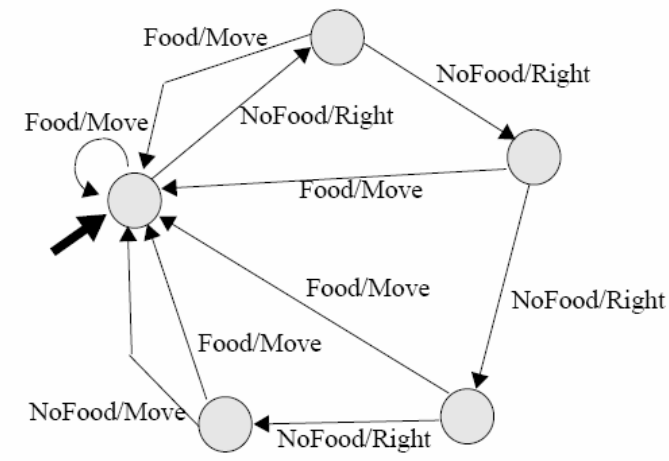

Рис. 5. Конечный автомат, задающий муравья

Этот автомат описывает поведение муравья, который не решает задачу – за 200 ходов съедает только 81 яблоко, а за 314 ходов — все 89 яблок. Муравей действует по принципу «Вижу яблоко – иду вперед. Не вижу – поворачиваю направо. Сделал круг, но яблок нет – иду вперед».

Приведем автомат (рис. 6), который построен в работе [46] с помощью генетических алгоритмов и решает поставленную задачу.

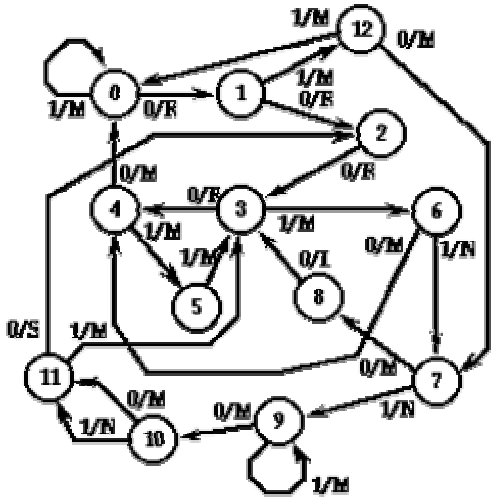

Рис. 6. Конечный автомат с 13 состояниями, задающий муравья, съедающего все яблоки за 200 ходов

На этом рисунке используются следующие обозначения: `N` – непосредственно перед муравьем нет еды, `F` – непосредственно перед муравьем есть еда; `M` – идти вперед, `L` – повернуть налево, `R` – повернуть направо, `NO` – стоять на месте. Можно заметить, что действие `NO` никогда не выполняется. Данный автомат изображен некорректно – из *одиннадцатого* состояния изображен переход, помеченный входным воздействием 0, с «непонятным» действием `S`. Однако если это действие заменить на `N`, то автомат корректно решает задачу.

Автомат из работы [28], приведенный на рис. 7, содержит 11 состояний. По утверждению авторов, он съедает все яблоки за 193 хода.

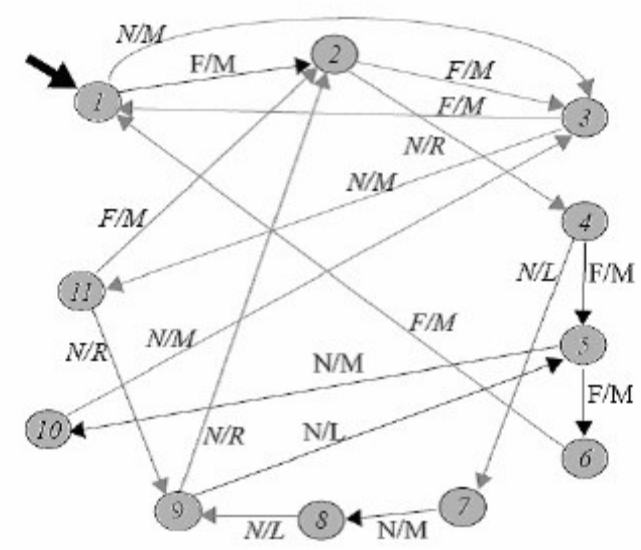

Рис. 7. Конечный автомат с 11 состояниями, задающий муравья, съедающего все яблоки за 193 хода

Для генерации конечных автоматов, задающего муравья, в работах [28, 46] были применены генетические алгоритмы. В работе [46] приспособленность особи (*fitness*) определяется как число яблок, съеденное за 200 ходов. Кодирование автомата, задающего поведение муравья, в особь генетического алгоритма (битовую строку) в этой работе осуществлялось следующим образом: входному

воздействию F сопоставлялась единица, а N – ноль. Каждое из четырех действий муравья кодировалось двоичными числами: 00 – NO, 01 – L, 10 – R, 11 – M.

Покажем, как выполняется кодирование автомата в целом. На рис. 8 приведен пример графа переходов автомата, описывающего поведение некоторого муравья.

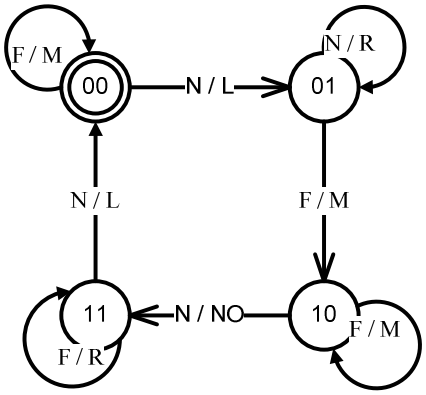

Рис. 8. Пример автомата, описывающего поведение муравья

Кодирование этого графа переходов приведено в табл. 1.

Таблица 1. Кодирование графа переходов

Состояние	**Вход**	**Новое состояние**	**Действие**
00	0	01	01
00	1	00	11
01	0	01	10
01	1	10	11
10	0	11	00
10	1	10	11
11	0	00	01
11	1	11	10

Преобразуем эту таблицу в битовую строку. Для этого сначала требуется запомнить число состояний автомата (в данном случае четыре). В начале строки задан двоичный номер начального состояния автомата (в данном примере – 00).

Далее записаны пары (Новое состояние, Действие). Содержимое полей (Состояние, Вход) не записываются, так как они повторяются для каждого автомата с фиксированным числом состояний. Для рассматриваемого примера искомая строка имеет вид:

00 *01*01 *00*11 *01*10 *10*11 *11*00 *10*11 *00*01 *11*10

Здесь курсивом выделены биты, соответствующие новому состоянию, а обычным шрифтом — биты, соответствующие выполняемому на переходе действию. Жирным шрифтом выделены биты, кодирующие начальное состояние.

Отметим, что автором настоящей диссертации в бакалаврской работе [24] был предложен генетический алгоритм, с помощью которого был построен автомат, содержащий семь состояний и решающий задачу об «Умном муравье». Диаграмма переходов этого автомата приведена на рис. 9. В работе [26] с помощью перебора показано, что автоматы с меньшим, чем семь числом состояний задачу «Умный муравей» решить не могут.

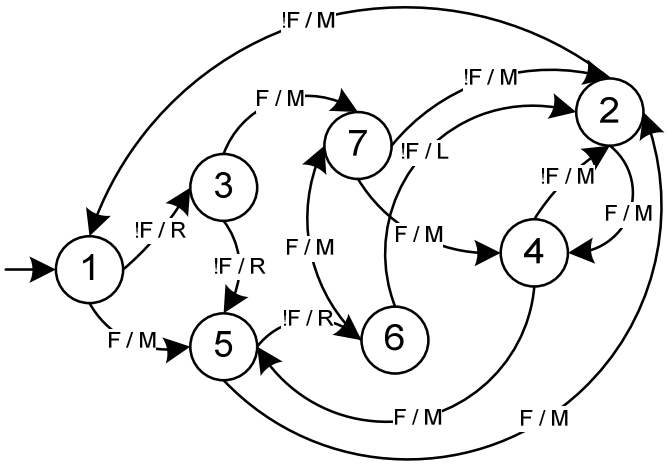

Рис. 9. Автомат с семью состояниями, решающий задачу об «Умном муравье»

Отметим, что в книге [49] создатель генетического программирования *J. R. Koza* предлагает другой подход к представлению автомата, по сравнению с кодированием с помощью битовой строки, – поведение муравья задается в виде

дерева разбора, которое далее может быть преобразовано в автомат. Апробация этого метода проводилась на другом игровом поле – *Santa Fe Trail*.

1.3.3. Построение конечных детерминированных автоматов для распознавания регулярных языков

Из теории формальных языков известно, что конечные детерминированные автоматы способны распознавать регулярные языки [23]. 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 [41] показано, что эта задача является *NP*-трудной.

В работе [32] автоматы представлялись с помощью таблицы переходов. Разработанный в этой работе метод позволяет поддерживать в популяции автоматы с разным числом состояний. Функция приспособленности учитывает три компонента – число верно распознанных тестовых примеров, число состояний и переходов автомата, степень общности языка, соответствующего построенному автомату. Это позволяет сузить область поиска, и находить языки, соответствующие определенным критериям. В тестовую коллекцию могут входить примеры слов, которые как принадлежат, так и не принадлежат языку.

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

Генетическая операция мутации 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 ряде задач. Авторам удалось построить автомат из семи состояний, который распознает по множеству из ста примеров русские двухсложные слова.

В работе [34] для генерации автомата был применен следующий вариант генетического программирования: выводилось выражение, результатом вычисления которого являлся автомат. Выражение описывало последовательность действий, преобразующих исходный автомат (рис. 10) в результирующий. Для вывода выражений, кодирующих развитие автомата, было применено традиционное генетическое программирование, оперирующее над деревьями выражений.

Рис. 10. Исходный автомат

Опишем процесс преобразования автомата. Построение автомата является неким аналогом онтогенеза – перехода клетки в организм путем деления и модификации. В каждый момент некая дуга автомата является активной. Операции, используемые в выражении, могут делить активную дугу или модифицировать ее, выбрав при этом другую дугу в качестве активной. Авторами этой работы вводятся следующие нетерминалы:

- `PARALLEL` – создать новое допускающее состояние автомата и направить туда активную дугу. При этом созданное состояние копируется из состояния, в которое активная дуга вела до операции;

- PARALLEL-REJ – создать новое отвергающее состояние и направить туда активную дугу аналогично PARALLEL;
- RETRACT – зациклить активную дугу: направить ее в состояние, из которого она исходит;
- WRITE-NODE – положить в стек состояние, в которое ведет активная дуга;
- TO-NODE – снять состояние со стека и направить в него активную дугу. Если стек пуст, то активная дуга не модифицируется.

В работе [34] введен единственный терминал DONE, означающий конец редактирования. Пример выражения, использующего эти функции, приведен на рис. 11. Для наглядности нетерминалы пронумерованы в порядке исполнения. Для нетерминалов PARALLEL и PARALLEL-REJ первый аргумент соответствует действиям, совершаемым над созданным состоянием. При этом активная дуга переходит в первую дугу, исходящую из нового состояния. Второй аргумент этих нетерминалов соответствует действиям, совершаемым над исходным состоянием. При этом активная дуга заменяется следующей дугой, исходящей из состояния. Для остальных нетерминалов единственный аргумент вычисляется аналогично второму аргументу PARALLEL и PARALLEL-REJ.

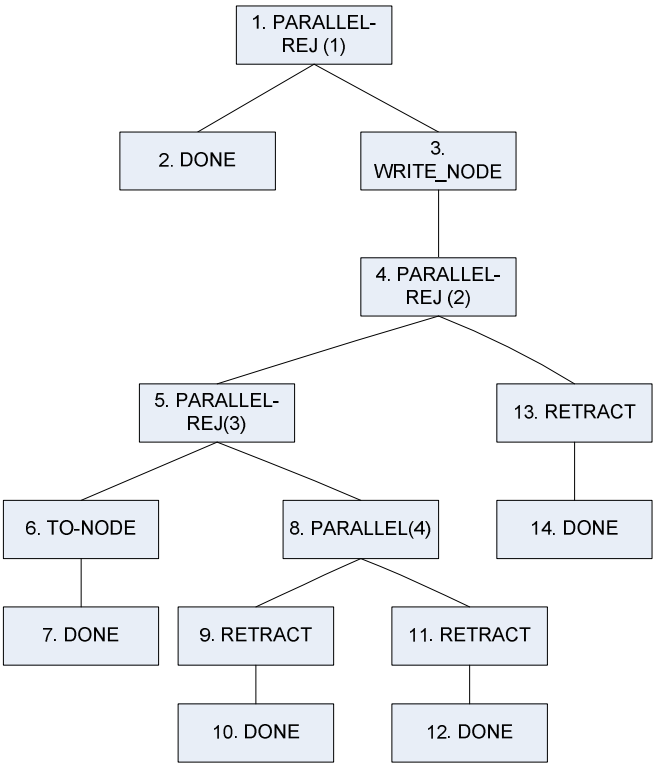

Рис. 11. Выражение, описывающее развитие автомата

На рис. 12 показан процесс развития автомата из исходного по приведенному выражению.

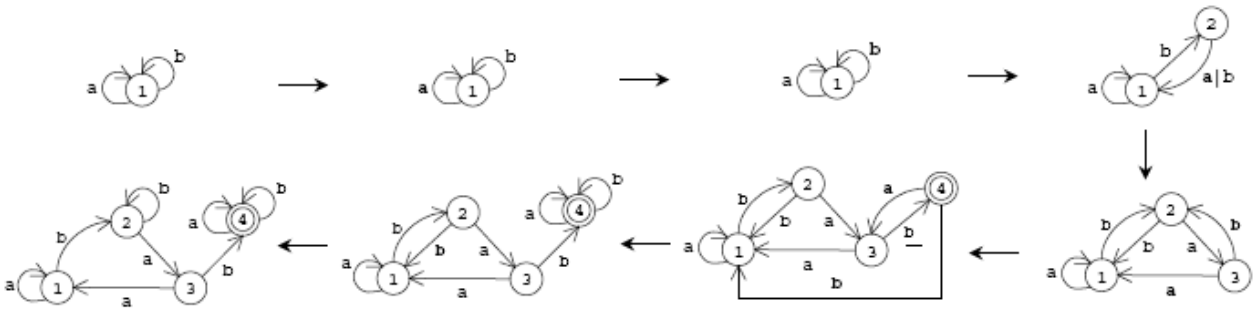

Рис. 12. Процесс построения автомата при вычислении выражения

Известно, что регулярные языки могут быть представлены как объединение, пересечение или дополнение других регулярных языков. Следовательно, можно произвести декомпозицию языка на более простые языки, для которых проще найти распознающие автоматы. В работе [34] определено решение в виде

комбинации трех автоматов через нетерминалы AND, OR и NOT. Разложение также определяется с помощью генетического программирования. Пример выведенного с помощью этой методики распознавателя изображен на рис. 13.

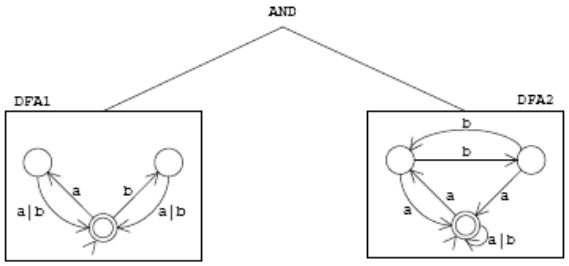

Рис. 13. Декомпозиция распознавателя

Описанный подход был протестирован авторами указанной работы на наборе из девяти регулярных языков. К сожалению, для одного из тестовых языков так и не удалось построить автоматический распознаватель.

В работе [51] произведено сравнение двух подходов к автоматическому построению автоматов: использование генетических алгоритмов и применение эвристики *Evidence Driven State Merging (EDSM)*. После анализа проведенных экспериментов был сделан вывод, что автомат, построенный с помощью генетических алгоритмов, лучше соответствует искомому языку. С другой стороны, эвристика *EDSM* всегда строит автомат, который верно распознает множество примеров.

Кроме этого, в работе [53] предлагается подход, позволяющий существенно сократить пространство поиска в задаче построения конечного автомата для распознавания регулярного языка. Этот подход состоит в том, что в хромосоме генетического алгоритма кодируется только структура автомата, а его вершины помечаются одним из значений – «допускающее состояние» или «не допускающее состояние» – с помощью процедуры, которую авторы работы назвали «расстановкой пометок» (*smart state labeling*). Применение этого подхода позволило существенно ускорить работу генетического алгоритма.

1.3.4. Построение конечных преобразователей

Одной из областей применения автоматического построения автоматов является построение конечных преобразователей (*Finite State Transducers*), которые преобразуют входной поток в последовательность символов выходного алфавита. На рис. 14 приведен пример простого преобразователя двоичной записи числа *N* в двоичную запись числа *N + 1*. При этом двоичная запись числа подается с конца.

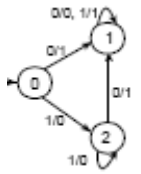

Рис. 14. Простой пример конечного преобразователя

Более сложным примером использования автоматических преобразователей является синтез речи – отображение последовательности букв в последовательность фонем. Важным достоинством применения конечных автоматов для описания преобразователей является легкость их реализации.

Как правило, в задаче построения конечного преобразователя исходными данными является «обучающее множество»: пары последовательностей символов, которые состоят из входной последовательности и соответствующей ей выходной последовательности. Задача же состоит в построении конечного преобразователя, который содержит заданное число состояний и корректно преобразует последовательности, входящие в обучающее множество.

В работе [57] предлагается метод генетического программирования для построения конечных автоматов, реализующих преобразователи. Автоматы в этой работе представляются в виде графов переходов, генетические операции определяются аналогично генетическим операциям над абстрактными помеченными графами. При этом ребра помечаются входным и выходным воздействиями.

Приведем описание операции рекомбинации. В используемом методе оператор рекомбинации принимает на вход две особи и возвращает две порожденных особи. Используется следующий алгоритм.

1. В двух рекомбинируемых графах выбирается по одной случайной вершине.
2. Подграфы, соответствующие вершинам, меняются местами.
3. Ребра, которые ведут наружу, направляются в случайные вершины.

На рис. 15 приведен пример рекомбинации.

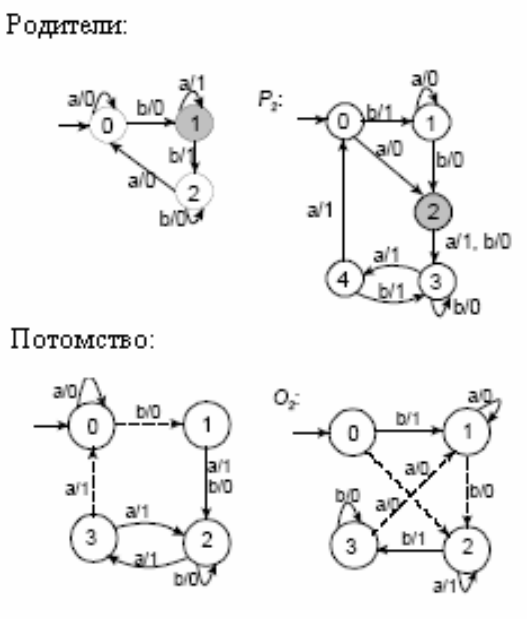

Рис. 15. Пример рекомбинации графов переходов

Операция мутации реализуется следующим образом.

1. Выбирается случайная вершина графа.
2. Подграф, соответствующий выбранной вершине, удаляется.
3. Из вершины выращивается случайный подграф.

Определенные таким образом генетические операции часто порождают потомство, имеющее меньшее значение функции приспособленности, чем у родителей. Для того чтобы исключить это, реализуется следующая эвристика: генетические операции повторяются, пока не будет построено потомство, превосходящее родителей. Если же после фиксированного числа итераций

приемлемого результата не достигается, то родители переходят в следующую популяцию.

Предложенный подход был проверен авторами указанной работы на шести простых задачах построения преобразователей. Результаты экспериментов подтверждают работоспособность метода.

В работе [52] выполняется сравнение эволюционных алгоритмов и методов, основанных на эвристическом объединении состояний. При этом конечные преобразователи представлены в генетическом алгоритме с помощью двух таблиц: функции переходов и функции выходов.

Кроме этого, исследуются три подхода к вычислению функции приспособленности – на основе строго сравнения выходной последовательности с последовательностью из «обучающего множества», на основе вычисления расстояния Хэмминга [43] и на основе вычисления редакционного расстояний (расстояния Левенштейна) [9]. Из этих трех методов наиболее эффективным оказался метод, основанный на редакционном расстоянии.

Результаты этой работы показывают, что наиболее эффективно (для рассмотренных в этой работе) примеров работает алгоритм *RMHC* (*random mutation hill climber*), использующий только операцию мутации. Вторым по эффективности является генетический алгоритм, а наименее эффективно работает эвристическое объединение состояний.

Отметим также, что задача построения конечного преобразователя, корректно преобразующего заданный «тренировочный» набор последовательностей, является одним из видов машинного обучения [14]. Поэтому указанные работы (а также настоящая работа, в которой рассматривается похожая задача) могут быть отнесены к этому разделу искусственного интеллекта.

1.3.5. Работы в СПбГУ ИТМО

В рамках исследований по теме «Технология генетического программирования для генерации автоматов управления системами со сложным поведением» [18, 19] на кафедре «Технологии программирования» СПбГУ ИТМО был разработан ряд методов генерации конечных автоматов с помощью генетических алгоритмов.

Один из этих методов – метод сокращенных таблиц переходов был предложен в работе [17]. Этот метод позволяет решить проблему экспоненциального роста размера описания автомата, которая возникает при использовании полных таблиц переходов (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 $|S| \cdot 2^n$ (|S| – размер множества 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 управляющих состояний 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. 16).

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. 16. 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. Даниловым в работах [6, 7]. 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 начальным 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 автором настоящей диссертации в работах [24, 25]. При использовании этого метода для управления системой со сложным поведением предлагается совместно применять нейронную сеть и конечный автомат, которые совместно строятся генетическим алгоритмом.

При этом, как отмечалось выше, нейронная сеть используется для классификации значений вещественных входных переменных и выработки входных логических переменных для автомата, а автомат – для выработки выходных воздействий на систему со сложным поведением (рис. 17).

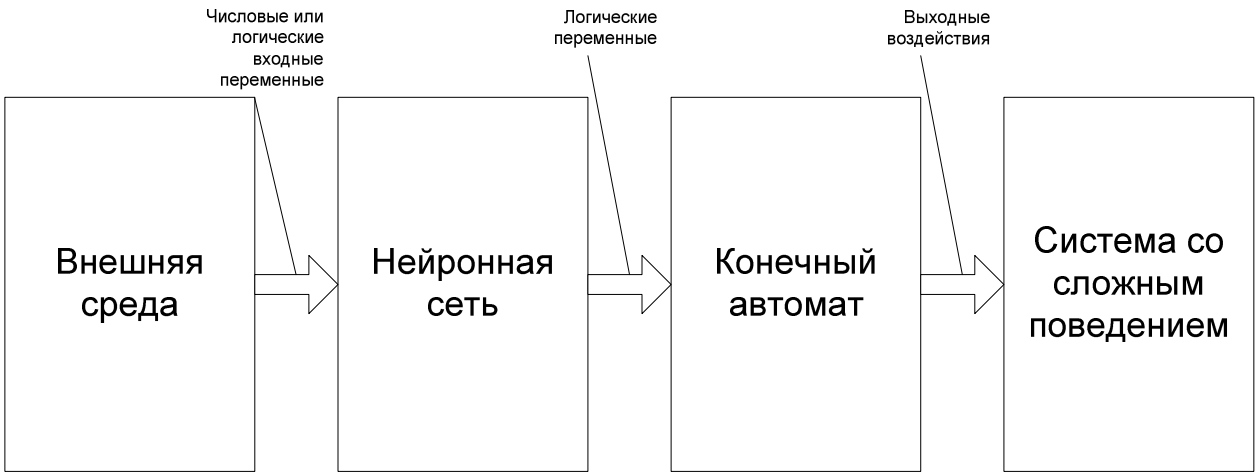

Рис. 17. Структурная схема системы управления при использовании метода совместного применения конечных автоматов и нейронных сетей

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

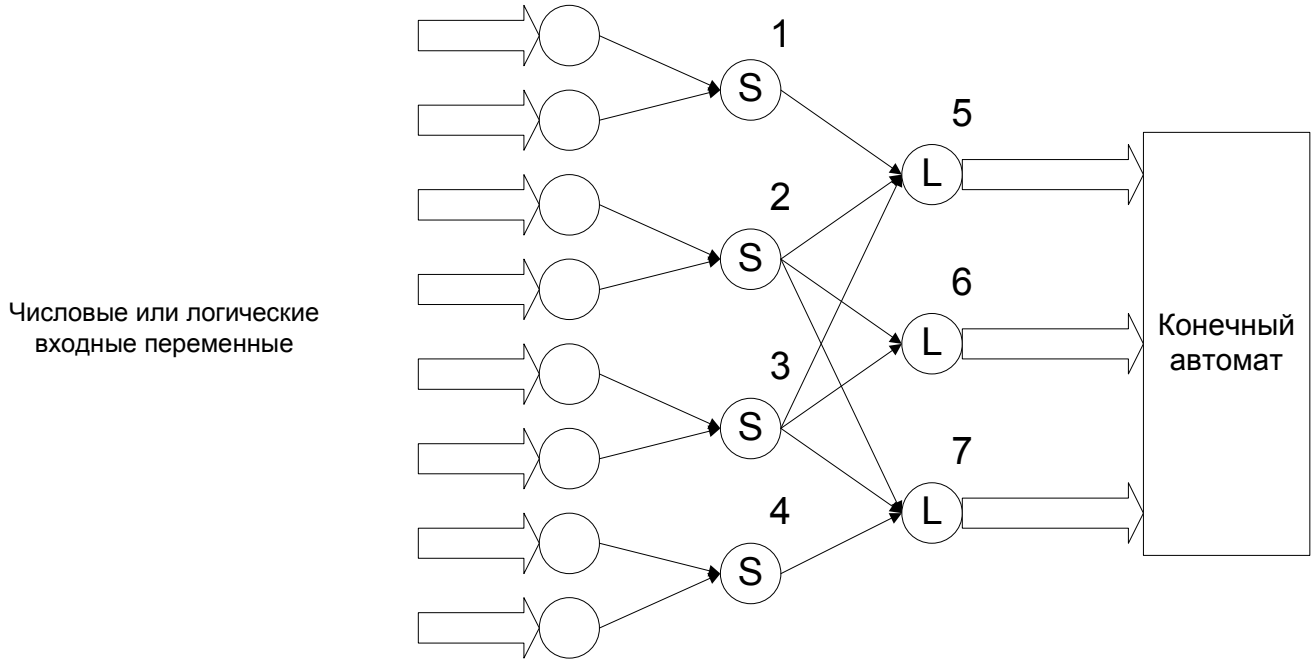

Рис. 18. Возможная структура нейронной сеть и ее взаимодействие с конечным автоматом

Символами *S* на рис. 18 обозначены нейроны с сигмоидальной функцией активации, символом *L* – нейроны с пороговой функцией активации. Рядом с нейронами указаны их номера (они используются при описании операции скрещивания нейронных сетей). На каждый из трех выходов нейронной сети поступает число равное нулю или единице. Таким образом, существует восемь вариантов комбинаций выходных сигналов нейронной сети (000, 001, 010, 011, 100, 101, 110, 111), подаваемых на вход конечного автомата.

Указанные методы применялись для построения конечных автоматов управления моделью беспилотного летательного аппарата. При этом в ряде случаев построенные автоматы были более эффективными, чем построенные вручную [18].

1.3.6. Другие работы по построению конечных автоматов с помощью генетических алгоритмов

Кроме указанных выше задач, генетические алгоритмы успешно применяются для построения автоматов в следующих задачах:

- итерированная дилемма узника (теория игр) [30, 37];
- распознавание изображений [33];
- автоматические переговоры [57, 60];
- управление человекоподобным роботом [61].

Выводы по главе 1

1. Автоматное программирование представляет собой парадигму программирования, которую целесообразно использовать при программировании систем, содержащих сущности со сложным поведением.
2. Существует ряд задач, в которых управляющие автоматы удается построить автоматически – с помощью генетических алгоритмов.

ГЛАВА 2. МЕТОД ПРЕДСТАВЛЕНИЯ АВТОМАТОВ С ПОМОЩЬЮ КОНЕЧНЫХ РАСПОЗНАВАТЕЛЕЙ

Традиционный метод совместного применения генетического и автоматного программирования основан на вычислении функции приспособленности особи с помощью моделирования работы системы со сложным поведением в некоторой внешней среде. Основные недостатки этого метода состоят в том, что для каждой задачи необходимо «с нуля» создавать программную реализацию функции приспособленности, а процесс моделирования, как правило, связан с большими затратами вычислительных ресурсов.

При применении традиционного подхода к совместному применению генетического и автоматного программирования акцент делается на представлении автоматов в виде особей генетического алгоритма – разработаны методы сокращенных таблиц переходов, представления автоматов деревьями решений и совместного применения конечных автоматов и нейронных сетей. Примером задачи, которая может быть решена с помощью этого подхода, является задача «Умный муравей–3».

В настоящей главе предлагается генетический алгоритм построения управляющего автомата на основе метода представления функции переходов с помощью конечных распознавателей, и приводится сравнение этого алгоритма с методом представления автоматов деревьями решений на примере задачи «Умный муравей–3».

2.1. Задача «Умный муравей–3»

В разд. 1.3.2 описана задача «Умный муравей». Постановка задачи «Умный муравей–3», предложенной в работе [1], 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. 19 изображена область обзора муравья (клетка, в которой находится муравей, обозначена серым цветом).

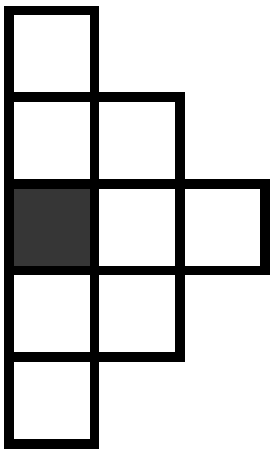

Рис. 19. Область видимости муравья

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 х 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 на 32. Поэтому для оценки эффективности автомата, задающего поведение муравья, вместо точного

вычисления этого математического ожидания, оно будет вычисляться приближенно – с помощью моделирования поведения муравья на 2000 случайно сгенерированных полях.

2.2. Решение задачи без применения конечных автоматов

Поведение муравья в задаче «Умный муравей-3» можно описывать следующим «жадным» алгоритмом – на каждом шаге муравей анализирует область видимости и находит клетку, содержащую еду и до которой можно добраться за минимальное число действий. Если такая клетка найдена, то муравей движется к ней по кратчайшему пути. Если же она не была найдена, то муравей делает шаг вперед в текущем направлении.

Отметим, что этот алгоритм можно реализовать и с помощью конечного автомата, однако он будет содержать достаточно большое число состояний.

Результаты вычислительных экспериментов для параметра μ, равного 0.01, 0.02, 0.03 и 0.04, приведены в табл. 2.

Таблица 2. Результаты вычислительного эксперимента для решения задачи «Умный муравей–3» без применения конечных автоматов

Значение параметра μ	**Результат**
0.01	2.8125
0.02	7.823
0.03	14.119
0.04	20.343

2.3. Решение задачи с помощью метода представления автоматов деревьями решений

В работах [6, 7] описано применение метода представления автоматов деревьями решений для построения управляющих конечных автоматов в задаче «Умный муравей–3». Результаты построения автоматов для различного числа состояний и различных значений параметра μ с применением метода представления автоматов деревьями решений приведены в табл. 3.

Таблица 3. Результаты вычислительного эксперимента для решения задачи «Умный муравей–3» с помощью метода представления автоматов деревьями решений (из работы [7])

Значение параметра μ	**Два состояния**	**Четыре состояния**	**Восемь состояний**	**16 состояний**
0.01	2.71	2.916	2.88	3.69
0.02	7.68	8.04	7.32	8.25
0.03	14.14	13.86	13.77	14.18
0.04	18.28	20.28	18.60	20.18

В работе [7] также приводятся результаты вычислительных экспериментов, проведенных с использованием методов представления автоматов битовыми строками и с помощью полных таблиц переходов (табл. 4 и 5).

Таблица 4. Результаты вычислительного эксперимента для решения задачи «Умный муравей–3» с помощью метода представления автоматов битовыми строками (из работы [7])

Значение параметра μ	**Два состояния**	**Четыре состояния**	**Восемь состояний**	**16 состояний**
0.01	2.74	2.93	2.83	2.89
0.02	7.66	8.38	7.95	6.98
0.03	14.46	13.81	13.23	11.93
0.04	19.11	18.68	17.47	15.10

Таблица 5. Результаты вычислительного эксперимента для решения задачи «Умный муравей–3» с помощью метода представления автоматов полными таблицами переходов (из работы [7])

Значение параметра μ	**Два состояния**	**Четыре состояния**	**Восемь состояний**	**16 состояний**
0.01	2.68	3.49	3.74	3.78
0.02	6.12	7.32	7.24	7.28
0.03	12.48	12.17	11.72	11.15
0.04	17.18	15.94	15.03	13.68

2.4. Предлагаемый метод

В настоящей работе для решения задачи «Умный муравей–3» также предлагается использовать генетическое программирование, но при другом методе представления автоматов в виде особей генетического алгоритма.

Предлагаемый метод является развитием метода представления автоматов с помощью деревьев решений, который описан в работах [6, 7]. При его применении каждому состоянию автомата соответствует некоторое дерево решений, в листьях которого хранится информация о том, в какое состояние должен перейти автомат и какое выходное воздействие он должен выработать.

В настоящей работе предлагается вместо деревьев решений применять *конечные распознаватели* (конечные автоматы, использующиеся для распознавания регулярных языков). Пусть конечный автомат, который необходимо представить, имеет n входных логических переменных. Если их значения в некоторый момент расположить в некотором фиксированном порядке, то получится слово из нулей и единиц длины n.

Если подать это слово на вход некоторому конечному распознавателю, то он перейдет из начального состояния в некоторое состояние S. Обычно каждое из состояний является либо допускающим, либо не допускающим. В предлагаемом методе состояния конечного распознавателя помечены выходными воздействиями и номерами состояний управляющего конечного автомата.

Состояние S, в которое перешел конечный распознаватель, будет аналогом листа дерева решений – из него будет считана информация о том, в какое состояние должен перейти управляющий автомат и какое выходное воздействие он должен выработать.

Достоинством конечных распознавателей является то, что для представления одной и той же функции переходов, им требуется меньшее число состояний, чем деревьям решений. Поясним это на примере. Пусть из состояния 0 некоторого управляющего автомата заданы три перехода (рис. 20).

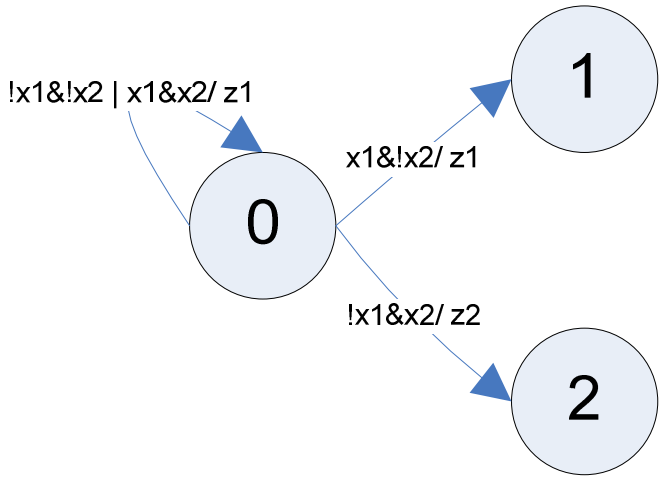

Рис. 20. Переходы из состояния 0 управляющего автомата

Пометки на переходах на рис. 20 имеют следующий формат: `условие/ выходное воздействие`. Отметим, что указанная система условий на переходах из рассматриваемого состояния является полной и непротиворечивой.

Если использовать представление автоматов с помощью деревьев решений, то дерево для рассматриваемого состояния будет содержать пять вершин (рис. 21).

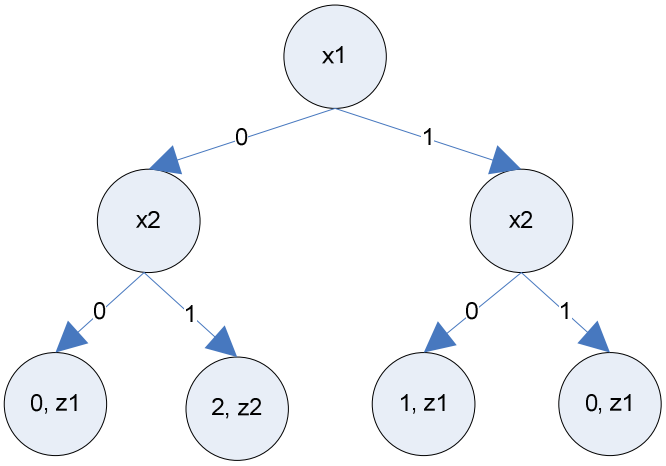

Рис. 21. Дерево решений, задающее переходы из состояния 0

Эту же систему переходов можно задать конечным распознавателем, который будет содержать всего три состояния. Его граф переходов изображен на

рис. 6. На вход этого автомата сначала подается значение переменной *x1*, а затем – значение переменной *x2*.

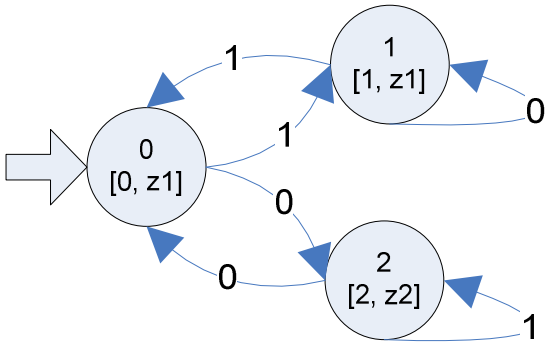

Рис. 22. Конечный распознаватель, задающий для управляющего автомата переходы из состояния 0

На рис. 22 начальное состояние указано большой стрелкой, а пометки состояний имеют следующий формат: номер состояния [номер состояния, в которое должен перейти управляющий автомат, выходное воздействие, которое он должен выработать].

2.4.1. Описание алгоритма

Приведем описание разработанного алгоритма генерации управляющих автоматов. Алгоритм генетического программирования состоит из пяти частей:

- создание начального поколения;
- мутация;
- скрещивание (кроссовер);
- отбор особей для формирования следующего поколения;
- вычисление функции приспособленности (фитнес-функции).

Начальное поколение состоит из фиксированного числа случайно сгенерированных автоматов. Все автоматы в поколении имеют одинаковое заранее заданное число состояний. При этом все конечные распознаватели, задающие переходы автоматов, также имеют одинаковое заранее заданное число состояний.

2.4.2. Операция мутации

Для выполнения мутации выбирается один из двух равновероятных вариантов:

- изменение начального состояния управляющего автомата на выбранное случайно;
- мутация соответствующего одному из состояний конечного распознавателя, задающего функцию переходов.

При мутации конечного распознавателя случайно выбирается один из следующих равновероятных вариантов:

- изменение начального состояния – в этом случае новое начальное состояние выбирается случайно и равновероятно;
- изменение состояния, в которое ведет переход, – случайно и равновероятно выбирается переход. После этого состояние, в которое ведет переход, заменяется на случайно выбранное состояние;
- изменение условия на переходе – случайно и равновероятно выбирается состояние. После этого переходы из этого состояния, соответствующие символам 0 и 1, меняются местами;
- изменение пометки одного из состояний – в этом случае равновероятно выбирается один из двух вариантов:
    - изменяется на случайно выбранное состояние, в которое должен перейти управляющий автомат;
    - изменяется выходное воздействие, которое он должен выработать.

2.4.3. Операция скрещивания

Скрещивание описаний управляющих конечных автоматов производится отдельно для каждого из состояний. При этом производится скрещивание

конечных распознавателей, задающих функции переходов. Опишем метод их скрещивания.

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

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

Опишем, как устроены переходы конечных распознавателей, которые получаются в результате скрещивания. Обозначим переход из состояния номер *i* в преобразователе *P1* по символу «1» как *P1(i, 1)*, а по символу «0» как *P1(i, 0)*. Аналогичный смысл придадим обозначениям *P2(i, 0)* и *P2(i, 1)*. Тогда для переходов из состояния с номером *i* в «потомках» *S1* и *S2* будет справедливо одно из четырех соотношений:

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

Все четыре варианта равновероятны. Возможные варианты переходов при графически изображены на рис. 23.

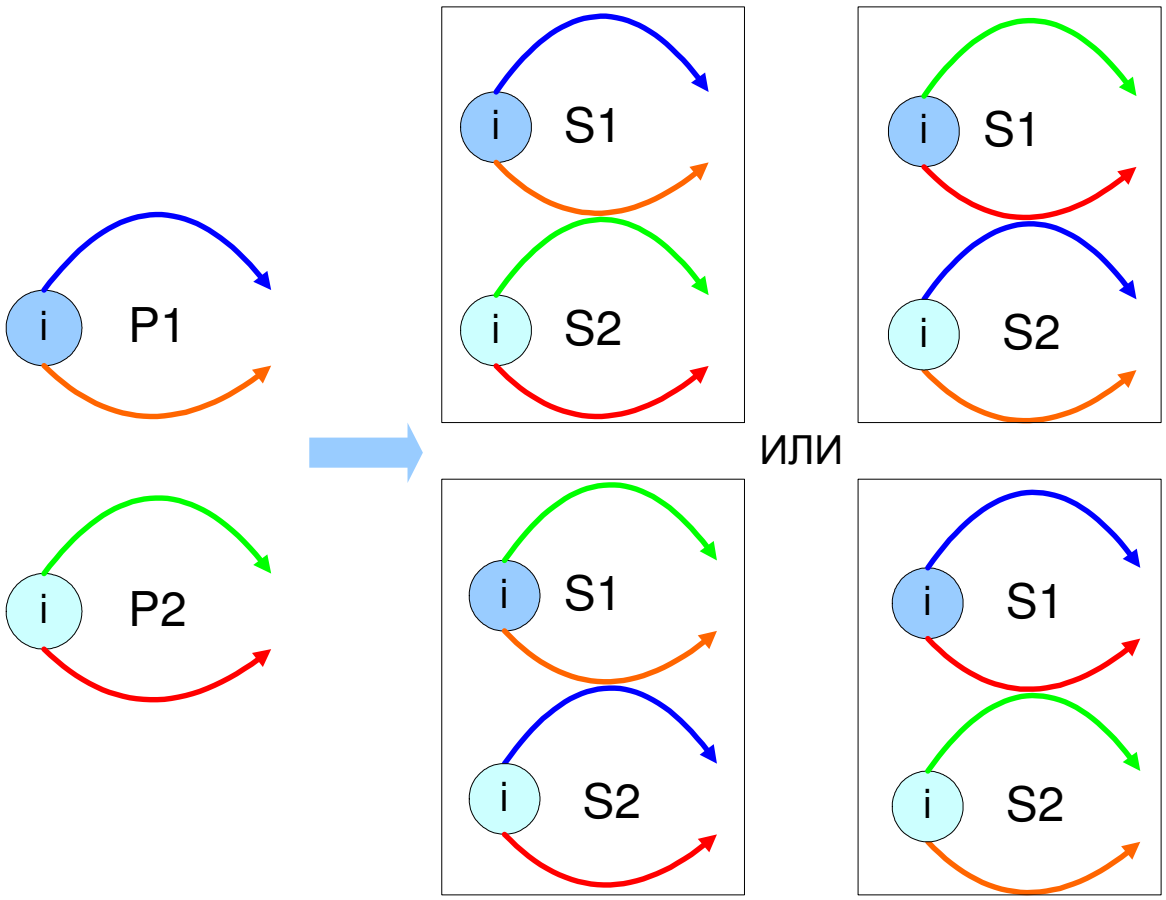

Рис. 23. Возможные варианты переходов при скрещивании

2.4.4. Формирование следующего поколения

В качестве основной стратегии формирования следующего поколения используется элитизм [4]. При обработке текущего поколения отбрасываются все особи, кроме нескольких наиболее приспособленных. Доля выживающих особей постоянна для каждого поколения и является одним из параметров алгоритма.

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

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

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

2.4.5. Настраиваемые параметры алгоритма

Следующие параметры алгоритма генетического программирования могут быть изменены:

- размер поколения;
- доля особей, переходящих в следующее поколение;
- число состояний в управляющем автомате;
- число состояний в конечных распознавателях, задающих функцию переходов;
- вероятность мутации;
- время до «малой» мутации поколения;
- время до «большой» мутации поколения.

Программная реализация описанного алгоритма генетического программирования выполнена на языке программирования *Java.*

2.5. Результаты вычислительных экспериментов

Вычислительные эксперименты проводились при следующих параметрах алгоритма генетического программирования:

- размер поколения – 200 особей;
- доля особей, переходящих в следующее поколение напрямую, составляла 10 %;

- вероятность применения операции мутации – 10 %;
- работа алгоритма останавливалась после генерации 200 поколений.

Результаты вычислительных экспериментов для различных значений параметра μ и различного числа состояний приведены в табл. 6.

Таблица 6. Результаты вычислительного эксперимента для решения задачи «Умный муравей–3» с помощью предлагаемого метода

Значение параметра μ	**Два состояния**	**Четыре состояния**	**Восемь состояний**	**16 состояний**
0.01	2.8565	2.8505	2.9525	3.5915
0.02	6.917	8.508	8.3765	8.075
0.03	13.637	13.452	13.636	14.1975
0.04	18.4915	19.804	19.2815	20.7895

Анализ полученных результатов показывает, что, в целом, предложенный метод на основе нового представления автоматов в качестве хромосом работает примерно так же эффективно, как и метод представления автоматов деревьями решений. Его преимущество на некоторых значениях параметров можно объяснить тем, что применение конечных распознавателей обеспечивает более компактное представление функции переходов. Это позволяет ускорить процесс поиска автомата, достаточно хорошо решающего задачу «Умный муравей–3». Отметим также, что при $\mu=0.04$ имеет место следующая ситуация – «жадный» алгоритм (разд. 2.2) показывает лучший результат по сравнению с автоматом, построенным с помощью метода деревьев решений. В то же время, автомат, построенный с помощью предлагаемого метода, работает лучше, чем «жадный» алгоритм.

Выводы по главе 2

1. Предложен метод представления функции переходов управляющих автоматов с помощью конечных распознавателей.
2. Для предложенного метода разработаны операции мутации и скрещивания.
3. Эффективность разработанного метода представления функции переходов продемонстрирована на примере построения автомата для задачи «Умный муравей–3».

ГЛАВА 3. МЕТОД ПОСТРОЕНИЕ КОНЕЧНЫХ АВТОМАТОВ УПРАВЛЕНИЯ СИСТЕМАМИ СО СЛОЖНЫМ ПОВЕДЕНИЕМ НА ОСНОВЕ ТЕСТОВ

В настоящей главе описывается метод построения конечных автоматов управления системами со сложным поведением на основе тестов.

3.1. 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 [11].

Другой подход, который практически не применяется для построения автоматных программ, но достаточно широко распространен при разработке традиционных программ, состоит в их разработке на основе тестов (*test-driven development*) [2]. При применении этого метода процесс написания кода на языке программирования идет параллельно с написанием тестов для программы. При этом добавление функциональности в программу осуществляется 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. 1.3.4), для решения которой успешно применяются генетические алгоритмы. Кроме этого, как говорилось раньше, 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.

3.2. 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» – 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.

3.2.1. 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 из состояний хранится список переходов. Каждый переход описывается событием, при поступлении которого этот переход выполняется и числом выходных воздействий, которые должны быть сгенерированы при выборе этого перехода.

Таким образом, в особи кодируется только «скелет» управляющего конечного автомата, а конкретные выходные воздействия, вырабатываемые на переходах, определяются с помощью алгоритма расстановки пометок, который аналогичен предложенному в работе [53].

Выбор представления графа переходов автомата с помощью списков ребер (в отличие от работы [52], 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.

3.2.1.1. Обработка входных переменных

Отметим, что в общем случае функция переходов и функция выходов (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* [5] переходы обычно помечаются не только событиями, но и так

называемыми «охранными условиями» (*guard conditions*) – логическими формулами, которые задают ограничения на значения входных переменных.

3.2.1.2. Алгоритм расстановки пометок

Идея алгоритма расстановки пометок состоит в том, что генетическим алгоритмом 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 [53] аналогичный принцип использовался при построении конечных автоматов для распознавания регулярных языков. Основное 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 конечного автомата последовательность событий, соответствующую одному из тестов, и будем наблюдать, за тем, какие переходы выполнит 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.

Пусть, например, 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. 24).

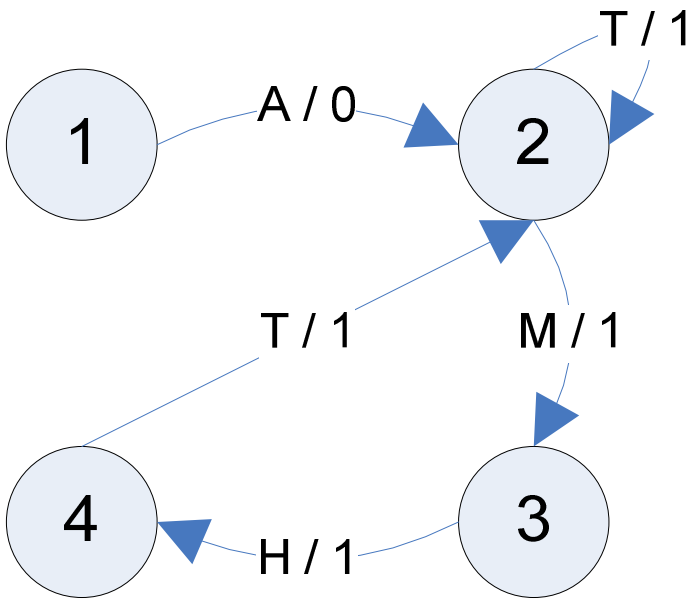

Рис. 24. Некоторые переходы «скелета» автомата

На рис. 24 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 четверное должно вырабатываться *z5*, а на переходе по событию *T* из четвертого состояния во второе – *z3* (рис. 25).

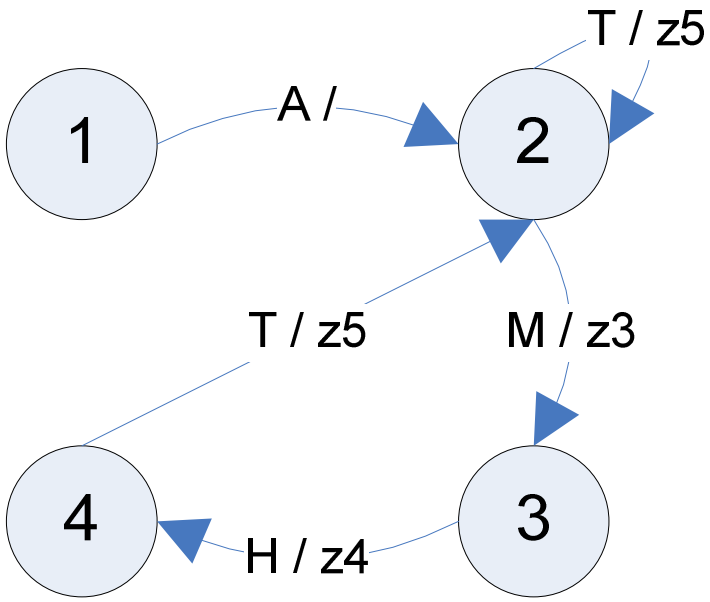

Рис. 25. Помеченные на основании теста переходы автомата

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 (имеет место 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*. Далее, каждый переход помечается той последовательностью zs_0, для которой величина $C[T][zs_0]$ максимальна.

3.2.2. 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 функция:

$$\mathrm{FF}_1 = \frac{\sum_{i=1}^{n}\left(1 - \frac{\mathrm{ED}(\mathrm{Output}[i], \mathrm{Answer}[i])}{\max(|\mathrm{Output}[i]|, |\mathrm{Answer}[i]|)}\right)}{n}.$$

Отметим, что значения этой функции лежат в пределах от 0 до 1.

Функции приспособленности должна учитывать, что чем «лучше» автомат соответствует тестам, тем больше ее значение. Она должна зависеть не только от того, насколько «хорошо» автомат работает на тестах, но и от числа переходов, которые он содержит. Функцию приспособленности предлагается вычислять 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 *cnt* – число переходов в автомате. Эта функция 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» автомата, 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 это расстояние взять из вычисленной таблицы динамического программирования.

3.2.3. Операция мутации

При выполнении операции мутации с заданной вероятностью (по умолчанию, она равна 0.05) осуществляется 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 заменяется на случайно выбранное;
- изменение события, по которому выполняется этот переход, – оно заменяется 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 в автомате может возникнуть ситуация, когда в автомате из одного состояния присутствуют два перехода по донному и тому же событию. Для устранения таких переходов применяется операция удаления дублирующихся переходов, описанная в следующем разделе.

3.2.3.1. Операция удаления дублирующихся переходов

Для удаления таких переходов для каждого состояния выполняются 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 переход *T* удаляется из списка.

3.2.4. Операция скрещивания

Скрещивание описаний автоматов производится следующим образом. Обозначим *P1* и *P2* «родительские» особи, а *S1* и *S2* – особи-«потомки». Для начальных состояний *S1.is* и *S2.is* автоматов *S1* и *S2* будет верно одно из 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 *i* автомата *P1* как *P1.T[i]*, а список переходов из состояния *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 переходы, выполняемые при обработке 10% тестов, для которых минимально *нормированное редакционное расстояние* $\frac{\mathrm{ED}(\mathrm{Output}[i], \mathrm{Answer}[i])}{\max(|\mathrm{Output}[i]|, |\mathrm{Answer}[i]|)}$ между «правильным ответом» *Answer* и последовательностью *Output* выходных воздействий, генерируемой автоматом. Здесь *ED(A, B)* – редакционное расстояние между строками *A* и *B*.
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.

3.2.5. 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.

3.2.6. Генетический алгоритм

Генетический алгоритм, применяемый для построения управляющего конечного автомата, аналогичен описанному в разд. 2.4.1. Начальное поколение также генерируется случайным образом, при этом все конечные автоматы содержат одинаковое число состояний. При генерации очередного поколения, как и ранее, применяется метод элитизма – фиксированная доля особей с наибольшими значениями функции приспособленности напрямую переходит в следующее поколение.

Программная реализация описанного алгоритма генетического программирования выполнена на языке программирования *Java*.

3.3. Пример применения метода

В настоящем разделе описывается применение предлагаемого метода для построения конечного автомата управления часами с будильником [17]. Эти часы имеют три кнопки (рис. 26), которые предназначены для изменения режима их работы и для настройки текущего времени или времени срабатывания будильника.

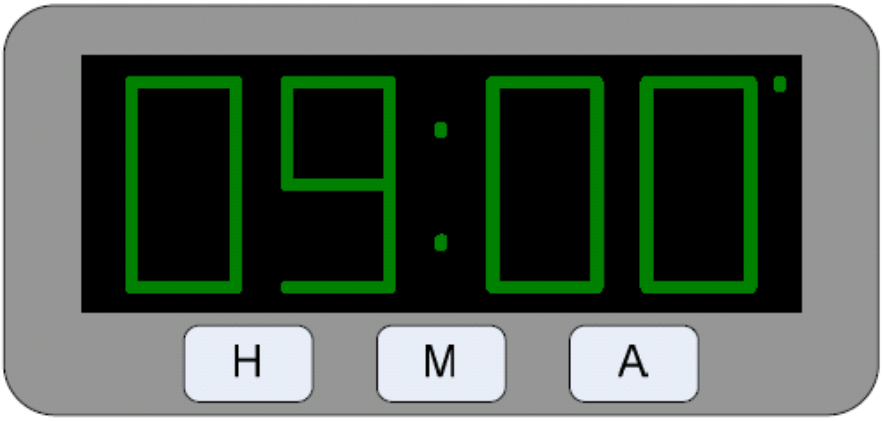

Рис. 26. Внешний вид часов с будильником

Если будильник выключен, то кнопки «H» и «M» используются, соответственно, для увеличения на единицу числа часов и минут в текущем времени. Кнопка «A» в этом режиме применяется для перехода в режим настройки времени срабатывания будильника. В этом режиме кнопки «H» и «M» служат 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 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 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 (рис. 27) конечного автомата, который построен вручную и приведен в работе [17].

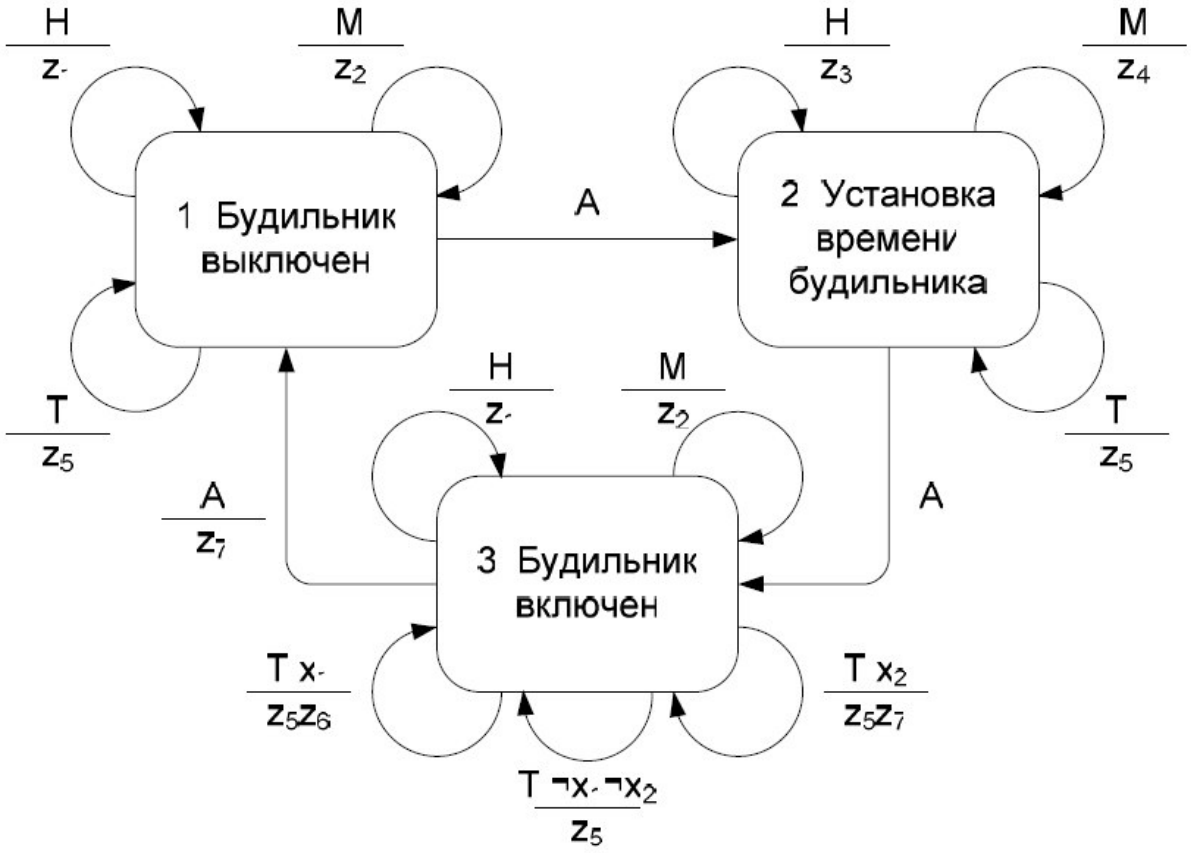

Рис. 27. Граф переходов конечного автомата управления часами с будильником из работы [17]

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

3.3.1. Система тестов

В систему тестов для построения автомата управления часами с будильником включены тесты, описывающие его работу во всех трех состояниях (режимах). Тесты, описывающие поведение часов с будильником в состоянии «Будильник выключен», приведены в табл. **Error! Reference source not found.**.

Таблица 7. 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 состоянии «Будильник выключен». Аналог третьего теста.

Тесты, описывающие поведение часов с будильником, в состоянии «Установка времени будильника» приведены в табл. 8.

Таблица 8. 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 «А» часы должны находиться в состоянии «Установка времени будильника». Описывается обработка нескольких нажатий кнопки «Н».
Input: *A, A, A, A, M* Answer: *z7, z4*	После четырех нажатий кнопки «А» часы должны находиться в состоянии «Установка времени будильника». Описывается обработка нажатия кнопки «М».
Input: *A, A, A, A, M, M, M, M* Answer: *z7, z4, z4, z4, z4*	После четырех нажатий кнопки «А» часы должны находиться в состоянии «Установка времени будильника». Описывается обработка нескольких нажатий кнопки «М».

Тесты, описывающие поведение часов с будильником в состоянии «Будильник включен», приведены в табл. 9.

Таблица 9. Тесты для состояния «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 состоянии «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. 10.

Таблица 10. 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 состояниях: «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 состояниях: «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 состояниях: «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 состояниях: «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 режимов работы, то система тестов должна содержать тесты для каждого из этих режимов и для переходов между ними. Во-вторых, тесты не должны противоречить друг другу – если входная

последовательность в одном из тестов является префиксом входной последовательности в другом тесте, то и выходная последовательность из первого теста должна быть префиксом выходной последовательности из второго теста. В-третьих, набор тестов в целом должен содержать все существующие в системе входные события и выходные воздействия.

3.3.2. 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.

Цель состояла в том, чтобы построить автомат, содержащий 14 переходов и соответствующий всем тестам (значение функции приспособленности, соответствующее такому автомату – 20.86). В результате работы алгоритма генетического программирования был построен автомат (рис. 28), в котором из начального (отмечено «жирной» рамкой) достижимы только три состояния из четырех. Если удалить недостижимое состояние, то этот граф переходов будет изоморфен построенному вручную.

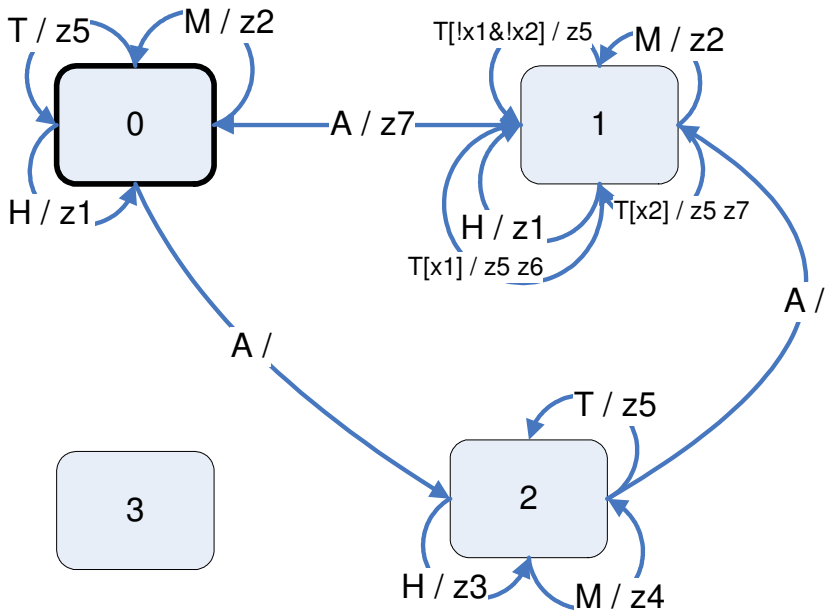

Рис. 28. Граф переходов автомата, построенного с помощью алгоритма генетического программирования

График зависимости максимального значения функции приспособленности от номера поколения приведен на рис. 29.

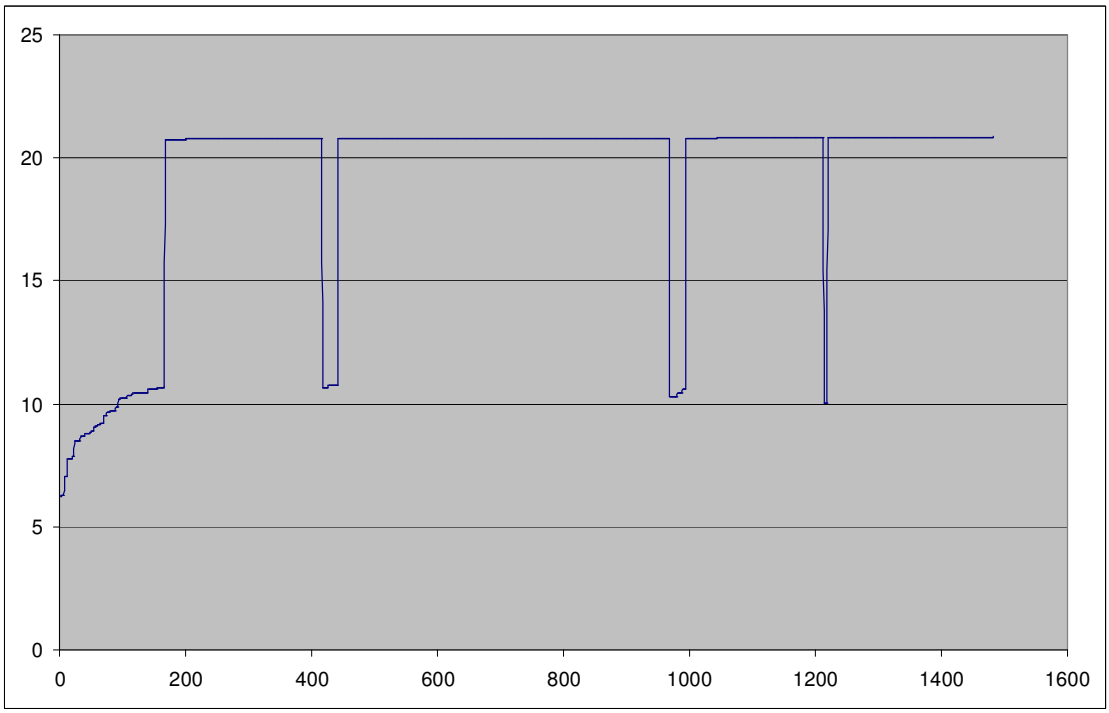

Рис. 29. Зависимость максимального значения функции приспособленности от номера поколения

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 уменьшалось число переходов, в процессе которого три раза (в районе 400-го, 1000-го и 1200-го поколений) к популяции применялась операция «большой мутации». В результате значение функции приспособленности уменьшалось до величины близкой к десяти. В результате в 1482-ом поколении был построен автомат, полностью проходящий все тесты и содержащий 14 переходов.

В процессе построения этого автомата функция приспособленности была вычислена 2674290 раз, а время работы алгоритма составило 530 с на компьютере с процессором *Intel Core 2 Duo T7300*.

Выводы по главе 3

1. Предложен новый метод совместного применения генетического и автоматного программирования – метод построения конечных автоматов на основе тестов.
2. Предложен алгоритм генетического программирования, осуществляющий построение автомата управления системой со сложным поведением на основе построения его структуры и алгоритма расстановки пометок.
3. Предложен метод скрещивания описаний конечных автоматов, учитывающий поведение автоматов на тестах.
4. Приведен пример применения предложенного метода и алгоритма генетического программирования для построения конечного автомата управления часами с будильником.

ЗАКЛЮЧЕНИЕ

В работе предложен новый метод представления управляющего конечного автомата – метод представления с помощью конечных распознавателей. Этот метод может применяться в контексте традиционного метода совместного применения генетического и автоматного программирования, который основан на вычислении функции приспособленности с помощью моделирования работы системы со сложным поведением. Выполнено сравнение предлагаемого метода с методом представления автоматов деревьями решений на примере задачи «Умный муравей–3». Результаты, которые показывают автоматы, построенные с помощью обоих методов, для большинства значений параметров задачи оказываются примерно одинаковыми. Однако, при $\mu=0.04$ автомат, построенный с помощью метода деревьев решений, проигрывает «жадному» алгоритму, а автомат, построенный с помощью предлагаемого метода, выигрывает у него.

В работе также предложен новый метод совместного применения генетического и автоматного программирования – метод генерации автоматов на основе тестов. При его использовании не требуется для каждой задачи «с нуля» программировать вычисление функции приспособленности – необходимо лишь задать новый набор тестов. «Ядро» функции приспособленности, основанное на вычислении редакционного расстояния между эталонной выходной последовательностью и сгенерированной автоматом, для всех задач одинаково. Предложена структура хромосомы алгоритма генетического программирования, метод скрещивания, учитывающий тесты, и алгоритм расстановки пометок. Проведена апробация предлагаемого метода на примере задачи построения автомата управления часами с будильником.

Сформулируем направления дальнейшего исследования по применению генетических алгоритмов для построения конечных автоматов управления системами со сложным поведением:

- совместное применение метода построения автоматов на основе тестов и метода сокращенных таблиц переходов;
- совместное применение метода построения автоматов на основе тестов и метода представления автоматов деревьями решений;
- изучение возможности применения методов верификации на моделях (*Model checking*) [8] при вычислении функции приспособленности.

По теме работы сделан доклад на VI межвузовской конференции молодых ученых (14–17 апреля 2009 года, СПбГУ ИТМО) и на научно-практической конференции студентов, аспирантов, молодых ученых и специалистов «Интегрированные модели, мягкие вычисления, вероятностные системы и комплексы программ в искусственном интеллекте» (ИММВИИ-2009, 26–27 мая 2009 года, г. Коломна). Кроме этого, материалы докладов по теме диссертации приняты к публикации в трудах Международной научной конференции «Компьютерные науки и информационные технологии» (Саратовский государственный университет имени Н. Г. Чернышевского, 1–4 июля 2009 года, Саратов).

Таким образом, задачи диссертационного исследования полностью решены, цель достигнута, полученные результаты опубликованы и прошли апробацию на научных конференциях по соответствующей тематике.

ИСТОЧНИКИ

1. *Бедный Ю. Д., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов в задаче «Умный муравей». СПбГУ ИТМО. 2007. http://is.ifmo.ru/works/ant/
2. *Бек К.* Экстремальное программирование: разработка через тестирование. СПб: Питер, 2003.
3. *Воронин О., Дьюдни А.* Дарвинизм в программировании //Мой компьютер. 2004. № 35. http://www.mycomp.kiev.ua/text/7458
4. *Гладков Л. А., Курейчик В. В., Курейчик В. М.* Генетические алгоритмы. М.: Физматлит. 2006.
5. *Гуров В. С., Мазим М. А., Нарвский А. С., Шалыто А. А.* UML. SWITCH-технология. Eclipse // Информационно-управляющие системы. 2004. № 6. с. 12–17.
6. *Данилов В. Р.* Метод представления автоматов деревьями решений для использования в генетическом программировании //Научно-технический вестник СПбГУ ИТМО. Выпуск 53. Автоматное программирование, с. 103–108.
7. *Данилов В. Р.* Технология генетического программирования для генерации автоматов управления системами со сложным поведением. СПбГУ ИТМО. Бакалаврская работа. 2007.
8. *Кларк Э., Грамберг О., Пелед Д.* Верификация моделей программ: Model Checking. М.: МЦНМО. 2002.
9. *Левенштейн В. И.* Двоичные коды с исправлением выпадений, вставок и замещений символов //Доклады Академии наук СССР. 1963. № 4, с. 845–848.
10. *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

11. *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/
12. *Непейвода Н. Н.* Стили и методы программирования. М.: Интернет-университет информационных технологий, 2005.
13. *Николенко С. И.* Лекции по генетическим алгоритмам. http://logic.pdmi.ras.ru/~sergey/teaching/ml/
14. *Николенко С. И., Тулупьев А. Л.* Самообучающиеся системы. М.: МЦНМО, 2009.
15. *Норенков И. П., Арутюнян Н. М.* Метагенетический алгоритм оптимизации и структурного синтеза проектных решений //Информационные технологии. 2007. № 3.
16. *Поликарпова Н. И., Точилин В. Н., Шалыто А. А.* Применение генетического программирования для генерации автоматов с большим числом входных переменных //Научно-технический вестник СПбГУ ИТМО. Выпуск 53. Автоматное программирование, с. 24–42.
17. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб: Питер, 2009.
18. *Промежуточный отчет* по теме «Технология генетического программирования для генерации автоматов управления системами со сложным поведением». Этап III. «Экспериментальные исследования поставленных перед НИР задач». http://is.ifmo.ru/genalg/_2007_03_report-genetic.pdf
19. *Промежуточный отчет* по теме «Технология генетического программирования для генерации автоматов управления системами со

сложным поведением». Этап IV. «Обобщение и оценка результатов исследований». http://is.ifmo.ru/genalg/_2007_04_report-genetic.pdf

20. *Рассел С., Норвиг П.* Искусственный интеллект. Современный подход. М.: Вильямс. 2006.
21. Сайт по генетическому программированию. www.genetic-programming.com
22. *Туккель Н. И., Шалыто А. А.* Система управления дизель-генератором (фрагмент). Программирование с явным выделением состояний. Проектная документация. http://is.ifmo.ru/projects/dg/
23. *Хопкрофт Дж., Мотвани Р., Ульман Дж.* Введение в теорию автоматов, языков и вычислений. М.: Вильямс, 2002.
24. *Царев Ф. Н.* Разработка метода совместного применения генетического программирования и конечных автоматов на примере игры «Летающие тарелки». СПбГУ ИТМО. Бакалаврская работа. 2007.
25. *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, с. 42–60.
26. *Царев Ф. Н., Шалыто А. А.* Применение генетических алгоритмов для построения автоматов с минимальным числом состояний для задачи об «Умном муравье» /Труды научно-технической конференции «Научное программное обеспечение в образовании и научных исследованиях». СПб.: СПбГПУ. 2008, с. 209–215.
27. *Шалыто А. А.* Switch-технология. Алгоритмизация и программирование задач логического управления СПб.: Наука, 1998.
28. *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

29. *Armstrong, D.* A programmed algorithm for assigning internal codes to sequential machines // IRE Transactions on Electronic Computers. EC.11. 1962. I.4, pp.466–472.
30. *Axelrod R.* The Evolution of Cooperation. NY: Basic Books, 1984.
31. *Axelrod R.* The Evolution of Strategies in the Iterated Prisoner's Dilemma / In Genetic Algorithms and Simulated Annealing. London: Pitman, 1987. pp. 32–41. http://www-personal.umich.edu/~axe/research_papers.html
32. *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
33. *Benson K.* Evolving Finite State Machines with Embedded Genetic Programming for Automatic Target Detection. http://ieeexplore.ieee.org/iel5/6997/18853/00870838.pdf
34. *Brave S.* Evolving Deterministic Finite Automata Using Cellular Encoding / Proceeding of First Annual Conference (Genetic Programming-1996), pp. 39–44. http://citeseer.ist.psu.edu/131538.html
35. *Chambers L.* Practical Handbook of Genetic Algorithms. Complex Coding Systems. Volumes I. II, III. CRC Press, 1999.
36. *Das R., Mitchell M., Crutchfield J.P.* A genetic algorithm discovers particle-based computation in cellular automata // Lecture Notes in Computer Science. 1994. www.santafe.edu/research/publications/workingpapers/94-03-015.pdf
37. *Fogel D.* The Evolution of Intelligent Decision Making in Gaming // Cybernetics and Systems. 2001. 22, pp. 223–236.
38. *Fogel L.* Autonomous Automata // Industrial Research. 1962. V.4, pp. 14–19.
39. *Fogel L., Owens A., Walsh M.* Artificial Intelligence through Simulated Evolution. NY: Wiley. 1966.

40. *Ghnemat R., Khatatneh K., Oqeili S., Bertelle C., Duchamp G.* Automata-based adaptive behavior for economic modeling using game theory. 2005. http://arxiv.org/abs/cs/0510089
41. *Gold E. M.* Complexity of automaton identification from given data. // Information and Control, 1978, № 37, pp. 302–320.
42. *Goldberg D.* A Note on Boltzmann Tournament Selection for Genetic Algorithms and Population-Oriented Simulated Annealing // Complex Systems. 1990. V. 4, I. 4, pp. 445–460.
43. *Hamming R.* Error detecting and error correcting codes //Bell System Technical Journal 29 (2), pp. 147–160.
44. *Hillis W.* Co-Evolving Parasites Improve Simulated Evolution as an Optimization Procedure / In Artificial Life II. MA: Addison-Wesley, 1992. pp. 313–324.
45. *Holland J.* Adaptation in Natural and Artificial Systems. University of Michigan Press, 1975.
46. *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
47. *Juillie H., Pollack J.* Coevolving the «Ideal» Trainer: Application to the Discovery of Cellular Automata Rules. 1998. http://citeseer.ist.psu.edu/16712.html
48. *Koza J.* Genetic Evolution and Co-Evolution of Computer Programs / Proceedings of Second Conference on Artificial Life. Redwood City, CA: Addison-Wesley. 1992, pp. 603–629. http://citeseer.ist.psu.edu/177879.html
49. *Koza J.* Genetic programming. On the Programming of Computers by Means of Natural Selection. MA: The MIT Press, 1998.
50. *Levy S.* Artificial Life: The Quest for a New Creation. New York: Pantheon, 1992.

51. *Lucas S., Reynolds T.* Learning DFA: Evolution versus Evidence Driven State Merging.
52. *Lucas S., Reynolds T.* Learning Finite State Transducers: Evolution versus Heuristic State Merging // IEEE Transactions on Evolutionary Computation. Volume 11. 2007. Issue 3, June, pp. 308–325.
53. *Lucas S., Reynolds T.* Learning Deterministic Finite Automata with a Smart State Labelling Evolutionary Algorithm // IEEE Transactions on Evolutionary Computation. Volume 27. 2005. Issue 7, June.
54. *Miller J.* The Coevolution of Automata in the Repeated Prisoner's Dilemma. Working Paper. Santa Fe Institute. 1989 // Journal of Economic Behavior & Organization. 1996. V. 29. I. 1, pp. 87–112.
55. *Mitchell M.* An Introduction to Genetic Algorithms. MA: The MIT Press, 1996.
56. *Mitchell M., Crutchfield J. P., Hraber P. T.* Evolving cellular automata to perform computations // Physica D. 1993. vol. 75, pp. 361–391. http://web.cecs.pdx.edu/~mm/mech-imped.pdf
57. *Naidoo A., Pillay N.* The Induction of Finite Transducers Using Genetic Programming / Proceedings of Euro GP. Springer. 2007. http://saturn.cs.unp.ac.za/~nelishiap/papers/eurogp07.pdf
58. *Nedjah N., Mourelle L.* Mealy Finite State Machines: An Evolutionary Approach // International Journal of Innovative Computing, Information and Control. 2006. V.2, I. 4.
59. *Reynolds C. W.* Competition, Coevolution and the Game of Tag / Proceedings of Artificial Life IV. Cambridge. MA: MIT Press. 1994, pp. 59–69. http://www.red3d.com/cwr/papers/1994/alife4.html
60. *Tu M., Wolff E., Lamersdorf W.* Genetic Algorithms for Automated Negotiations: A FSM-based Application Approach /Proceedings of the 11-th International Conference on Database and Expert Systems. 2000. http://ieeexplore.ieee.org/iel5/7035/18943/00875153.pdf

61. *Wolff K., Nordin P.* An Evolutionary Based Approach for Control Programming of Humanoids / Proceedings of the 3rd International Conference on Humanoid Robots (Humanoids'03). Karlsruhe: VDI/VDE-GMA. 2003. http://citeseer.ist.psu.edu/641298.html