Министерство образования и науки Российской Федерации
Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Факультет Информационных технологий и программирования
Направление (специальность) прикладная математика и информатика
Квалификация (степень) бакалавр прикладной математики и информатики
Кафедра Компьютерных технологий **Группа** 4538

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Применение методов решения задачи о выполнимости булевой формулы для построения управляющих конечных автоматов по сценариям работы

Автор квалификационной работы Ульянцев В.И.
Руководитель Царев Ф.Н.

К ЗАЩИТЕ ДОПУСТИТЬ
Зав. кафедры Васильев В.Н.

Санкт-Петербург, 2011 г.

ОГЛАВЛЕНИЕ

Оглавление 2

Введение 4

1. Методы машинного обучения для построения конечных автоматов и методы решения задачи о выполнимости булевой формулы 5

1.1. Методы машинного обучения для построения конечных автоматов 5

1.1.1. Машинное обучение 5

1.1.2. Типы автоматных моделей 6

1.1.3. Построение конечных автоматов-распознавателей по обучающим примерам 9

1.1.3.1. Алгоритм *EDSM* 10

1.1.3.2. Эволюционные алгоритмы и их сравнение с алгоритмами, основанными на объединении состояний 12

1.1.3.3. Алгоритм, основанный на методах решения задачи о выполнимости булевой формулы 19

1.1.4. Построение конечных преобразователей по обучающим примерам 21

1.2. Программные средства для решения задачи о выполнимости булевой формулы и их применение 23

1.2.1. Программное средство *zChaff* 24

1.2.2. Программное средство *MiniSat* 25

1.2.3. Программное средство *CryptoMiniSat* 26

1.3. Постановка задачи 27

Выводы по главе 1 29

2. Теоретическая оценка сложности построения управляющего автомата по сценариям работы программы 30

2.1. Доказательство принадлежности классу NP поставленной задачи 30

2.2. Доказательство принадлежности классу NP-трудных задач ... 31

Выводы по главе 2 33

3. Предлагаемый метод машинного обучения 34

3.1. Построение дерева сценариев 34

3.2. Построение графа совместимости дерева сценариев 36

3.3. Построение булевой КНФ-формулы 39

3.4. Запуск сторонней программы, решающей задачу о выполнимости булевой КНФ-формулы, и построение автомата по найденному выполняющему набору значений переменных 41

Выводы по главе 3 43

4. Экспериментальное исследование 44

4.1. Исследование на задаче о построении автомата управления часами с будильником 44

4.2. Исследование на задачах, полученных случайным образом ... 44

Выводы по главе 4 46

Заключение 48

Источники 49

ВВЕДЕНИЕ

В последнее время все шире начинает применяться автоматное программирование, в рамках которого поведение программ описывается с помощью детерминированных конечных автоматов [4].

Для многих задач автоматы удается строить эвристически, однако существуют задачи, для которых такое построение автоматов затруднительно. К задачам этого класса относятся, в частности, задачи об «Умном муравье» [9, 12, 18] и об управлении моделью беспилотного летательного аппарата [7].

Для построения автоматов в задачах такого типа успешно применяются генетические алгоритмы [1], в том числе на основе обучающих примеров [6]. Недостатком генетических алгоритмов является то, что время их работы весьма велико, и его достаточно трудно оценить аналитически.

Целью настоящей работы является разработка метода машинного обучения для построения управляющих конечных автоматов, лишенного указанных недостатков.

1. МЕТОДЫ МАШИННОГО ОБУЧЕНИЯ ДЛЯ ПОСТРОЕНИЯ КОНЕЧНЫХ АВТОМАТОВ И МЕТОДЫ РЕШЕНИЯ ЗАДАЧИ О ВЫПОЛНИМОСТИ БУЛЕВОЙ ФОРМУЛЫ

1.1. Методы машинного обучения для построения конечных автоматов

В настоящем разделе приводится обзор существующих методов машинного обучения для построения конечных автоматов.

1.1.1. Машинное обучение

В соответствии с книгой [3] алгоритм является алгоритмом машинного обучения, если он улучшает свое поведение по мере накопления опыта. Это означает, что алгоритм настраивает параметры модели либо на заранее подготовленных тестовых примерах, либо на собственных ошибках, и со временем решает поставленную задачу все лучше и лучше. Некоторые алгоритмы машинного обучения способны подмечать ранее неизвестные закономерности в данных, выделять знания, которых раньше не было.

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

Как правило, в машинном обучении конечные автоматы рассматриваются как распознаватели регулярных языков или как преобразователи слов одного регулярного языка в слова другого. Кроме того, существующие алгоритмы машинного обучения для конечных

автоматов основаны на принципе обучения «на собственных ошибках» и не являются достаточно эффективными для ряда задач, так как не позволяют эффективно учитывать знания человека, и не позволяют строить автоматы с гарантированным поведением.

1.1.2. Типы автоматных моделей

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

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

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

Расширенная функция переходов $\hat{\delta}: X^* \times Y \to Y$, сопоставляющая новое состояние текущему состоянию и цепочке символов, определяется индуктивно следующим образом [8]:

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

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

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

Класс языков, распознаваемых ДКА, называют регулярными языками. Известно, что он совпадает с классом языков, описываемых регулярными выражениями и автоматными грамматиками [8].

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

$$f(\varepsilon) = \varepsilon$$

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

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

Понятие *управляющего* (или структурного) *автомата Мура* аналогично понятию абстрактного автомата Мура, введенному выше. В таком автомате выходное воздействие зависит только от состояния и не зависит от входного воздействия. На рис. 1 приведен пример графа переходов автомата Мура.

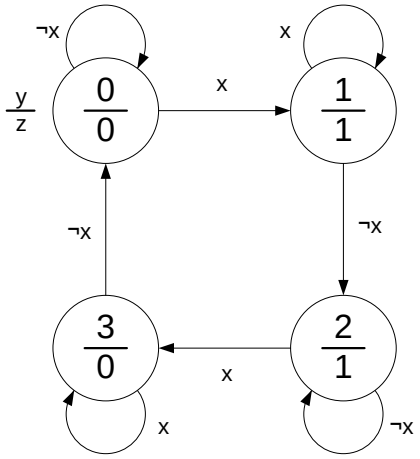

Рис. 1. Пример графа переходов автомата Мура

По аналогии с абстрактными автоматами, структурный автомат, выходные воздействия которого зависят не только от состояния, но и от входных воздействий, называется автоматом Мили. Известно [5], что для любого автомата Мили можно построить эквивалентный ему автомат Мура. Число состояний в таком автомате будет не меньше, чем в исходном.

В качестве примера рассмотрим последовательный двоичный одноразрядный сумматор, который выполняет сложение одноименных бит двух двоичных чисел с учетом переноса. Результатом работы сумматора является одноименный бит суммы и перенос в следующий разряд. На рис. 2 приведен автомат Мили, реализующий это «устройство».

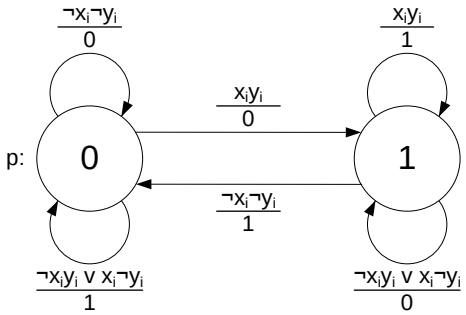

Рис. 2. Автомат Мили последовательного двоичного одноразрядного сумматора

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

Если часть выходных переменных автомата зависит только от состояний, а остальные – также и от входных воздействий, то удобно разделить функцию выходов φ на две составляющие: $\varphi_1(y)$ и $\varphi_2(x, y)$. В структурную схему автомата в этом случае вводится не один, как в автоматах Мура и Мили, а два выходных преобразователя. Такие автоматы называются смешанными, автоматами Мура-Мили или С-автоматами.

1.1.3. Построение конечных автоматов-распознавателей по обучающим примерам

Из теории формальных языков известно, что конечные детерминированные автоматы способны распознавать регулярные языки [8]. В связи с этим актуальна задача построения автомата, распознающего по множеству примеров некий язык. Задача может быть усилена до построения автомата с минимальным числом состояний. Однако, в работе [16] показано, что эта задача является *NP*-трудной.

В настоящем разделе приводится обзор алгоритмов машинного обучения для задачи построения автоматов-распознавателей по обучающим примерам.

1.1.3.1. Алгоритм *EDSM*

Алгоритм объединения состояний на основе свидетельств *EDSM* (*Evidence-Driven State Merging*) был предложен на соревновании *Abbadingo One* [20]. Опишем данный алгоритм.

На этапе инициализации строится префиксное дерево (APTA – augmented prefix tree acceptor) по набору слов, для каждого из которых известно, должно ли оно допускаться автоматом (множество слов S^+) или не должно (множество S^-). На рис. 3 [19] приведен пример префиксного дерева, построенного по входным данным $S^+ = \{a, abaa, bb\}$, $S^- = \{abb, b\}$. Заметим, что префиксное дерево также является частным случаем автомата-распознавателя.

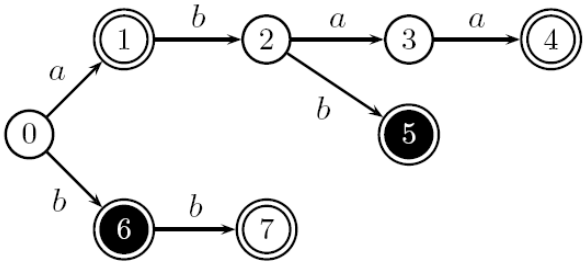

Рис. 3. Пример префиксного дерева

Затем на каждом шаге алгоритма рассматриваемый автомат обобщается с помощью слияния выбранной пары состояний. После слияния рассматриваемый детерминированный автомат A должен допускать слова из S^+ и не допускать слова из S^-. Алгоритм заканчивает свою работу, когда не найдется такой пары вершин, что после их слияния описанное свойство не нарушится.

Существует несколько реализаций данного алгоритма, отличающихся стратегией выбора пары вершин для слияния на каждом шаге. Опишем стратегию *Blue-Fringe* [20], которая была предложена на соревновании *Abbadingo One*, на котором алгоритм, использующий данную стратегию, занял первое место. Опишем данную стратегию.

На каждом шаге алгоритма состояния автомата разделены на три множества, каждому множеству условно соответствует цвет. Множеству

необработанных состояний соответствует белый цвет. Множество красных состояний – попарно не сливаемые состояния, которые являются частью результирующего автомата. Все состояния, не являющиеся красными, в которые ведут переходы из красных состояний, являются синими. Пример автомата после нескольких шагов работы алгоритма приведен на рис. 4 [19]. На рисунке символам «R» соответствуют красные состяния, символам «B» – синие, а непомеченным – белые.

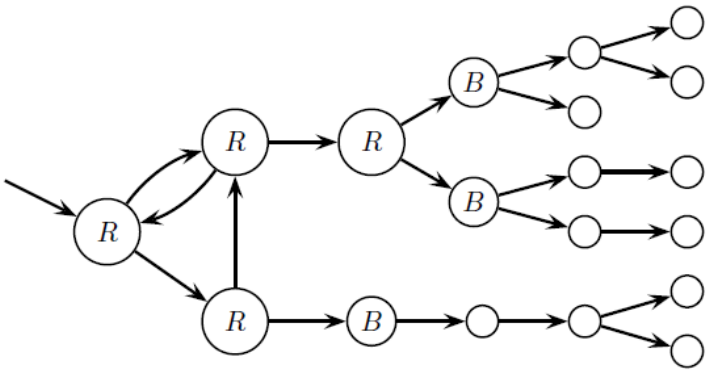

Рис. 4. Пример автомата после нескольких шагов работы *Blue-Fringe*

На каждом шаге алгоритма все синие состояния, которые нельзя слить ни с одним из красных, перекрашиваются в красный цвет. Затем выбирается пара состояний красного и синего цвета с наибольшим значением *функции, оценивающей слияния*. После этого производится слияние выбранной пары состояний. В процессе работы алгоритма производится перекраска в синий цвет белых состояний, которые стали потомками красных.

В предлагаемом алгоритме функция, оценивающая слияния, принимает на вход красное состояние r и синее состояние b. Функция возвращает суммарное число различных слов, которые одинаково допускает или не допускает автомат, если начнет процесс из состояний r и b. Если же находится слово, которое допускается при старте из r и не допускается при старте из b (или наоборот), то данные состояния сливать нельзя и функция, оценивающая слияния, должна вернуть $-\infty$.

1.1.3.2. Эволюционные алгоритмы и их сравнение с алгоритмами, основанными на объединении состояний

В работе [22] производится сравнение алгоритмов машинного обучения для построения конечных автоматов-распознавателей. При этом рассматриваются два типа алгоритмов: эволюционные и основанные на построении префиксного дерева (бора) и дальнейшем объединении состояний на основе свидетельств.

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

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

Для сокращения пространства поиска в работе [22] применяется специальный метод определения того, какие состояния автомата являются допускающими, а какие – нет. Для этого предлагается использовать так называемый «алгоритм расстановки пометок» на состояниях. Он состоит в следующем. Для каждой строки t из множества обучающих примеров на единицу увеличивается значение $h[f(t)][c]$, где $f(t)$ – номер состояния, в

которое приходит рассматриваемый конечный автомат после обработки строки t, а число c равно единице, если строка s должна допускаться автоматом, и равно нулю, если не должна допускаться автоматом. После этого те состояния s, для которых $h[s][1] > h[s][0]$, помечаются как допускающие, а все остальные – как недопускающие. За счет применения описанного алгоритма размер пространства поиска существенно сокращается – его размер становится равным n^{2n}.

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

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

Сравнение эволюционного алгоритма проводилась на двух наборах тестовых примеров. Первый набор тестовых данных предложен в работах [25] и [31]. На этом наборе сравнение проводилось не только с алгоритмом *EDSM,* но и с алгоритмом генетического программирования, предложенным в работе [25]. Это сравнение показывает, что разработанный алгоритм эволюционного программирования превосходит и алгоритм *EDSM,* и алгоритм генетического программирования, как по точности построенных автоматов, так и по времени работы. Например,

среднее время, которое требуется предложенному в рассматриваемой работе алгоритму для построения конечного автомата, составляет 1.6 миллисекунды, а алгоритму *EDSM* требуется 37 миллисекунд в среднем. Вычислительные эксперименты проводились на компьютере с процессором *Pentium IV* с тактовой частотой 2.4 ГГц. Алгоритмы реализовывались на языке программирования *Java*.

Второй набор тестовых данных строился следующим образом:

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

На этом тестовом наборе разработанный эволюционный алгоритм при n=4, 8, 16 превосходил алгоритм *EDSM*, но при n=32 ему уступал.

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

В работе [23] также рассматривается задача построения конечных автоматов-распознавателей с помощью эволюционных алгоритмов. Предлагаемый в рассматриваемой работе эволюционный алгоритм является улучшенной версией алгоритма, предложенного в работе [22].

Конечный распознаватель в рассматриваемой работе представляется так же, как и в работе [22] – в хромосому входит таблица переходов, а пометки на состояниях расставляются с помощью специального алгоритма. В качестве эволюционного алгоритма в работе [23] также используется (1+1)-эволюционная стратегия, но операцию мутации предлагается выполнять не так, как в работе [22].

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

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

Кроме описанного способа выполнения операции мутации в рассматриваемой работе также предлагается следующий способ – для каждого состояния запоминаются строки, которые оно должно принимать, и строки, которое оно не должно принимать. В соответствии с алгоритмом

расстановки пометок, состояние будет *допускающим*, если строк, которое оно должно допускать больше, чем строк, которое оно не должно допускать, и *недопускающим* – в противном случае. При выполнении операции мутации случайным образом выбирается строка, которую автомат классифицирует неправильно. После этого случайным образом выбирается один из переходов, которые используется при обработке выбранной строки. Изменению подвергается та ячейка таблицы переходов, которая ему соответствует.

Для сравнения алгоритмов машинного обучения в рассматриваемой работе применялось два набора тестовых данных. Первый строился так же, как второй набор тестовых данных в работе [22] (описание приведено выше). Второй набор тестовых данных был зашумлен – для десятой части строк из обучающего набора было неправильно указано, должны они допускаться автоматом или нет.

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

- метода, не использующего алгоритм расстановки пометок (plain);
- метода из работы [22] (*smart labeling*);
- метода, использующего первый из предлагаемых в настоящей работе подходов к выполнению операции мутации (*sampled*);
- метода, использующего второй из предлагаемых в настоящей работе подходов к выполнению операции мутации (*quick-sampled*).

Результаты вычислительных экспериментов показали, что при восьми состояниях автомата, на основе которого генерировались обучающие наборы, из 50 случаев алгоритм *plain* построил автомат в 26 случаях, алгоритм *smart labeling* – в 42 случаях, алгоритм *sampled* – в 47 случаях, а алгоритм *quick-sampled* – в 43 случаях. При использовании

автомата из 16 состояний результаты были такими: *plain* – пять из 50, smart – 21 из 50, *sampled* – 26 из 50, *quick-sampled* – четыре из 50. Если же в исходном автомате было 32 состояния, то только два алгоритма справились с задачей: *smart* правильно строил автомат в шести случаях из 50, а *sampled* – в 12 случаях из 50.

На втором наборе тестовых данных, который содержал зашумленные обучающие наборы, сравнение проводилось с алгоритмом *EDSM* и с другими алгоритмами, участвовавшими в соревновании *GECCO 2004* [21]. Результаты вычислительных экспериментов, выполненные в этой работе, показали, что предлагаемый в ней эволюционный алгоритм (*sampled*) превосходит все существующие методы машинного обучения конечных автоматов на этом обучающем наборе по точности построения автоматов. При этом отметим, что предложенный алгоритм эффективен только для построения автоматов из относительно небольшого числа состояний (порядка нескольких десятков), в то время как с помощью алгоритма *EDSM* могут быть успешно построены автоматы из сотен состояний.

В работе [10] автоматы представлялись с помощью таблицы переходов. Разработанный в этой работе метод позволяет поддерживать в популяции автоматы с разным числом состояний. Функция приспособленности учитывает три компонента – число верно распознанных тестовых примеров, число состояний и переходов автомата, степень общности языка, соответствующего построенному автомату. Это позволяет сузить область поиска, и находить языки, соответствующие определенным критериям. В обучающий набор могут входить примеры слов, которые как принадлежат, так и не принадлежат языку.

Приведем описание генетической операции – репродукция. Число состояний потомка выбирается случайно из диапазона $[N - 2, N + 2]$, где N – число состояний первого родителя. После этого каждый переход копируется из родителей, причем вероятность копирования перехода из

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

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

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

В работе [29] также рассматривается задача построения конечных автоматов-распознавателей. В качестве метода представления конечных автоматов выбраны двоичные строки, в качестве операций мутации и скрещивания используется однородное скрещивание и мутация.

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

Рассматривается три варианта функции приспособленности. Первый из них основан на строгом сравнении строк, второй – на вычислении редакционного расстояния, третий – на вычислении длины общего префикса строк. В качестве метода генерации следующего поколения применяется метод рулетки.

Алгоритмы, предлагаемые в рассматриваемой работе, реализованы в инструментальном средстве *GeSM*. В этом средстве кроме генетического

алгоритма также реализован алгоритм удаления недостижимых состояний из автоматов.

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

1.1.3.3. Алгоритм, основанный на методах решения задачи о выполнимости булевой формулы

В работе [19] представлен алгоритм построения автоматов-распознавателей по тестовым наборам, принципиально отличающийся от предлагаемых ранее алгоритмов для решения поставленной задачи. Отличие состоит в том, что данный алгоритм основан на сведении поставленной задачи *к задаче о выполнимости булевой формулы* (*Boolean satisfiability – SAT*).

Напомним, что входными данными для алгоритма построения конечного автомата-распознавателя A является набор слов, для каждого из которых известно, должно ли оно допускаться автоматом (множество слов S^+) или не должно (множество S^-). Подробней опишем этапы предлагаемого алгоритма.

Как и в алгоритме *EDSM*, первым этапом работы алгоритма является построение префиксного дерева. Вторым этапом – построение графа совместимости по полученному префиксному дереву. Множество вершин этого графа совпадает с множеством вершин префиксного дерева, и две вершины соединены ребром, если они не могут соответствовать одному состоянию искомого автомата-распознавателя. На рис. 5 [19] приведен пример графа совместимости, построенного по префиксному дереву, приведенному на рис. 3.

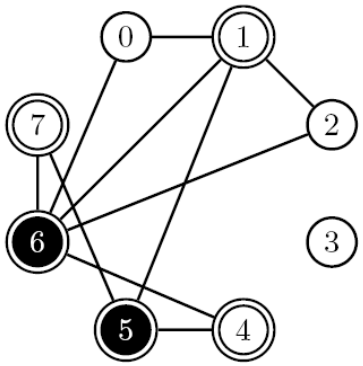

Рис. 5. Пример графа совместимости

Третий, самый сложный, этап состоит в построении и упрощении КНФ-формулы, кодирующей требования, накладываемые графом совместимости на автомат-распознаватель. В рассматриваемой работе сравниваются два способа построения КНФ-формулы. Первый из них имеет название *direct encoding*, он строит формулу, состоящую из $O(k^2|S|^2)$ скобок. Здесь k – размер искомого автомата, $|S|$ - суммарная длина обучающих примеров. Второй способ строит формулу, состоящую из $O(k^2|S|)$ скобок. Таким образом, в этой работе предложен способ построения КНФ-формулы, существенно превосходящий по качеству существующие методы. Также авторы этой работы предложили набор способов упрощения полученной формулы, используя методы нахождения полных подграфов в графе совместимости.

Полученная КНФ-формула подавалась на вход программе *picosat*, находящей выполняющую постановку для формулы. Затем, по найденной выполняющей подстановке строится искомый автомат-распознаватель.

Результаты экспериментальных исследований, выполненных авторами статьи, показали, что производительность предлагаемого метода выше всех ранее рассматриваемых методов построения автоматов-распознавателей.

На основании полученных в этой работе результатов сделан вывод, что разработка методов машинного обучения на основе методов решения задачи о выполнимости булевой формулы является перспективной областью.

1.1.4. Построение конечных преобразователей по обучающим примерам

По построению конечных преобразователей на основе обучающих примеров существует намного меньше работ, чем по построению конечных распознавателей. Одной из работ по конечным автоматам-распознавателям является работа [24]. Такие автоматы при обработке символов входного слова при выполнении переходов записывают в выходной поток символы выходного алфавита, тем самым осуществляя преобразование слов одного формального языка в слова другого.

Так же, как и в рассмотренных выше работах [22, 23], посвященных построению конечных автоматов-распознавателей, в работе [24] применяется (1+1)-эволюционная стратегия. Конечный преобразователь представляется в виде набора двух таблиц – таблицы значений функции переходов и таблицы значений функции выходов. При этом предполагалось, что на каждом из переходов конечный преобразователь может вывести не более одного символа.

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

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

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

Последние две мутации из перечисленных выбираются равновероятно.

В указанной работе рассматриваются три варианта функции приспособленности:

- на основе строгого сравнения (*strict*);
- на основе вычисления расстояния Хэмминга (*hamming*);
- на основе вычисления редакционного расстояния (расстояния Левенштейна) (*edit*).

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

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

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

меньшее значение соответствует большему соответствию выходной строки эталону.

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

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

1.2. Программные средства для решения задачи о выполнимости булевой формулы и их применение

В настоящем разделе приводится аналитический обзор программных средств для решения задачи о выполнимости булевой формулы (в дальнейшем такое средство будем также называть *SAT-solver*) Будут рассмотрены наиболее известные программные средства, выпущенные за последнее десятилетие.

Все программные средства объединяет то, что булева формула должна быть представлена в конъюнктивной нормальной форме и записана в файл в формате *DIMACS* [28].

1.2.1. Программное средство *zChaff*

Программное средство *zChaff* [32] является реализацией алгоритма *Chaff* [26]. На соревновании *SAT 2002 Competition* программное средство *zChaff* был признан лучшим по нескольким категориям. Известны случаи успешного решения им промышленных задач, включающих миллион переменных и десятки миллионов ограничений на переменные.

Программное средство *zChaff* можно скомпилировать в подключаемую библиотеку, что позволяет использовать *SAT-solver* без создания дополнительных файлов. Средство успешно интегрировано в такие продукты, как верификатор моделей *NuSMV* [27] и доказатель теорем *GrAnDe* [17].

Как и большинство алгоритмов, используемых для решения задачи о выполнимости булевой формулы, алгоритм *Chaff* является частным случаем алгоритма *DPLL* (*Davis-Putnam-Logemann-Loveland*) [14]. Опишем схему работы данного алгоритма.

Функции *DPLL* подается на вход формула *f*. Если формула является тривиальной, то функция возвращает, разрешима (TRUE) или не разрешима (FALSE) данная формула *f* (имеется ли выполняющая подстановка). Если формула *f* не является тривиальной, то некоторым образом выбирается переменная *v* и рекурсивно запускается функция *DPLL* с допущением *v* = TRUE. Если при данном допущении формула *f* имеет выполняющую подстановку, то возвращаем TRUE – формула разрешима. В противном случае повторяем процесс с допущением *v* = FALSE. Если ни одно из допущений не привело к нахождению выполняющей подстановки, то возвращаем FALSE - формула неразрешима. Иллюстрация работы данного алгоритма приведена на рис. 6. Иллюстрация взята из работы [30].

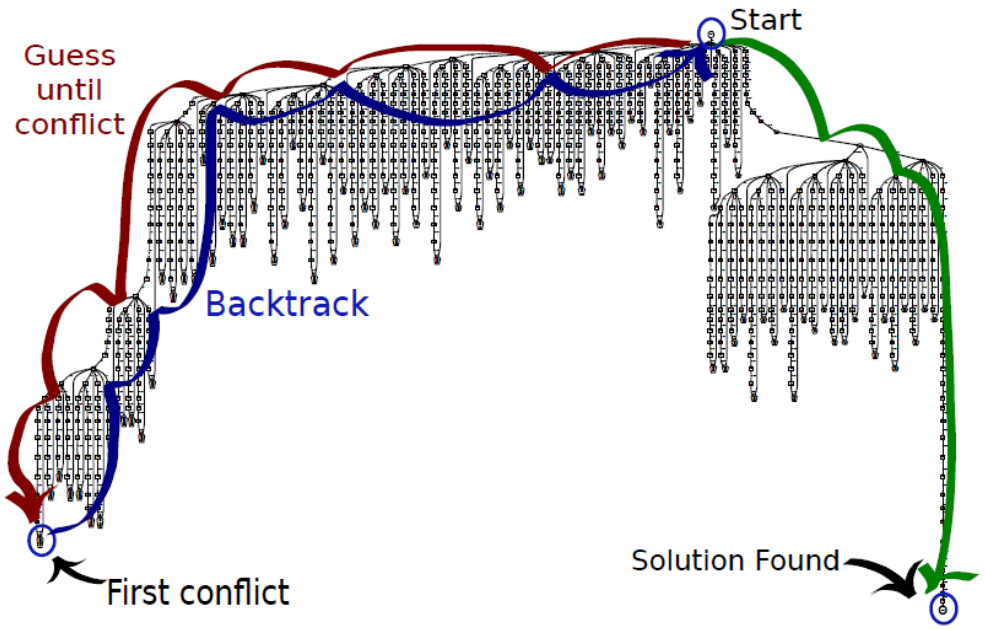

Рис. 6. Иллюстрация работы алгоритма *DPLL*

1.2.2. Программное средство *MiniSat*

MiniSat [15] – одно из самых популярных программных средств для решения задачи о выполнимости булевой формулы. Данный *SAT-solver* популярен за счет того, что авторы выполнили поставленную перед собой задачу – создать средство с малым числом строк исходного кода, который будет понятен начинающим исследователям и разработчикам.

На соревновании *SAT 2005 Competition MiniSat* стал победителем в четырех категориях, что позволяет говорить о высокой производительности данного программного средства. На соревнованиях последующих лет средство *MiniSat* почти всегда попадало в пятерку лучших средств для решения задачи о выполнимости булевой формулы.

Особенностью данного программного средства является использование препроцессора *SatELite* [11], упрощающего исходную задачу за счет удаления некоторых переменных и ограничений на них. На рис. 7 приведены графики, демонстрирующие на двух тестовых наборах улучшение производительности программного средства *MiniSat* за счет использования препроцессора. Графики взяты из работы [11].

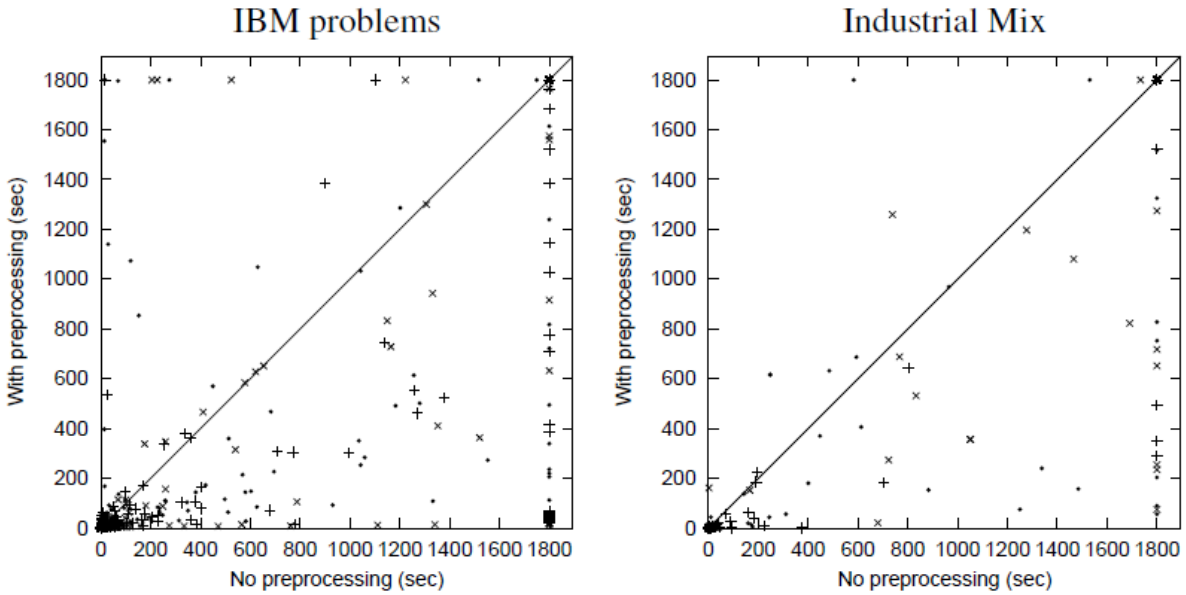

Рис. 7. Сравнение производительности средства *MiniSat* с использованием и без использования препроцессора *SatELite* на двух тестовых наборах

1.2.3. Программное средство *CryptoMiniSat*

Программным средством для решения задачи о выполнимости булевой формулы с самой высокой производительностью на данный момент является *CryptoMiniSat* [13]. Данное средство победило в главной категории (*main track*) соревнования *SAT-Race 2010*. Основной целью автора являлась разработка программного средства, подбирающего код для некоторого шифрования. Цель была достигнута – два дня работы *CryptoMiniSat* был подобран ключ для программатора *HiTag2*, используемого в современных системах безопасности автомобилей.

В качестве ядра программы было использовано ранее описанное средство *MiniSat*, содержащее примерно 1500 строк исходного кода, в то время как число строк исходного кода *CryptoMiniSat* достигает 13000. Код является открытым – распространяется по лицензии *GPL*.

Отличительной особенностью *CryptoMiniSat* является то, что он может обрабатывать не только ограничения, записанные через бинарную операцию «или», но и через бинарную операцию «исключающее или» (сложение по модулю два, XOR).

В дальнейшем предполагается использование именно этого программного средства.

1.3. Постановка задачи

В настоящей работе *управляющим конечным автоматом* будем называть детерминированный конечный автомат, каждый переход которого помечен *событием*, последовательностью *выходных воздействий* и *охранным условием*, представляющей собой логическую формулу от *входных переменных*.

Автомат получает события от так называемых *поставщиков событий* (в их роли могут выступать внешняя среда, интерфейс пользователя и т. д.) и генерирует выходные воздействия для *объекта управления*. При поступлении события автомат выполняет переход в соответствии с охранными условиями и значениями входных переменных. При выполнении перехода генерируются выходные воздействия, которыми он помечен, и автомат переходит в соответствующее состояние. Отметим, что состояния такого автомата не делятся на допускающие и не допускающие. Пример управляющего автомата приведен на рис. 8.

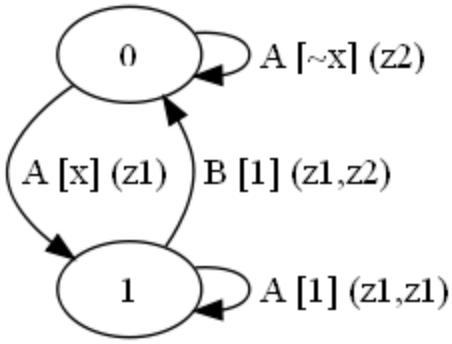

Рис. 8. Пример управляющего автомата

Для данного автомата множество входных событий равно {*A*, *B*}, охранные условия зависят от единственной логической входной переменной *x*, множество выходных воздействий равно {*z*1, *z*2}. Далее, состояние автомата с номером 0 будем считать начальным.

В настоящей работе в качестве исходных данных для построения управляющего конечного автомата используется множество *сценариев работы*. Сценарием работы будем называть последовательность $T_1 \ldots T_n$ троек $T_i = \langle e_i, f_i, A_i \rangle$, где e_i – входное событие, f_i – булева формула от входных переменных, задающая охранное условие, A_i – последовательность выходных воздействий. В дальнейшем тройки T_i будем называть *элементами сценария*.

Будем говорить, что автомат, находясь в состоянии *state*, *удовлетворяет элементу сценария* T_i, если из *state* исходит переход, помеченный событием e_i, последовательностью выходных воздействий A_i и охранным условием, тождественно равным f_i как булева формула. Автомат *удовлетворяет сценарию работы* $T_1 \ldots T_n$, если он удовлетворяет каждому элементу данного сценария, находясь при этом в состояниях пути, образованного соответствующими переходами.

Например, автомат, приведенный на рис. 8, удовлетворяет сценарию работы <*A*, ~*x*, (z2)>, <*A*, *x*, (z1)>, но не удовлетворяет сценариям работы:

- <*B*, 1, (*z*2)>, так как из начального состояния не исходит перехода, помеченного событием *B*;
- <*A*, *x*, (*z*1)>, <*A*, *x*, (*z*1,*z*1)>, так как из состояния номер 1, при значении $x = 1$ выполнится переход, помеченный охранным условием 1, а не *x*;
- <*A*, *x*, (*z*2)>, так как из начального состояния выполнится переход, помеченный последовательностью выходных воздействий (*z*1), а не (*z*2).

В настоящей работе решается задача построения управляющего конечного автомата с заданным числом состояний *C* по заданному множеству сценариев работы *Sc*, которым автомат должен удовлетворять.

Выводы по главе 1

1. Обзор методов машинного обучения для построения автоматов-распознавателей позволяет утверждать, что целесообразно применение методов решения задачи о выполнимости булевой формулы.
2. Обзор программных средств для решения задачи о выполнимости булевой формулы показал, что наилучшим по производительности средством на данный момент является *CryptoMiniSat*.
3. Была поставлена задача машинного обучения, решаемая в настоящей работе с помощью применения методов решения задачи о выполнимости булевой формулы.

2. ТЕОРЕТИЧЕСКАЯ ОЦЕНКА СЛОЖНОСТИ ПОСТРОЕНИЯ УПРАВЛЯЮЩЕГО АВТОМАТА ПО СЦЕНАРИЯМ РАБОТЫ ПРОГРАММЫ

В настоящем разделе приводится теоретическая оценка сложности построения автомата по сценариям работы программы – доказывается *NP*-полнота поставленной задачи.

Рассматриваемой задаче соответствует язык L – множество пар $<Sc, C>$ (здесь Sc – набор сценариев работы программы, C – натуральное число, записанное в унарной системе счисления), для которых существует автомат A, удовлетворяющий сценариям из Sc и число состояний которого равно C.

Согласно книге [8] для доказательства *NP*-полноты необходимо доказать принадлежность задачи классу *NP* и классу *NP*-трудных задач.

2.1. Доказательство принадлежности классу *NP* поставленной задачи

Докажем принадлежность классу *NP* задачи построения управляющего автомата по сценариям работы. Для доказательства воспользуемся определением класса *NP* на языке сертификатов [8]. Говорят, что y является *сертификатом* принадлежности элемента x языку L, если существует полиномиальное отношение (верификатор) R, такое, что $R(x, y) = 1$ тогда и только тогда, когда x принадлежит L.

Для исследуемой задачи сертификатом y для элемента $x = <Sc, C>$ является управляющий автомат, удовлетворяющий всем сценариям из Sc и число состояний которого равно C. Таким образом, верификатором $R(x, y)$ является программа, которая проверяет два утверждения.

Во-первых, для каждого сценария из множества Sc программа R проверяет, удовлетворяет ли ему автомат y. Следуя определению, введенному ранее, для каждого элемента сценария проверка,

удовлетворяет ли элементу автомат, производится за $O(|y|)$ – в худшем случае будут рассмотрены все переходы автомата. Таким образом, для всех заданных сценариев время работы данного шага составит $O(|Sc|\cdot|y|)$, где как $|Sc|$ обозначена суммарная длина всех сценариев набора Sc.

Во-вторых, программа R проверяет то, что автомат состоит ровно из C состояний. В начале доказательства было оговорено, что число C задано в унарной системе счисления. Если бы это было иначе, то размер любого элемента x с пустым множеством Sc составлял бы $O(log\ |y|)$. Таким образом, размер сертификата y зависел бы экспоненциально от размера элемента x, что не позволило бы работать верификатору R полиномиальное время.

Таким образом, было показано существование полиномиального отношения R, что доказывает принадлежность классу NP исследуемой задачи.

2.2. Доказательство принадлежности классу NP-трудных задач

Докажем принадлежность рассматриваемой задачи классу NP-трудных задач. Для этого докажем, что к поставленной задаче можно свести задачу из класса NP-трудных. В качестве такой задачи выберем задачу построения автомата-распознавателя с заданным числом состояний по заданному набору слов, которые автомат должен распознавать. Согласно работе [16] данная задача является NP-полной, а, следовательно, и NP-трудной.

Языком M данной задачи является множество троек $<S^+, S^-, C>$ таких, что существует конечный автомат-распознаватель, который принимает слова из словаря S^+, не принимает слова из словаря S^- и число его состояний равно C.

Согласно определению сведения по Карпу [8], язык M сводится к языку L, если существует такая функция $f(w)$, что w принадлежит M тогда и только тогда, когда элемент $f(w)$ принадлежит L.

Опишем сводящую функцию f, принимающую на вход элемент $w = <S^+, S^-, C>$ языка M и возвращающую элемент $f(w) = <Sc, C>$ языка L рассматриваемой в работе задачи. Множество сценариев Sc составляется следующим образом. Для каждого слова $a_1 \ldots a_n$ из словаря S^+ во множество Sc добавляется сценарий работы

$$<a_1, 1, ()>, \ldots, <a_n, 1, ()>, <\$, 1, (accept)>,$$

где как «$» обозначен символ конца строки. Аналогично, для каждого слова из словаря S^- добавляется сценарий работы, последний элемент которого равен <$, 1 (*reject*)>.

Докажем, что если $w = <S^+, S^-, C>$ принадлежит M, то $f(w) = <Sc, C>$ принадлежит языку L − если существует автомат-распознаватель A для элемента w, то можно построить управляющий автомат для $f(w)$. Для этого преобразуем автомат A следующим образом. Во-первых, каждый переход автомата-распознавателя по символу c преобразуем в переход управляющего автомата, помеченный тройкой $<a, 1, ()>$ − событием a, тождественным охранным условием и пустой последовательностью выходных воздействий. Во-вторых, из каждого допускающего состояния добавим переход, помеченный тройкой <$, 1, (*accept*)> и ведущий в это же состояние. При этом, разумеется, допускающие состояния преобразуются в обычные состояния управляющего автомата. В-третьих, для остальных состояний автомата аналогичным образом добавим переход, помеченный тройкой <$, 1 (*reject*)>. Таким образом, легко видеть, что построенный управляющий автомат удовлетворяет сценариям из Sc и число его состояний равно C.

Докажем обратное. Если $f(w)$ принадлежит языку L, то w принадлежит языку M – если существует управляющий автомат A',

удовлетворяющий построенным сценариям из множества *Sc* и число его состояний равно *C*, то существует и автомат-распознаватель для элемента *w*. Преобразуем автомат *A'* в искомый автомат-распознаватель. Для этого выполним следующие действия:

- все переходы автомата, помеченные тройкой <*a*, 1, ()>, преобразуем в переходы по символу *a;*
- переходы, помеченные <$, 1, (*accept*)> удалим, при этом каждое состояние, из которого вел такой переход, сделаем допускающим;
- удалим остальные переходы.

Таким образом, полученный автомат-распознаватель по построению принимает слова из S^+, не принимает слова из S^- и число его состояний равно *C*.

Следовательно, доказано, что существует функция *f*(*w*) такая, что *w* принадлежит *M* тогда и только тогда, когда элемент *f*(*w*) принадлежит *L*. Следовательно, язык *M* сводится к языку *L*, что доказывает принадлежность задачи построения управляющего автомата классу *NP*-трудных задач.

Выводы по главе 2

1. Доказана принадлежность классу *NP* задачи построения управляющего автомата по сценариям работы.
2. Доказана принадлежность классу *NP*-трудных задач рассматриваемой задачи.
3. Приведена теоретическая оценка сложности рассматриваемой задачи – доказана ее *NP*-полнота.

3. ПРЕДЛАГАЕМЫЙ МЕТОД МАШИННОГО ОБУЧЕНИЯ

Построение управляющего автомата осуществляется в пять этапов.

1. Построение дерева сценариев.
2. Построение графа совместимости вершин дерева сценариев.
3. Построение булевой КНФ-формулы, задающей требования к раскраске построенного графа и выражающей непротиворечивость системы переходов результирующего автомата.
4. Запуск сторонней программы, решающей задачу о выполнимости булевой КНФ-формулы.
5. Построение автомата по найденному выполняющему набору значений переменных.

3.1. Построение дерева сценариев

Деревом сценариев назовем дерево, каждый переход которого помечен событием, булевой формулой и последовательностью выходных воздействий. Опишем алгоритм построения дерева сценариев по заданному множеству сценариев *Sc*.

Сначала дерево сценариев состоит из единственной вершины – корня дерева. Затем по очереди добавим в дерево все сценарии работы из *Sc*.

Для каждого из сценариев будем добавлять его элементы в дерево в порядке возрастания их номеров, начиная с первого. При этом будем хранить указатель на текущую вершину дерева v и номер i первого необработанного элемента сценария.

В начале процесса добавления v указывает на корень дерева сценариев, а $i = 1$. На каждом шаге проверяется существование исходящего из вершины v ребра, помеченного событием e_i и логической формулой, совпадающей с f_i как булевой функции. Если такое ребро не существует, то создается новая вершина дерева u, и в нее направляется ребро, помеченное

тройкой $<e_i, f_i, A_i>$. После этого u становится текущей вершиной, а значение i увеличивается на единицу.

Если такое ребро существует, то производится сравнение последовательности A_i и последовательности выходных воздействий A', которой помечено рассматриваемое ребро. Если $A_i = A'$, то текущей становится вершина, в которую ведет рассматриваемое ребро дерева, а значение i увеличивается на единицу.

Если же указанные последовательности не совпадают, то заданное множество сценариев Sc является противоречивым, поэтому работа алгоритма прерывается, и пользователю выводится соответствующее сообщение.

После завершения добавления всех сценариев в дерево производится проверка охранных условий. Для каждой вершины перебираются все пары исходящих из нее ребер. Если существует такая пара ребер, что они помечены одним и тем же событием, а их охранные условия имеют общий выполняющий набор значений входных переменных, то множество сценариев предполагает недетерминированное поведение. Поэтому работа алгоритма прерывается, и пользователю выводится соответствующее сообщение.

На рис. 9 приведен пример дерева сценариев, построенного по сценариям:

<*A*, ~*x*, (*z*2)>, <*A*, *x*, (*z*1)>, <*A*, 1, (*z*1,*z*1)>;

<*A*, *x*, (*z*1)>, <*B*, 1, (*z*1,*z*2)>, <*A*, *x*, (*z*1)>;

<*A*, *x*, (*z*1)>, <*B*, 1, (*z*1,*z*2)>, <*A*, ~*x*, (*z*2)>, <*A*, ~*x*, (*z*2)>.

Данным сценариям удовлетворяет автомат, приведенный на рис. 8.

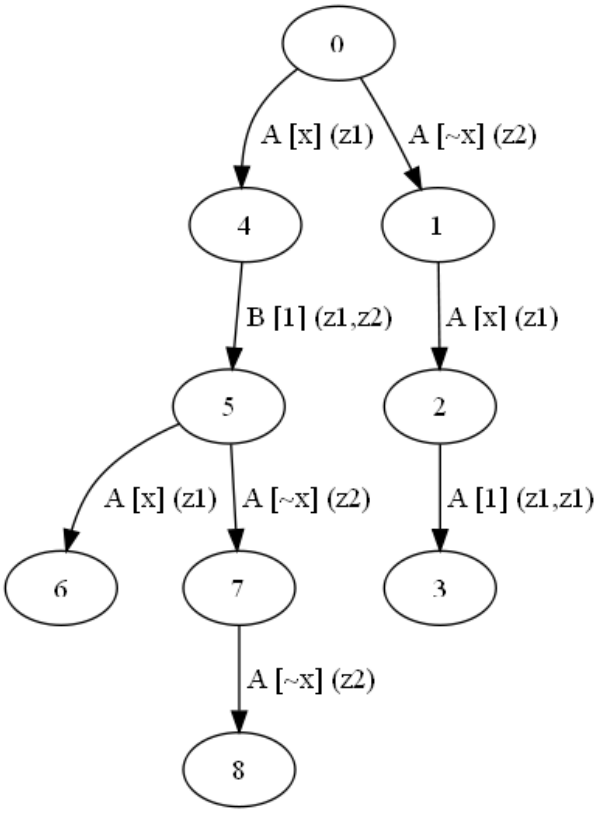

Рис. 9. Пример дерева сценариев

3.2. Построение графа совместимости дерева сценариев

Для построения управляющего автомата необходимо «раскрасить» вершины дерева сценариев в заданное число цветов (равное числу состояний, которое задается в качестве одного из параметров алгоритма). При этом вершины одного цвета будут объединены в одно состояние результирующего автомата, а множество исходящих из состояния переходов будет строиться из объединения множеств ребер исходящих из вершин заданного цвета.

Для задания ограничений на раскраску построим так называемый *граф совместимости* вершин дерева сценариев. Множество вершин этого графа совпадает с множеством вершин дерева сценариев, поэтому в дальнейшем вершины графа и дерева различаться не будут. Ребра графа определяются следующим образом.

Вершины графа совместимости u и v соединены ребром (далее такие вершины будем называть *несовместимыми*), если существует

последовательность пар $<e_1, values_1>\ldots<e_k, values_k>$ событий и наборов значений входных переменных, которая различает соответствующие вершины дерева. Будем говорить, что указанная последовательность *различает вершины* u и v, если выполняется совокупность следующих условий:

- из вершины u существует путь P_u, ребра которого помечены соответственно событиями $e_1\ldots e_k$ и такими охранными условиями $f_1\ldots f_k$, что наборы значений входных переменных $values_1\ldots values_k$ являются, соответственно, их выполняющими подстановками;
- аналогичный путь P_v существует из вершины v;
- для последних ребер путей P_u и P_v верно хотя бы одно из двух условий:

— пометки этих ребер различаются в части выходных воздействий;

— у охранных условий этих ребер есть общий выполняющий набор значений входных переменных, но они не совпадают как булевы функции.

Опишем алгоритм построения графа совместимости. Напомним, что множество вершин этого графа совпадает с множеством вершин дерева сценариев. Основной идеей данного алгоритма является метод динамического программирования [2].

Для каждой вершины дерева сценариев v найдем все несовместимые с ней вершины. Обозначим $S(v)$ множество вершин, несовместимых с v. Будем вычислять значения функции $S(v)$ начиная с листьев дерева сценариев. Для каждого из листьев u множество $S(u)$ пусто по определению несовместимых вершин.

Покажем, как вычислить значение $S(v)$, если оно уже вычислено для всех значений «детей» вершины v. Переберем все вершины дерева – вершина u входит в множество $S(v)$, если существует пара ребер ux (помечено событием e, формулой f_1 и последовательностью действий A_1) и

vy (помечено также событием *e*, формулой f_2 и последовательностью действий A_2) такая, что выполняется одно из трех:

- формулы f_1 и f_2 имеют общий выполняющий набор значений входных переменных, но не совпадают, как булевы функции. Тогда <*e*, *values*>, где как *values* обозначена выполняющая подстановка f_1, – последовательность, различающая *u* и *v*;
- формулы f_1 и f_2 совпадают, как булевы функции, а последовательности A_1 и A_2 не совпадают. Тогда вершины *u* и *v* различает такая же последовательность;
- формулы f_1 и f_2 совпадают, как булевы функции, и вершина *x* входит в множество *S*(*y*), посчитанное заранее. Тогда существует последовательность $<e_1, values_1>\ldots<e_k, values_k>$, различающая вершины *x* и *y*, а вершины *v* и *u* различает последовательность <*e*, *values*> $<e_1, values_1>\ldots<e_k, values_k>$.

Время работы этого алгоритма составляет $O(n^2)$ (где *n* – число вершин в дереве сценариев), так как каждая пара ребер дерева сценариев в процессе работы алгоритма будет рассмотрена не более одного раза. При этом такое время работы достижимо, если заранее для каждой пары формул вычислено, равны ли они как булевы функции и имеют ли общий выполняющий набор значений входных переменных.

В худшем случае время работы этапа обработки формул составляет $O(2^{2m}n^2)$, где за *m* обозначено максимальное число входных переменных, использующихся в одном охранном условии. На практике число *m* не превышает четырех.

На рис. 10 приведен граф совместимости для дерева сценариев, приведенного на рис. 9.

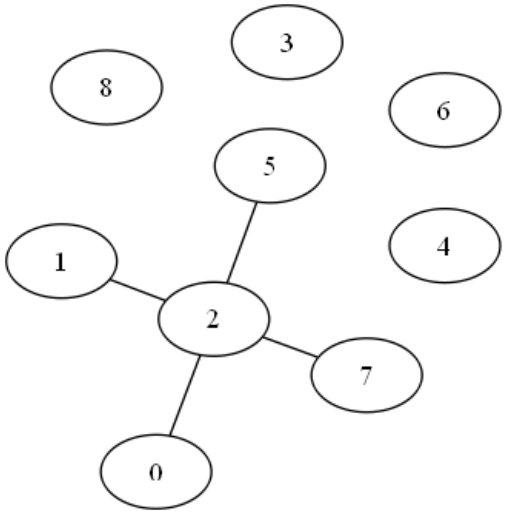

Рис. 10. Пример графа совместимости

3.3. Построение булевой КНФ-формулы

Опишем алгоритм построения булевой КНФ-формулы, задающей требования к раскраске построенного графа и выражающей непротиворечивость системы переходов результирующего автомата. Данное построение аналогично построению КНФ-формулы, используемой в работе [19] для построения автомата-распознавателя.

Напомним, что в настоящей работе на вход алгоритму подается число C состояний результирующего автомата.

В этой формуле будут использоваться следующие логические переменные:

- $x_{v,i}$ (для каждой вершины дерева сценариев v и цвета вершины i – числа от 1 до C) – верно ли, что вершина v имеет цвет i. Напомним, что вершины одного цвета будут объединены в одно состояние результирующего автомата;
- $y_{a,b,e,f}$ (для каждой пары состояний результирующего автомата a и b, каждого события e, каждой формулы f, встречающейся в сценариях) – верно ли, что в результирующем автомате существует переход из состояния a в состояние b, помеченный событием e и формулой f.

КНФ-формула состоит из следующих дизъюнктов:

- $(x_{v,1} \vee ... \vee x_{v,C})$ (для каждой вершины v дерева сценариев) – накладываем условие существования цвета вершины v;
- $(\sim x_{v,i} \vee \sim x_{v,j})$ (для каждой вершины v дерева сценариев, цвета i и цвета j, где $i < j$) – накладываем условие, что вершина v не покрашена одновременно в цвета i и j;
- $(\sim x_{v,i} \vee \sim x_{u,i})$ (для каждой пары несовместимых вершин u и v дерева сценариев и цвета i) – накладываем ограничения на раскраску, задаваемые графом совместимости;
- $(\sim y_{a,b,e,f} \vee \sim y_{a,d,e,f})$ (для каждой тройки состояний результирующего автомата a, b и d ($b < d$), каждого события e, каждой формулы f) – из состояния a выходит не более одного ребра, помеченного событием e и формулой f;
- $(y_{a,b,e,f} \vee \sim x_{v,a} \vee \sim x_{u,b})$ (для каждого ребра vu дерева сценариев) – если выполняется:

— ребро из вершины дерева v в вершину u помечено событием e и формулой f;
— вершина v покрашена в цвет a;
— вершина u покрашена в цвет b;

то в результирующем автомате существует переход из состояния a в состояние b, помеченный событием e и формулой f;

- $(\sim y_{a,b,e,f} \vee \sim x_{v,a} \vee x_{u,b})$ (для каждого ребра vu дерева сценариев) – если выполняется:

— ребро из вершины дерева v в вершину дерева u помечено событием e и формулой f;
— вершина v покрашена в цвет a;
— в результирующем автомате существует переход из состояния a в состояние b, помеченный событием e и формулой f;

то вершина u покрашена в цвет b.

Таким образом, число дизъюнктов КНФ-формулы можно оценить как $O(|V|^2 \cdot C)$, где как $|V|$ обозначено число вершин дерева сценариев.

3.4. Запуск сторонней программы, решающей задачу о выполнимости булевой КНФ-формулы, и построение автомата по найденному выполняющему набору значений переменных

Для того чтобы найти выполняющий набор для построенной КНФ-формулы, воспользуемся одной из сторонних программ (*SAT-solver*), решающих задачу *SAT*. Обзор существующих программных средств, приведенный выше, показал, что наилучшим по производительности средством для этой цели на данный момент является *CryptoMiniSat*.

Подадим на вход выбранной программе построенную КНФ-формулу, записанную в формате *DIMACS* [28]. Если программа не обнаружила выполняющий набор значений переменных, то будем считать, что по данному набору сценариев невозможно построить управляющий автомат с заданным числом состояний.

Если же программа обнаружила выполняющий набор, то построим искомый автомат. Для этого на основании полученных значений переменных $x_{v,i}$ определим цвет каждой вершины дерева сценариев. На рис. 11 приведен пример раскраски дерева сценариев, приведенного на рис. 9.

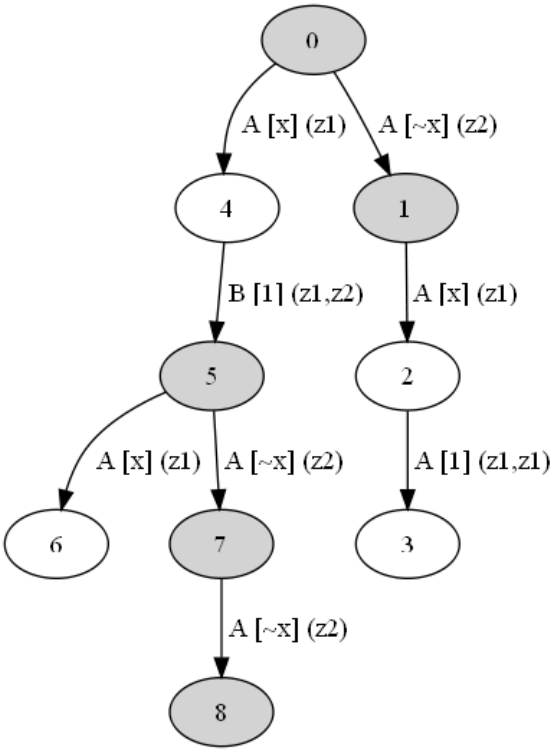

Рис. 11. Пример раскраски дерева сценариев в два цвета

После этого объединим все вершины одного цвета в одно состояние автомата, а начальным состоянием положим состояние, соответствующее цвету корня дерева сценариев. Множество исходящих из состояния переходов построим из объединения множеств ребер, исходящих из вершин заданного цвета.

Например, после объединения вершин дерева, приведенного на рис. 11, получим автомат, изображенный на рис. 12. Заметим, что данный автомат изоморфен автомату, приведенному на рис. 8. Отметим, что не исключено существование нескольких автоматов, удовлетворяющих сценариям множества *Sc*.

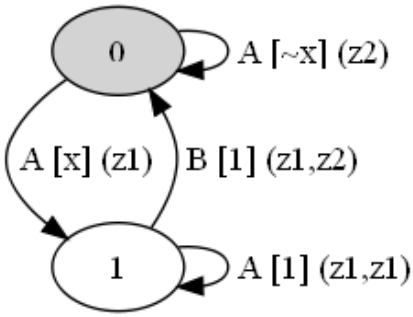

Рис. 12. Пример автомата, полученного после объединения вершин дерева

Выводы по главе 3

1. Предложен метод машинного обучения для построения управляющих автоматов по сценариям работы.
2. Этот метод основан на сведении указанной задачи к задаче о выполнимости булевой формулы.
3. Число дизъюнктов КНФ-формулы, подающейся на вход стороннему программному средству, можно оценить как $O(|V|^2 \cdot C)$.

4. ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ

Был проведен ряд экспериментальных исследований для оценки производительности предложенного алгоритма. Алгоритм сведения был реализован на языке *Java*.

4.1. Исследование на задаче о построении автомата управления часами с будильником

Было проведено исследование на задаче о построении автомата управления часами с будильником [6]. На данной задаче алгоритм работает менее одной секунды на компьютере с процессором *Intel Core 2 Quad Q9400*, в то время как генетический алгоритм работает порядка пяти минут. Это позволяет говорить о превосходстве предложенного метода над существующими.

4.2. Исследование на задачах, полученных случайным образом

Исследование осуществлялось в три этапа:

- Генерация случайного управляющего автомата с n состояниями.
- Генерация сценариев работы – случайных путей в полученном автомате.
- Запуск предложенного алгоритма на полученных сценариях работы и числе n, измерение времени работы программы.

Управляющие автоматы генерировались со следующими параметрами:

- Число входных событий равно двум.
- Число воздействий равно двум, наибольшее число воздействий на переходе равно двум. Таким образом, число различных последовательностей выходных воздействий, которыми может быть помечен переход, равно $2^3 - 1 = 7$.

- Охранные условия зависят от единственной входной переменной $x0$. Таким образом, число различных охранных условий равно трем.
- В управляющем автомате присутствует половина возможных ребер.

Пример сгенерированного управляющего автомата с пятью состояниями приведен на рис. 13.

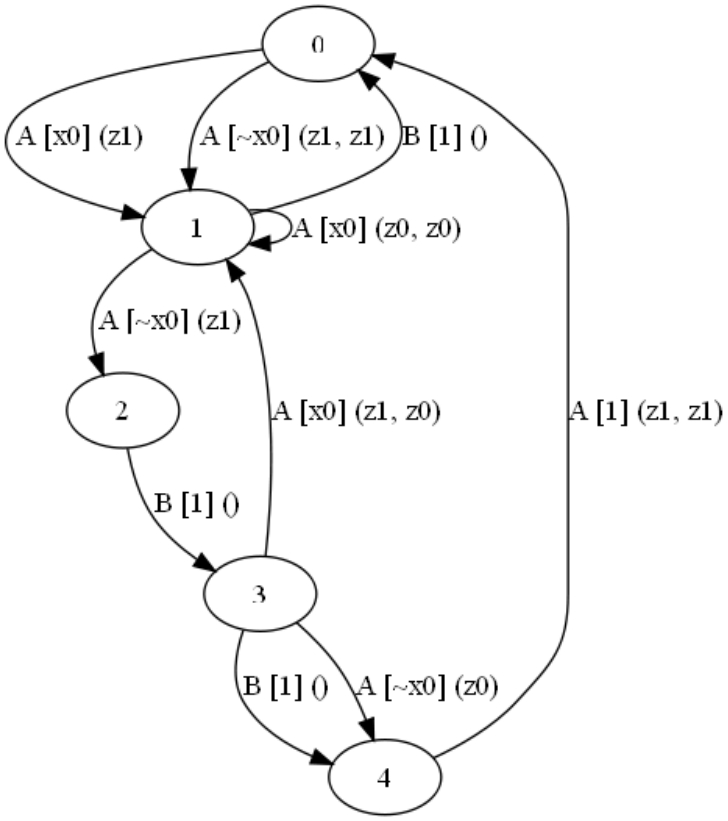

Рис. 13. Пример сгенерированного управляющего автомата

Рассматривалось по десять автоматов с 5, 10, 15 и 20 состояниями. По каждому сгенерированному автомату генерировалось восемь наборов сценариев работы с 20, 40, 60, 80, 100, 120, 140, 160 сценариями и суммарными длинами 250, 500, 750, 1000, 1250, 1500, 1750, 2000 соответственно. Длина каждого сценария работы не превышает 25.

Экспериментальные исследования проводились на компьютере с процессором *Intel Core 2 Quad Q9400*. Отметим, что программа использует на предложенных тестах до 3.5 гигабайт оперативной памяти, а число дизъюнктов в КНФ-формуле достигает пяти миллионов (при 25000 переменных). Результаты экспериментальных исследований приведены на

рис. 14. Точки кривой вычислялись как средние значения времени работы из десяти запусков на разных автоматах при данной суммарной длине сценариев работы.

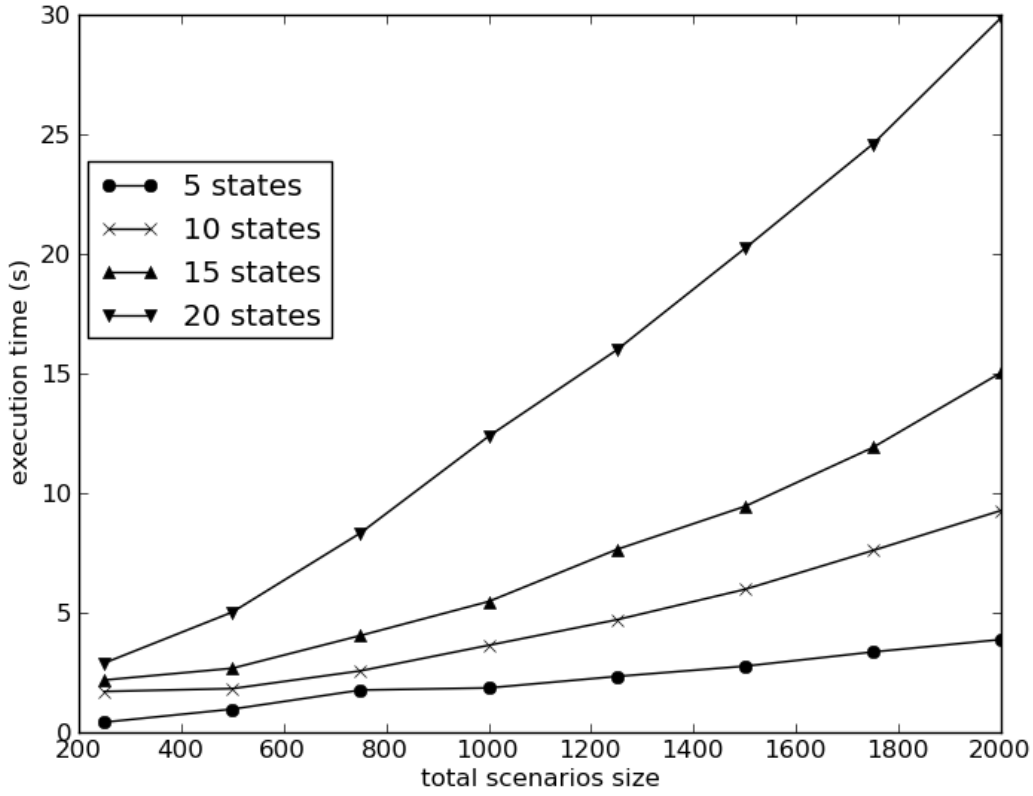

Рис. 14. Время работы программы на автоматах с 5, 10, 15 и 20 состояниями

Выводы по главе 4

1. Работоспособность метода проверена на сгенерированных вручную сценариях работы для задачи построения автомата управления часами с будильником. Производительность предложенного метода в сотни раз превышает производительность существующих методов.
2. Также было проведено детальное экспериментальное исследование на задачах, сгенерированных случайным образом. Предложенный метод позволяет строить на персональном компьютере управляющие автоматы, число состояний которых составляет несколько десятков, по сценариям работы с суммарной

длиной несколько тысяч единиц. Это позволяет говорить о высокой производительности предложенного метода.

ЗАКЛЮЧЕНИЕ

В настоящей работе предложен метод построения управляющих конечных автоматов по сценариям работы. Этот метод основан на сведении указанной задачи к задаче о выполнимости булевой формулы.

Работоспособность метода проверена на задачах как сгенерированных вручную, так и сгенерированных случайным образом. Результаты экспериментов позволяют говорить о высокой производительности предложенного метода по сравнению с существующими методами решения рассматриваемой задачи.

Была проведена теоретическая оценка сложности задачи – доказана ее принадлежность классу *NP*-полных задач.

ИСТОЧНИКИ

1. *Гладков Л. А., Курейчик В. В., Курейчик В. М.* Генетические алгоритмы. М.: Физматлит, 2006.
2. *Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К.* Алгоритмы. Построение и анализ. М.: Вильямс, 2010.
3. *Николенко С. И., Тулупьев А. Л.* Самообучающиеся системы. М.: МЦНМО, 2009.
4. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. СПб: Питер, 2009.
5. *Фридман А., Меннон П.* Теория и проектирование переключательных схем. М.: Мир, 1978.
6. *Царев Ф. Н.* Метод построения управляющих конечных автоматов на основе тестовых примеров с помощью генетического программирования // Информационно-управляющие системы. 2010. № 5, с. 31 – 36.
7. *Царев Ф. Н., Шалыто А. А.* Гибридное управление беспилотными летательными объектами на основе автоматного программирования /1-я Российская мультиконференция по проблемам управления. Сборник докладов четвертой научной конференции «Управление и информационные технологии». СПб ГЭТУ «ЛЭТИ». 2006, с. 138 – 144.
8. *Хопкрофт Дж., Мотвани Р., Ульман Дж.* Введение в теорию автоматов, языков и вычислений. М.: Вильямс, 2002.
9. *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
10. *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
11. *Biere A. Een N.* Effective Preprocessing in SAT through Variable and Clause Elimination / Proceedings of the SAT 2005. http://minisat.se/downloads/SatELite.pdf
12. *Chambers L.* Practical Handbook of Genetic Algorithms. Complex Coding Systems. Volume III. CRC Press, 1999.
13. CryptoMiniSat SAT-solver. http://www.msoos.org/cryptominisat2
14. *Davis M., Logemann G., Loveland D.* A machine program for theorem proving // Communications of the ACM. 1962 (5), pp. 394 – 397.

15. *Een N.*, *Sörensson N.* An extensible SAT-solver. Chalmers University of Technology. Sweden, 2003. http://minisat.se/downloads/MiniSat.pdf
16. *Gold E. M.* Complexity of automaton identification from given data. // Information and Control. 1978. 3, pp. 302 − 320.
17. GrAnDe (Ground And Decide) - system for CNF first order problems with a finite Herbrand universe. http://www.cs.miami.edu/~tptp/ATPSystems/GrAnDe/
18. *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
19. *Heule M.*, *Verwer S.* Exact DFA Identification Using SAT Solvers // Grammatical Inference: Theoretical Results and Applications 10th International Colloquium, ICGI 2010, pp. 66 – 79. Lecture Notes in Computer Science. Vol. 6339, Springer.
20. *Lang K.*, *Pearlmutter B.*, *Price R.* Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm / Grammatical Inference. 1998. Lecture Notes in Computer Science. Vol. 1433, pp. 1 – 12.
21. Learning DFS from Noisy Samples /A contest from GECCO 2004. http://cswww.essex.ac.uk/staff/sml/gecco/NoisyDFA.html
22. *Lucas S.*, *Reynolds J.* Learning DFA: Evolution versus Evidence Driven State Merging / The 2003 Congress on Evolutionary Computation (CEC '03). Vol. 1, pp. 351 – 358.
23. *Lucas S.*, *Reynolds J.* Learning Deterministic Finite Automata with a Smart State Labeling Algorithm // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2005. №7, pp. 1063–1074.
24. *Lucas S.* Evolving Finite-State Transducers: Some Initial Explorations. Lecture Notes in Computer Science. Springer Berlin / Heidelberg. Volume 2610. 2003, pp. 241 – 257. http://www.springerlink.com/content/41a34vg70fp1hltb/
25. *Mitchell M.*, *Holland J.*, *Forrest S.* When will a genetic algorithm outperform hill climbing? /Advances in Neural Information Processing Systems 6, pp. 51 – 58. California, 1994.
26. *Moskewicz M. W.*, *Madigan C. F.*, *Zhao Y.*, *Zhang L.*, *Malik S.* Chaff: engineering an efficient SAT solver /Design Automation Conference. 2001, pp. 530 – 535.
27. NuSMV: A new symbolic model checker. http://nusmv.fbk.eu/
28. Satisfiability suggested format DIMACS http://www.domagoj-babic.com/uploads/ResearchProjects/Spear/dimacs-cnf.pdf
29. *Spichakova M.* Genetic Inference of Finite State Machines. Masters thesis. Tallinn, 2007. http://s-ma-u-g.googlecode.com/files/thesis.pdf

30. *Soos M.* Breaking Industrial Ciphers at a Whim / Presentation at HES'11 http://www.msoos.org/wordpress/wp-content/uploads/2011/04/hes2011.pdf
31. *Tomita M.* Dynamic construction of finite automata from examples using hill climbing / Proceedings of the 4th Annual Cognitive Science Conference (USA, 1982), pp. 105 – 108.
32. *zChaff SAT-solver*. http://www.princeton.edu/~chaff/zchaff.html